NIST SPECIAL PUBLICATION 1800-15

Securing Small-Business and Home Internet of Things (IoT) Devices:

Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)

Includes Executive Summary (A); Approach, Architecture, and Security Characteristics (B); and How-To Guides (C)

Donna Dodson*
Douglas Montgomery
Tim Polk
Mudumbai Ranganathan
Murugiah Souppaya
NIST

Steve Johnson
Ashwini Kadam
Craig Pratt
Darshak Thakore
Mark Walker
CableLabs

Eliot Lear
Brian Weis
Cisco

*Former NIST Employee

William C. Barker
Dakota Consulting

Dean Coclin
Avesta Hojjati
Clint Wilson
DigiCert

Tim Jones
Forescout

Adnan Baykal
Global Cyber Alliance

Drew Cohen
Kevin Yeich
MasterPeace Solutions, Ltd.

Yemi Fashina
Parisa Grayeli
Joshua Harrington
Joshua Klosterman
Blaine Mulugeta
Susan Symington
The MITRE Corporation

Jaideep Singh
Molex

DRAFT

This publication is available free of charge from:
https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos

National Institute of Standards and Technology
U.S. Department of Commerce

NCCoE
NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

NIST SPECIAL PUBLICATION 1800-15

Securing Small-Business and Home Internet of Things (IoT) Devices: Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)

Includes Executive Summary (A); Approach, Architecture, and Security Characteristics (B); and How-To Guides (C)

Donna Dodson*
Douglas Montgomery
Tim Polk
Mudumbai Ranganathan
Murugiah Souppaya
NIST

Steve Johnson
Ashwini Kadam
Craig Pratt
Darshak Thakore
Mark Walker
CableLabs

Eliot Lear
Brian Weis
Cisco

William C. Barker
Dakota Consulting

Dean Coclin
Avesta Hojjati
Clint Wilson
DigiCert

Tim Jones
ForeScout

Adnan Baykal
Global Cyber Alliance

Drew Cohen
Kevin Yeich
MasterPeace Solutions, Ltd.

Yemi Fashina
Parisa Grayeli
Joshua Harrington

Joshua Klosterman
Blaine Mulugeta
Susan Symington
The MITRE Corporation

Jaideep Singh
Molex

*Former NIST Employee

DRAFT
September 2020

U.S. Department of Commerce
Wilbur Ross, Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

NIST SPECIAL PUBLICATION 1800-15A

Securing Small-Business and Home Internet of Things (IoT) Devices

Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)

Volume A:
Executive Summary

Donna Dodson*
Tim Polk
Murugiah Souppaya
NIST

William C. Barker
Dakota Consulting

Parisa Grayeli
Susan Symington
The MITRE Corporation

*Former NIST Employee

September 2020

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos

Executive Summary

WHY WE WROTE THIS GUIDE

Gartner predicts there will be 25 billion Internet of Things (IoT) devices by 2021. While such rapid growth has the potential to provide many benefits, it is also a cause for concern because IoT devices are tempting targets for attackers. State-of-the-art security software protects full-featured devices, such as laptops and phones, from most known threats, but many IoT devices, such as connected thermostats, security cameras, and lighting control systems, have minimal security or are unprotected. Because they are designed to be inexpensive and limited purpose, IoT devices may have unpatched software flaws. They also often have processing, timing, memory, and power constraints that make them challenging to secure. Users often do not know what IoT devices are on their networks and lack means for controlling access to them over their life cycles. However, the consequences of not addressing the security of IoT devices can be catastrophic. For instance, in typical networking environments, malicious actors can detect and attack an IoT device within minutes of it connecting to the internet. If it has a known vulnerability, this weakness can be exploited at scale, enabling an attacker to commandeer sets of compromised devices, called *botnets*, to launch large-scale distributed denial of service (DDoS) attacks, such as Mirai, as well as other network-based attacks. DDoS attacks can significantly harm an organization, rendering it impossible for the organization's customers to reach it and thereby resulting in revenue loss, potential liability exposure, reputation damage, and eroded customer trust.

CHALLENGE

Because IoT devices are designed to be low cost and for limited purposes, it is not realistic to try to solve the problem of IoT device vulnerability by requiring that all IoT devices be equipped with robust, state-of-the-art security mechanisms. Instead, we are challenged to develop ways to improve IoT device security without requiring costly or complicated improvements to the devices themselves.

A second challenge lies in the need to develop security mechanisms that will be effective even though IoT devices will, by their very nature, remain vulnerable to attack, and some will inevitably be compromised. These security mechanisms should protect the rest of the network from any devices that become compromised.

Given the widespread use of IoT devices by consumers who may not even be aware that the devices are accessing their network, a third challenge is the practical need for IoT security mechanisms to be easy to use. Ideally, security features should be so transparent that a user need not even be aware of their operation.

To address these challenges, the National Cybersecurity Center of Excellence (NCCoE) and its collaborators have demonstrated the practicality and effectiveness of using the Internet Engineering Task Force's Manufacturer Usage Description (MUD) standard to reduce both the vulnerability of IoT devices to network-based attacks and the potential for harm from any IoT devices that become compromised.

SOLUTION

The NCCoE and its collaborators have demonstrated how MUD can be deployed to strengthen security for IoT devices on home and small-business networks by helping prevent IoT devices from becoming both victims and perpetrators of network-based attacks. The solution outlined in this guide uses MUD to enable networks to automatically permit each IoT device to send and receive only the traffic it requires to perform its intended function and to prohibit all other communication with the device. By prohibiting unauthorized traffic to and from a device, the solution outlined in this guide both reduces the opportunity for an IoT device to be compromised by a network-based attack and reduces the ability of compromised devices to participate in network-based attacks such as DDoS campaigns. The NCCoE built four implementations of the MUD-based reference solution:

- Build 1 uses products from Cisco Systems to support MUD, from DigiCert to provide certificates, from Forescout to perform non-MUD-related discovery of devices, and from Molex to provide a MUD-capable IoT device.
- Build 2 uses products from MasterPeace Solutions, Ltd. to support MUD, perform non-MUD-related device discovery, and apply traffic rules to all devices based on a device's manufacturer and model. It uses certificates from DigiCert, and it integrates with services provided by Global Cyber Alliance and ThreatSTOP to prevent devices from connecting to domains that have been identified as potentially malicious based on current threat intelligence.
- Build 3 uses equipment supplied by CableLabs to support MUD. It leverages the Wi-Fi Easy Connect specification to securely onboard devices to the network and uses software-defined networking to create separate trust zones (e.g., network segments) to which devices can be assigned according to their intended network function. It also uses certificates from DigiCert.
- Build 4 uses DigiCert certificates and software developed by the National Institute of Standards and Technology's (NIST's) Advanced Networking Technologies Division as a working prototype that demonstrates feasibility and scalability of the MUD specification.

The NCCoE also developed this practice guide, which details the MUD-based reference solution and its four example implementations and maps the solution's capabilities to security controls specified in NIST Special Publication (SP) 800-53 and the NIST Cybersecurity Framework. This practice guide can help:

- organizations that rely on the internet to understand how MUD can be used to protect internet availability and performance against network-based attacks
- IoT device manufacturers see how MUD can protect against reputational damage resulting from their devices being exploited to support DDoS or other network-based attacks
- service providers benefit from reduced numbers of IoT devices that can be used to participate in DDoS attacks against their networks and degrade service for their customers
- users of IoT devices understand how MUD-capable products protect their internal networks and thereby help them avoid suffering increased costs and bandwidth saturation that could result from having their machines compromised and used to launch network-based attacks

While the NCCoE used a suite of technologies to address this challenge, this guide does not endorse any particular products, nor does it guarantee compliance with any regulatory initiatives. Your organization's information security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a solution.

HOW TO USE THIS GUIDE

This guide contains three volumes and a supplement:

- NIST SP 1800-15A: *Executive Summary–why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge* ***(you are here)***
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics–what we built and why, including the risk analysis performed and the security control map*
- NIST SP 1800-15C: *How-To Guides–instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project*
- Functional Demonstration Results—supplement to NIST SP 1800-15B: *describes the functional demonstration results for the four implementations of the MUD-based reference solution*

SUPPORTING RESOURCES

The supporting resources for this project include:

- Methodology for Characterizing Network Behavior of IoT Devices white paper—demonstrates how to use device characterization techniques to describe the communication requirements of IoT devices in support of the MUD specification
- NCCoE MUD-PD—a tool for characterizing IoT devices particularly for use with MUD and MUD file generation

SHARE YOUR FEEDBACK

You can view or download the guide and the supporting resources at https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos. Help the NCCoE make this guide better by sharing your thoughts with us as you read the guide. If you adopt this solution for your own organization, please share your experience and advice with us. We recognize that technical solutions alone will not fully enable the benefits of our solution, so we encourage organizations to share lessons learned and best practices for transforming the processes associated with implementing this guide.

To provide comments or to learn more by arranging a demonstration of this example implementation, contact the NCCoE at mitigating-iot-ddos-nccoe@nist.gov.

TECHNOLOGY PARTNERS/COLLABORATORS

Organizations participating in this project submitted their capabilities in response to an open call in the Federal Register for all sources of relevant security capabilities from academia and industry (vendors and integrators). The following respondents with relevant capabilities or product components (identified as “Technology Partners/Collaborators” herein) signed a Cooperative Research and Development Agreement (CRADA) to collaborate with NIST in a consortium to build this example solution.

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity challenges. Through this collaboration, the NCCoE develops modular, adaptable example cybersecurity solutions demonstrating how to apply standards and best practices using commercially available technology.

LEARN MORE

Visit https://www.nccoe.nist.gov
nccoe@nist.gov
301-975-0200

NIST SPECIAL PUBLICATION 1800-15B

Securing Small-Business and Home Internet of Things (IoT) Devices

Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)

Volume B:
Approach, Architecture, and Security Characteristics

Douglas Montgomery
Tim Polk
Mudumbai Ranganathan
Murugiah Souppaya
NIST

William C. Barker
Dakota Consulting

Drew Cohen
Kevin Yeich
MasterPeace Solutions, Ltd.

Steve Johnson
Ashwini Kadam
Craig Pratt
Darshak Thakore
Mark Walker
CableLabs

Eliot Lear
Brian Weis
Cisco

Dean Coclin
Avesta Hojjati
Clint Wilson
DigiCert

Tim Jones
Forescout

Adnan Baykal
Global Cyber Alliance

Yemi Fashina
Parisa Grayeli
Joshua Harrington
Joshua Klosterman
Blaine Mulugeta
Susan Symington
The MITRE Corporation

Jaideep Singh
Molex

September 2020

DRAFT

This publication is available free of charge from:
https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-15B, Natl. Inst. Stand. Technol. Spec. Publ. 1800-15B, 219 pages, (September 2020), CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback. As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: mitigating-iot-ddos-nccoe@nist.gov.

Public comment period: September 16, 2020 through October 16, 2020

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, easily adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align more easily with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

ABSTRACT

The goal of the Internet Engineering Task Force's Manufacturer Usage Description (MUD) specification is for Internet of Things (IoT) devices to behave as intended by the manufacturers of the devices. MUD provides a standard way for manufacturers to indicate the network communications that a device requires to perform its intended function. When MUD is used, the network will automatically permit the IoT device to send and receive only the traffic it requires to perform as intended, and the network will prohibit all other communication with the device, thereby increasing the device's resilience to network-based attacks. In this project, the NCCoE demonstrated the ability to ensure that when an IoT device connects to a home or small-business network, MUD can automatically permit the device to send and

receive only the traffic it requires to perform its intended function. This NIST Cybersecurity Practice Guide explains how MUD protocols and tools can reduce the vulnerability of IoT devices to botnets and other network-based threats as well as reduce the potential for harm from exploited IoT devices. It also shows IoT device developers and manufacturers, network equipment developers and manufacturers, and service providers who employ MUD-capable components how to integrate and use MUD to satisfy IoT users' security requirements.

KEYWORDS

access control; bootstrapping; botnets; firewall rules; flow rules; Internet of Things; IoT; Manufacturer Usage Description; MUD; network segmentation; onboarding; router; server; software update server; threat signaling; Wi-Fi Easy Connect.

DOCUMENT CONVENTIONS

The terms "shall" and "shall not" indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted.

The terms "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable without mentioning or excluding others or that a certain course of action is preferred but not necessarily required or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited.

The terms "may" and "need not" indicate a course of action permissible within the limits of the publication.

The terms "can" and "cannot" indicate a possibility and capability, whether material, physical, or causal.

Acronyms used in figures can be found in the Acronyms appendix.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory [ITL] draft publication). Such guidance and/or requirements may be directly stated in this ITL publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

1. assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

2. assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:
    a. under reasonable terms and conditions that are demonstrably free of any unfair discrimination or
    b. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to mitigating-iot-ddos-nccoe@nist.gov.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Allaukik Abhishek	Arm
Michael Bartling	Arm
Tao Wan	CableLabs
Russ Gyurek	Cisco
Peter Romness	Cisco
Rob Cantu	CTIA
Katherine Gronberg	Forescout

Name	Organization
Rae'-Mar Horne	MasterPeace Solutions, Ltd.
Nate Lesser	MasterPeace Solutions, Ltd.
Tom Martz	MasterPeace Solutions, Ltd.
Daniel Weller	MasterPeace Solutions, Ltd.
Nancy Correll	The MITRE Corporation
Sallie Edwards	The MITRE Corporation
Drew Keller	The MITRE Corporation
Sarah Kinling	The MITRE Corporation
Karri Meldorf	The MITRE Corporation
Mary Raguso	The MITRE Corporation
Allen Tan	The MITRE Corporation
Mo Alhroub	Molex
Bill Haag	National Institute of Standards and Technology
Paul Watrobski	National Institute of Standards and Technology
Bryan Dubois	Patton Electronics
Stephen Ochs	Patton Electronics

Name	Organization
Karen Scarfone	Scarfone Cybersecurity
Matt Boucher	Symantec
Petros Efstathopoulos	Symantec
Bruce McCorkendale	Symantec
Susanta Nanda	Symantec
Yun Shen	Symantec
Pierre-Antoine Vervier	Symantec
John Bambenek	ThreatSTOP
Russ Housley	Vigil Security

The Technology Partners/Collaborators who participated in this project submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build these example solutions. We worked with:

Technology Partner/Collaborator	Build Involvement
Arm	Subject matter expertise
CableLabs	Micronets Gateway Micronets cloud infrastructure Prototype IoT devices–Raspberry Pi with Wi-Fi Easy Connect support Micronets mobile application
Cisco	Cisco Catalyst 3850-S MUD manager

Technology Partner/Collaborator	Build Involvement
CTIA	Subject matter expertise
DigiCert	Private Transport Layer Security certificate Premium Certificate
Forescout	Forescout appliance–VCT-R Enterprise manager–VCEM-05
Global Cyber Alliance	Quad9 threat agent and Quad 9 MUD manager (integrated in Yikes! router) Quad9 domain name system Quad9 threat application programming interface ThreatSTOP threat MUD file server
MasterPeace Solutions, Ltd.	Yikes! router Yikes! cloud Yikes! mobile application
Molex	Molex light-emitting diode light bar Molex Power over Ethernet Gateway
Patton Electronics	Subject matter expertise
Symantec	Subject matter expertise

Contents

List of Figures

List of Tables

1 Summary

The [Manufacturer Usage Description Specification (Internet Engineering Task Force [IETF] Request for Comments [RFC] 8520)]() provides a means for increasing the likelihood that Internet of Things (IoT) devices will behave as intended by the manufacturers of the devices. This is done by providing a standard way for manufacturers to indicate the network communications that the device requires to perform its intended function. When the Manufacturer Usage Description (MUD) is used, the network will automatically permit the IoT device to send and receive only the traffic it requires to perform as intended, and the network will prohibit all other communication with the device, thereby increasing the device's resilience to network-based attacks. This project focuses on the use of IoT devices in home and small-business environments. Its objective is to show how MUD can practically and effectively reduce the vulnerability of IoT devices to network-based threats, and how MUD can limit the usefulness of any compromised IoT devices to malicious actors.

This volume describes a reference architecture that is designed to achieve the project's objective, the laboratory architecture employed for the demonstrations, and the security characteristics supported by the reference design. Four implementations of the reference design are demonstrated. These implementations are referred to as *builds*, and this volume describes all of them in detail:

- Build 1 uses products from Cisco Systems, DigiCert, Forescout, and Molex.
- Build 2 uses products from MasterPeace Solutions, Ltd.; Global Cyber Alliance (GCA); ThreatSTOP; and DigiCert.
- Build 3 uses products from CableLabs and DigiCert.
- Build 4 uses software developed at the National Institute of Standards and Technology (NIST) Advanced Networking Technologies laboratory and products from DigiCert.

The primary technical elements of this project include components that are designed and configured to support the MUD protocol. We describe these components as being *MUD-capable*. The components used include MUD-capable network gateways, routers, and switches that support wired and wireless network access; MUD managers; MUD file servers; MUD-capable Dynamic Host Configuration Protocol (DHCP) servers; update servers; threat-signaling servers; MUD-capable IoT devices; and MUD files and their corresponding signature files. We also used devices that are not capable of supporting the MUD protocol, which we call *non-MUD-capable* or *legacy* devices, to demonstrate the security benefits of the demonstrated approach that are independent of the MUD protocol, such as threat signaling and device onboarding. Non-MUD-capable devices used include laptops, phones, and IoT devices that cannot emit or otherwise convey a uniform resource locator (URL) for a MUD file as described in the MUD specification.

The demonstrated approach, which deploys MUD as an additional security tool rather than as a replacement for other security mechanisms, shows that MUD can make it more difficult to compromise

IoT devices on a home or small-business network by using a network-based attack. While MUD can be used to protect networks of any size, the scenarios examined by this National Cybersecurity Center of Excellence (NCCoE) project involve IoT devices being used in home and small-business networks. Owners of such networks cannot be assumed to have extensive network administration experience. This makes plug-and-play deployment a requirement. Although the focus of this project is on home and small-business network applications, the home and small-business network users are not the guide's intended audience. This guide is intended primarily for IoT device developers and manufacturers, network equipment developers and manufacturers, and service providers whose services may employ MUD-capable components. MUD-capable IoT devices and network equipment are not yet widely available, so home and small-business network owners are dependent on these groups to make it possible for them to obtain and benefit from MUD-capable equipment and associated services.

1.1 Challenge

The term *IoT* is often applied to the aggregate of single-purpose, internet-connected devices, such as thermostats, security monitors, lighting control systems, and connected television sets. The IoT is experiencing what some might describe as hypergrowth. Gartner forecasts that the number of IoT devices will reach 25 billion by 2021, while Forbes forecast that the market will exceed $457 billion before 2021. While such rapid growth has the potential to provide many benefits, it is also a cause for concern because IoT devices are tempting targets for attackers. State-of-the-art security software protects full-featured devices, such as laptops and phones, from most known threats, but many IoT devices, such as connected thermostats, security cameras, and lighting control systems, have minimal security or are unprotected. Because they are designed to be inexpensive and limited purpose, IoT devices may have unpatched software flaws. They also often have processing, timing, memory, and power constraints that make them challenging to secure. Users often do not know what IoT devices are on their networks and lack means for controlling access to them over their life cycles. However, the consequences of not addressing the security of IoT devices can be catastrophic. For instance, in typical networking environments, malicious actors can detect and attack an IoT device within minutes of it connecting to the internet. If it has a known vulnerability, this weakness can be exploited at scale, enabling an attacker to commandeer sets of compromised devices, called *botnets*, to launch large-scale distributed denial of service (DDoS) attacks, as well as other network-based attacks. A DDoS attack involves multiple computing devices in disparate locations sending repeated requests to a server with the intent to overload it and ultimately render it inaccessible. On October 12, 2016, a botnet consisting of more than 100,000 devices, called Mirai, launched a large DDoS attack on the internet infrastructure firm Dyn. Mirai interfered with Dyn's ability to provide domain name system (DNS) services to many large websites, effectively taking those websites offline for much of a day.

A DDoS or other network-based attack may result in substantial revenue losses and potential liability exposure, which can degrade a company's reputation and erode customer trust. Victims of a DDoS attack can include

- businesses that rely on the internet, who may suffer if their customers cannot reach them
- IoT device manufacturers, who may suffer reputational damage if their devices are exploited
- service providers, who may suffer service degradation that affects their customers
- users of IoT devices, who may suffer service degradation and potentially incur extra costs due to increased activity by their compromised machines

Because IoT devices are designed to be low cost and for limited purposes, it is not realistic to try to solve the problem of IoT device vulnerability by requiring that all IoT devices be equipped with robust state-of-the-art security mechanisms. Instead, we are challenged to develop ways to improve IoT device security without requiring costly or complicated improvements to the devices themselves. A second challenge lies in the need to develop security mechanisms that will be effective even though IoT devices will, by their very nature, remain vulnerable to attack, and some will inevitably be compromised. These security mechanisms should protect the rest of the network from any devices that become compromised. Given the widespread use of IoT devices by consumers who may not even be aware that the devices are accessing their network, a third challenge is the practical need that IoT security mechanisms be easy to use. Ideally, security features should be so transparent that a user need not even be aware of their operation. To address these challenges, the National Cybersecurity Center of Excellence (NCCoE) and its collaborators have demonstrated the practicality and effectiveness of using the Internet Engineering Task Force's Manufacturer Usage Description (MUD) standard to reduce both the vulnerability of IoT devices to network-based attacks and the potential for harm from any IoT devices that become compromised.

1.2 Solution

This project demonstrates how to use MUD to strengthen security when deploying IoT devices on home and small-business networks. The demonstrated approach uses MUD to constrain the communication abilities of MUD-capable IoT devices, thereby reducing the potential for these devices to be attacked as well as reducing the potential for them to be used to launch network-based attacks—both attacks that could be launched across the internet and attacks on the MUD-capable IoT device's local network. Using MUD combats IoT-based, network-based attacks by providing a standardized and automated method for making access control information available to network control devices capable of prohibiting unauthorized traffic to and from IoT devices. When MUD is used, the network will automatically permit the IoT device to send and receive the traffic it requires to perform as intended, and the network will prohibit all other communication with the device. Even if an IoT device becomes compromised, MUD prevents it from being used in any attack that would require the device to send traffic to an unauthorized destination.

In developing the demonstrated approach, the NCCoE sought existing technologies that use the MUD specification (RFC 8520). The NCCoE envisions using MUD as one of many possible tools that can be deployed, in accordance with best practices, to improve IoT security. This practice guide describes four implementations of the MUD specification that support MUD-capable IoT devices. It describes how Build 2 uses threat signaling to prevent both MUD-capable and non-MUD-capable IoT devices from connecting to internet locations that are known to be potentially malicious. It describes how Build 3 supports secure and automated onboarding of both MUD-capable and non-MUD-capable devices using the Wi-Fi Alliance's Wi-Fi Easy Connect protocol. It also describes the importance of using update servers to perform periodic updates to all IoT devices so that the devices will be protected with up-to-date software patches. It shows IoT device developers and manufacturers, network equipment developers and manufacturers, and service providers who employ MUD-capable components how to integrate and use MUD to help make home and small-business networks more secure.

1.3 Benefits

The demonstrated approach offers specific benefits to several classes of stakeholders:

- Organizations and others who rely on the internet, including businesses that rely on their customers being able to reach them over the internet, can understand how MUD can be used to protect internet availability and performance against network-based attacks.
- IoT device manufacturers can see how MUD can protect against reputational damage resulting from their devices being easily exploited to support DDoS or other network-based attacks.
- Service providers can benefit from a reduction in the number of IoT devices that malicious actors can use to participate in DDoS attacks against their networks and degrade service for their customers.
- Users of IoT devices, including small businesses and homeowners, can better understand what to ask for with respect to the set of tools available to protect their internal networks from being subverted by malicious actors. They will also better understand what they can expect regarding reducing their vulnerability to threats that can result from such subversion. By protecting their networks, they also avoid suffering increased costs and bandwidth saturation that could result from having their machines captured and used to launch network-based attacks.

2 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design and provides users with the information they need to replicate deployment of the MUD protocol to mitigate the threat of IoT devices being used to perform DDoS and other network-based attacks. This reference design is modular and can be deployed in whole or in part.

This guide contains three volumes and a supplement:

- NIST SP 1800-15A: *Executive Summary – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge*
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics – what we built and why, including the risk analysis performed, and the security control map* ***(you are here)***
- NIST SP 1800-15C: *How-To Guides – instructions for building the example implementations including all the security-relevant details that would allow you to replicate all or parts of this project*
- Functional Demonstration Results - supplement to NIST SP 1800-15B: *describes the functional demonstration results for the four implementations of the MUD-based reference solution*

It is intended for IoT device developers and manufacturers, network equipment developers and manufacturers, and service providers who employ MUD-capable components.

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary,* NIST SP 1800-15A, which describes the following topics:

- challenges that enterprises face in mitigating IoT-based DDoS threats
- example solutions built at the NCCoE
- benefits of adopting the example solutions

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in this part of the guide, NIST SP 1800-15B, which describes what we did and why. The following sections will be of particular interest:

- Section 3.4.3, Risk, provides a description of the risk analysis we performed.
- Section 5.2, Security Control Map, maps the security characteristics of this solution to cybersecurity standards and best practices.

You might share the *Executive Summary,* NIST SP 1800-15A, with your leadership team members to help them understand the importance of adopting standards-based mitigation of network-based distributed denial of service by using MUD protocols.

Information Technology (IT) professionals who want to implement an approach like this will find the whole practice guide useful. You can use the how-to portion of the guide, NIST SP 1800-15C, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solutions. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create each example solution.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial and open-source products to address this challenge, this guide does not endorse these particular products. Your organization can adopt one of these example solutions or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of the MUD protocol. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope you will seek products that are congruent with applicable standards and best practices. Section 5, Security Characteristic Analysis, maps the characteristics of the demonstrated approach to the cybersecurity controls provided by this reference solution.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but a possible solution. This is a draft guide. We seek feedback on its contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to mitigating-iot-ddos-nccoe@nist.gov.

2.1 Typographic Conventions

The following table presents typographic conventions used in this volume.

Typeface/ Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide.*
Bold	names of menus, options, command buttons, and fields	Choose **File > Edit.**

Typeface/ Symbol	Meaning	Example
`Monospace`	command-line input, onscreen computer output, sample code examples, and status codes	`Mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

3 Approach

The NCCoE issued an open invitation to technology providers to participate in demonstrating an approach to deploying IoT devices in home and small-business networks in a manner that provides higher security than is typically achieved in today's environments. In this project, the MUD specification (RFC 8520) is applied to home and small-business networks that are composed of both IoT and fully featured devices (e.g., personal computers and mobile devices). MUD constrains the communication abilities of MUD-capable IoT devices, thereby reducing the potential for these devices to be attacked as well as the potential for them to be used to launch attacks. Network gateway components and IoT devices leverage MUD to ensure that IoT devices send and receive only the traffic they require to perform their intended function. The resulting constraints on the MUD-capable IoT device's communication abilities reduce the potential for MUD-capable devices to be the victims of network-based attacks, as well as reduce the ability for these devices to be used in a DDoS or other network-based attack. In addition, in Build 2, we provide network-wide access controls based on threat signaling to protect legacy IoT devices, MUD-capable IoT devices, and fully featured devices (e.g., personal computers). In Build 3, the Wi-Fi Alliance's Wi-Fi Easy Connect protocol is used to securely onboard both MUD-capable and non-MUD-capable IoT devices that are Wi-Fi Easy Connect-capable. Automatic secure updates are also recommended for all devices.

The NCCoE prepared a Federal Register Notice inviting technology providers to provide products and/or expertise to compose prototypes. Components sought included MUD-capable routers or switches; MUD managers; MUD file servers; MUD-capable DHCP servers; IoT devices capable of emitting or otherwise conveying a MUD URL; and network access control based on threat signaling. Cooperative Research and

Development Agreements (CRADAs) were established with qualified respondents, and build teams were assembled. The build teams fleshed out the initial architectures, and the collaborators' components were composed into example implementations, i.e., builds. Each build team documented the architecture and design of its build. As each build progressed, its team documented the steps taken to install and configure each component of the build. The teams then conducted functional testing of the builds, including demonstrating the ability to retrieve a device's MUD file and use it to determine what traffic the device would be permitted to send and receive. We verified that attempts to perform prohibited communications would be blocked. Each team conducted a risk assessment and a security characteristic analysis and documented the results, including mapping the security contributions of the demonstrated approach to the *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and other relevant standards. Finally, the NCCoE worked with industry collaborators to suggest considerations for enhancing future support for MUD.

3.1 Audience

The focus of this project is on home and small-business deployments. Its solution is targeted to address the needs of home and small-business networks, which have users who cannot be assumed to have extensive network administration experience and who therefore require plug-and-play functionality. Although the focus of this project is on home and small-business network applications, we do not intend home and small-business network users to be this guide's primary audience. This guide is intended for the following types of organizations that provide products and services to homes and small businesses:

- IoT device developers and manufacturers
- network equipment developers and manufacturers
- service providers that employ MUD-capable components

3.2 Scope

The scope of this NCCoE project is IoT deployments in those home and small-business applications where plug-and-play deployment is required. The demonstrated approach includes MUD-capable IoT devices that interact with traditional computing devices, as permitted by their MUD files, and that also interact with external systems to access update servers and various cloud services. It employs both MUD-capable and non-MUD-capable IoT devices, such as connected lighting controllers, cameras, mobile phones, printers, baby monitors, digital video recorders, and connected assistants.

The primary focus of this project is on the technical feasibility of implementing MUD to mitigate network-based attacks. We show use of threat signaling to protect both MUD-capable devices and devices that are not MUD capable from known threats. We also show how Wi-Fi Easy Connect protocol can onboard both MUD-capable and non-MUD-capable devices, thereby securely providing each device with unique credentials for connecting to the network.

The reference architecture for the demonstrated approach includes support for automatic secure software updates. All builds include a server that is meant to represent an update server to which MUD will permit devices to connect. However, demonstrations of actual IoT device software updates and patching were not included in the scope of the project.

Providing security protections for each of the components deployed in the demonstrated approach is important. However, demonstrating these protections is outside the scope of this project. It is assumed that network owners deploying the architecture will implement best practices for securing it. Also, governance, operational, life cycle, cost, legal, and privacy issues are outside the project's current scope.

3.3 Assumptions

This project is guided by the following assumptions:

- IoT devices, by definition, are not general-purpose devices.
- Each IoT device has an intended function, and this function is specific enough that the device's communication requirements can be defined accurately and completely.
- An IoT device's communication should be limited to only what is required for the device to perform its function.
- Cost is a major factor affecting consumer purchasing decisions and consequent product development decisions. Therefore, it is assumed that IoT devices will not typically include organic support for all their own security needs and would therefore benefit from protections provided by an outside mechanism, such as MUD.
- IoT device manufacturers will use the MUD file mechanism to indicate the communications that each device needs.
- Network routers can be automatically configured to enforce these communications so that
    - intended communications are permitted
    - unintended communications are prohibited
- If all MUD-capable network components are deployed and functioning as intended, a malicious actor would need to compromise one of the systems with which an IoT device is permitted to communicate to launch a network-based attack on the device. If a device were to be compromised, it could be used in a network-based attack only against systems with which it is permitted to communicate.
- Network owners who want to provide the security protections demonstrated in this project will:
    - be able to acquire and deploy all necessary components of the architecture on their own network, including MUD-capable IoT devices, Wi-Fi Easy Connect-capable IoT devices, a MUD manager, a MUD-capable gateway/router/switch, a threat-signaling-

capable gateway/router/switch, a Wi-Fi Easy Connect-capable gateway, and a mobile application or other mechanism for scanning the quick response (QR) code of a Wi-Fi Easy Connect-capable device

  - have access to MUD file servers that host the MUD files for their IoT devices, update servers, threat-signaling servers, and current threat intelligence

- All deployed architecture components are secure and can be depended upon to perform as designed.
- Best practices for administrative access and security updates will be implemented, and these will reduce the success rate of compromise attempts.

3.4 Risk Assessment

NIST SP 800-30 Revision 1, *Guide for Conducting Risk Assessments*, states that risk is "a measure of the extent to which an entity is threatened by a potential circumstance or event, and typically a function of: (i) the adverse impacts that would arise if the circumstance or event occurs; and (ii) the likelihood of occurrence." The guide further defines risk assessment as "the process of identifying, estimating, and prioritizing risks to organizational operations (including mission, functions, image, reputation), organizational assets, individuals, other organizations, and the Nation, resulting from the operation of an information system. Part of risk management incorporates threat and vulnerability analyses, and considers mitigations provided by security controls planned or in place."

The NCCoE recommends that any discussion of risk management, particularly at the enterprise level, begins with a comprehensive review of NIST SP 800-37 Revision 2, *Risk Management Framework for Information Systems and Organizations*—material that is available to the public. The Risk Management Framework (RMF) guidance, as a whole, proved to be invaluable in giving us a baseline to assess risks, from which we developed the project, the security characteristics of the builds, and this guide.

Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks, NIST Interagency or Internal Report (NISTIR) 8228, identified security and privacy considerations and expectations that, together with the NIST Cybersecurity Framework and *Security and Privacy Controls for Federal Information Systems and Organizations* (NIST Special Publication [SP] 800-53 Revision 5), informed our risk assessment and subsequent recommendations from which we developed the security characteristics of the builds and this guide.

3.4.1 Threats

Historically, internet devices have enjoyed full connectivity at the network and transport layers. Any pair of devices with valid internet protocol (IP) addresses was, in general, able to communicate by using transmission control protocol (TCP) for connection-oriented communications or User Datagram Protocol (UDP) for connectionless protocols. Full connectivity was a practical architectural option for fully featured devices (e.g., servers and personal computers) because the identity of communicating hosts depended largely on the needs of inherently unpredictable human users. Requiring a reconfiguration of hosts to permit communications to meet the needs of system users as they evolved was not a scalable solution. However, a combination of allowing only certain device capabilities and blocking devices or domains that are considered suspicious allowed network administrators to mitigate some threats.

With the evolution of internet hosts from multiuser systems to personal devices, this security posture became impractical, and the emergence of IoT has made it unsustainable. In typical networking environments, a malicious actor can detect an IoT device and launch an attack on that device from any system on the internet. Once compromised, that device can be used to attack any other system on the internet. Anecdotal evidence indicates that a new device will be detected and will experience its first attack within minutes of deployment. Because the devices being deployed often have known security flaws, the success rate for compromising detected systems is very high. Typically, malware is designed to compromise a list of specific devices, making such attacks very scalable. Once compromised, an IoT device can be used to compromise other internet-connected devices, launch attacks on any victim device on the internet, or launch attacks on devices within the local network hosting the device.

3.4.2 Vulnerabilities

The vulnerability of IoT devices in this environment is a consequence of full connectivity, exacerbated by the large number of security vulnerabilities in complex software systems. Modern systems ship with millions of lines of code, creating a target-rich environment for malicious actors. Some vendors provide patches for security vulnerabilities and an efficient means for securely updating their products. However, patches are often unavailable or nearly impossible to install on many other products, including many IoT devices. In addition, poorly designed and implemented default configuration baselines and administrative access controls, such as hard-coded or widely known default passwords, provide a large attack surface for malicious actors. Many IoT devices include those types of vulnerabilities. The Mirai malware, which launched a large DDoS attack on the internet infrastructure firm Dyn that took down many of the internet’s top destinations offline for much of a day, relied heavily on hard-coded administrative access to assemble botnets consisting of more than 100,000 devices.

3.4.3 Risk

The demonstrated approach implements a set of protocols designed to permit users and product support staff to constrain access to MUD-capable IoT devices. A network that includes IoT devices will be vulnerable to exploitation if some but not all IoT devices are MUD-capable. MUD may help prevent a compromised IoT device from doing harm to other systems on the network, and a device acting out of profile may indicate that it is compromised. However, MUD does not necessarily help owners find and identify already-compromised systems, and it does not help owners correct compromised systems without replacing or reprogramming existing system components. For example, if a system is compromised so that it emits a new URL referencing a MUD file that permits malicious actors to send traffic to and from the IoT device, MUD may not be able to help owners detect such compromised systems and stop the communications that should be prohibited. However, if a system is compromised but it is still emitting the correct MUD URL, MUD can detect and stop any unauthorized communications that the device attempts. Such attempts would also indicate potential compromises.

If a network is set up so that it uses legacy IoT devices that do not emit MUD URLs, these devices could be associated with MUD URLs or with MUD files themselves by using alternative means, such as a device serial number or a public key. If the device is compromised and attempts unauthorized communication, the attempt should be detected, and the device would be subjected to the constraints specified in its MUD file. Under these circumstances, MUD can permit the owner to find and identify already-compromised systems. Moreover, where threat signaling is employed, a compromised system that reaches back to a known malicious IP address can be detected, and the connection can be refused.

4 Architecture

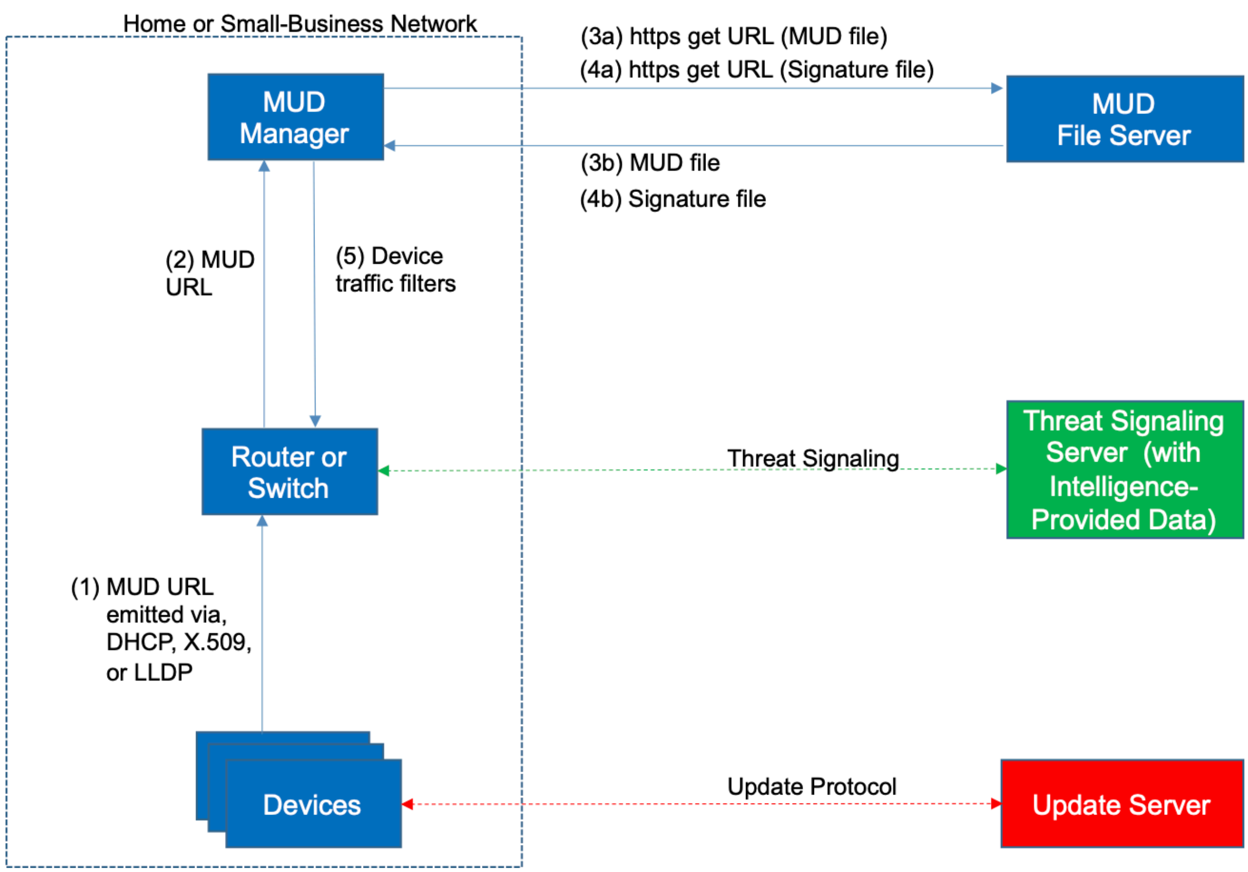
The project architecture is intended for home and small-business networks that are composed of both IoT components and fully featured devices (e.g., personal computers). The architecture is designed to provide three forms of protection:

- use of the MUD specification to automatically permit an IoT device to send and receive only the traffic it requires to perform as intended, thereby reducing the potential for the device to be the victim of a communications-based malware exploit or other network-based attack, and reducing the potential for the device, if compromised, to be used in a DDoS or other network-based attack
- use of network-wide access controls based on threat signaling to protect legacy (non-MUD-capable) IoT devices and fully featured devices, in addition to MUD-capable IoT devices, from connecting to domains that are known current threats
- automated secure software updates to all devices to ensure that operating system patches are installed promptly

4.1 Reference Architecture

Figure 4-1 depicts the logical architecture of the reference design. It consists of three main components: support for MUD, support for threat signaling, and support for periodic updates.

Figure 4-1 Reference Architecture



4.1.1 Support for MUD

A new functional component, the MUD manager, is introduced to augment the existing networking functionality offered by the home/small-business network router or switch. Note that the MUD manager is a logical component. Physically, the functionality that the MUD manager provides can and often is combined with that of the network router in a single device.

IoT devices must somehow be associated with a MUD file. The MUD specification describes three possible mechanisms through which the IoT device can provide the MUD file URL to the network: inserting the MUD URL into DHCP address requests that they generate when they attach to the network (e.g., when powered on) (supported by Builds 1, 2, and 4), providing the MUD URL in a Link Layer Discovery Protocol (LLDP) frame (also supported by Build 1), or providing the MUD URL as a field in an X.509 certificate that the device provides to the network via a protocol such as Tunnel Extensible Authentication Protocol. Each of these MUD URL emission mechanisms is listed as a possibility in Figure 4-1. In addition, the MUD specification provides flexibility to enable other mechanisms by which MUD file URLs can be associated with IoT devices. One alternative mechanism is to associate the device with its MUD file by using the bootstrapping information that the device conveys as part of the Wi-Fi Easy Connect onboarding process (supported by Build 3).

Figure 4-1 uses labeled arrows to depict the steps involved in supporting MUD when an IoT device emits its MUD file URL using one of the mechanisms specified in the MUD specification:

- The IoT device emits a MUD URL by using a mechanism such as DHCP, LLDP, or X.509 certificate (step 1).
- The router extracts the MUD URL from the protocol frame of whatever mechanism was used to convey it and forwards this MUD URL to the MUD manager (step 2).
- Once the MUD URL is received, the MUD manager uses hypertext transfer protocol secure (https) to request the MUD file from the MUD file server by using the MUD URL provided in the previous step (step 3a); if successful, the MUD file server at the specified location will serve the MUD file (step 3b).
- Next, the MUD manager uses https to request the signature file associated with the MUD file (step 4a) and upon receipt (step 4b) verifies the MUD file by using its signature file.
- The MUD file describes the communications requirements for the IoT device. Once the MUD manager has determined the MUD file to be valid, the MUD manager converts the access control rules in the MUD file into access control entries (e.g., access control lists—ACLs, firewall rules, or flow rules) and installs them on the router or switch (step 5).

If an alternative method of conveying the device's MUD file URL to the MUD manager is used (i.e., a mechanism other than emission of the MUD file URL via DHCP, X.509, or LLDP), steps 1 and 2 in Figure 4-1 would be replaced by that alternative mechanism.

Once the device's access control rules are applied to the router or switch, the MUD-capable IoT device will be able to communicate with approved local hosts and internet hosts as defined in the MUD file, and any unapproved communication attempts will be blocked.

As described in the MUD specification, the MUD file rules can limit both traffic between the device and external internet domains (north/south traffic), as well as traffic between the device and other devices on the local network (east/west traffic). East/west traffic can be limited by using the following constructs:

- controller—class of devices known to be controllers (could describe well-known services such as DNS or Network Time Protocol [NTP])
- my-controller—class of devices that the local network administrator admits to the class
- local-networks—class of IP addresses that are scoped within some local administrative boundary
- same-manufacturer—class of devices from the same manufacturer as the IoT device in question

- manufacturer—class of devices made by a particular manufacturer as identified by the authority component of its MUD URL

It is worth noting that while MUD requires use of a MUD-capable router on the local network, whether this router is stand-alone equipment provided by a third-party network equipment vendor (as is the case in Builds 1, 2, and 4) or integrated with the service provider's residential gateway equipment (Build 3) is not relevant to the ability of MUD to protect the network. While a service provider will be free to support MUD in its internet gateway equipment and infrastructure, such Internet Service Provider (ISP) support is not necessary. A home or small-business network can benefit from the protections that MUD has to offer without ISPs needing to make any changes or provide any support other than basic internet connectivity.

4.1.2 Support for Updates

To provide additional security, the reference architecture also supports periodic updates. All builds include a server that is meant to represent an update server to which MUD will permit devices to connect. Each device on an operational network should be configured to periodically contact its update server to download and apply security patches, ensuring that it is running the most up-to-date and secure code available. To ensure that such updates are possible, an IoT device's MUD file must explicitly permit the IoT device to receive traffic from the update server. Although regular manufacturer updates are crucial to security, the builds described in this practice guide demonstrate only the ability for IoT devices to receive faux updates from a notional update server. Communications between IoT devices and their corresponding update servers are not standardized.

4.1.3 Support for Threat Signaling

To provide additional protection for both MUD-capable and non-MUD-capable devices, the reference architecture also envisions support for threat signaling. The router or switch can receive threat feeds from a notional threat-signaling server to use as a basis for restricting certain types of network traffic. For example, both MUD-capable and non-MUD-capable devices can be prevented from connecting to internet domains that have been identified as being potentially malicious. Communications between the threat-signaling server and the router/switch are not standardized.

4.1.4 Build-Specific Features

The reference architecture depicted in Figure 4-1 is intentionally general. Each build instantiates this reference architecture in a unique way, depending on the equipment used and the capabilities supported. While all four builds support MUD and the ability to receive faux updates from a notional update server, only Build 2 currently supports threat signaling. Build 1 and Build 2 include nonstandard device discovery technology to discover, inventory, profile, and classify attached devices. Such classification can be used to validate that the access that is being granted to each device is consistent with that device's manufacturer and model. In Build 2, a device's manufacturer and model can be used as a basis for identifying and enforcing that device's traffic profile. Build 3 implements the Wi-Fi Easy Connect protocol to onboard both MUD-capable and non-MUD-capable devices, thereby securely providing each device with unique credentials for connecting to the network. For those devices that are both Easy Connect- and MUD-capable, the device's MUD rules are retrieved and installed on the local gateway during the onboarding process, ensuring that the device's MUD-based communication constraints are already in effect when the device connects to the network. Build 3 also creates and enforces separate trust zones (e.g., network segments) called *micronets* to which devices are assigned according to their intended network function.

The four builds of the reference architecture that have been completed and demonstrated are as follows:

- Build 1 uses products from Cisco Systems, DigiCert, Forescout, and Molex. The Cisco MUD manager supports MUD, and the Forescout virtual appliances and enterprise manager perform non-MUD-related device discovery on the network. Molex Power over Ethernet (PoE) Gateway and Light Engine are used as MUD-capable IoT devices. Certificates from DigiCert are also used.
- Build 2 uses products from MasterPeace Solutions, Ltd.; GCA,; ThreatSTOP; and DigiCert. The MasterPeace Solutions Yikes! router, cloud service, and mobile application support MUD as well as perform device discovery on the network and apply additional traffic rules to both MUD-capable and non-MUD-capable devices based on device manufacturer and model. The Yikes! router also integrates with the GCA Quad9 DNS service and the ThreatSTOP threat MUD file server to prevent devices (MUD-capable or not) from connecting to domains that have been identified as potentially malicious based on current threat intelligence. Certificates from DigiCert are also used.
- Build 3 uses products from CableLabs and DigiCert. CableLabs Micronets (e.g., Micronets Gateway, Micronets Manager, Micronets mobile phone application, and related service provider cloud-based infrastructure) supports MUD and implements the Wi-Fi Alliance's Wi-Fi Easy Connect protocol to securely onboard devices to the network. It also uses software-defined networking to create separate trust zones (e.g., network segments) called micronets to which devices are assigned according to their intended network function. Certificates from DigiCert are also used.
- Build 4 uses software developed at the NIST Advanced Networking Technologies laboratory. This software supports MUD and is intended to serve as a working prototype of the MUD specification to demonstrate feasibility and scalability. Certificates from DigiCert are also used.

The logical architectures and detailed descriptions of the builds mentioned above are in Section 6 (Build 1), Section 7 (Build 2), Section 8 (Build 3), and Section 9 (Build 4).

4.2 Physical Architecture

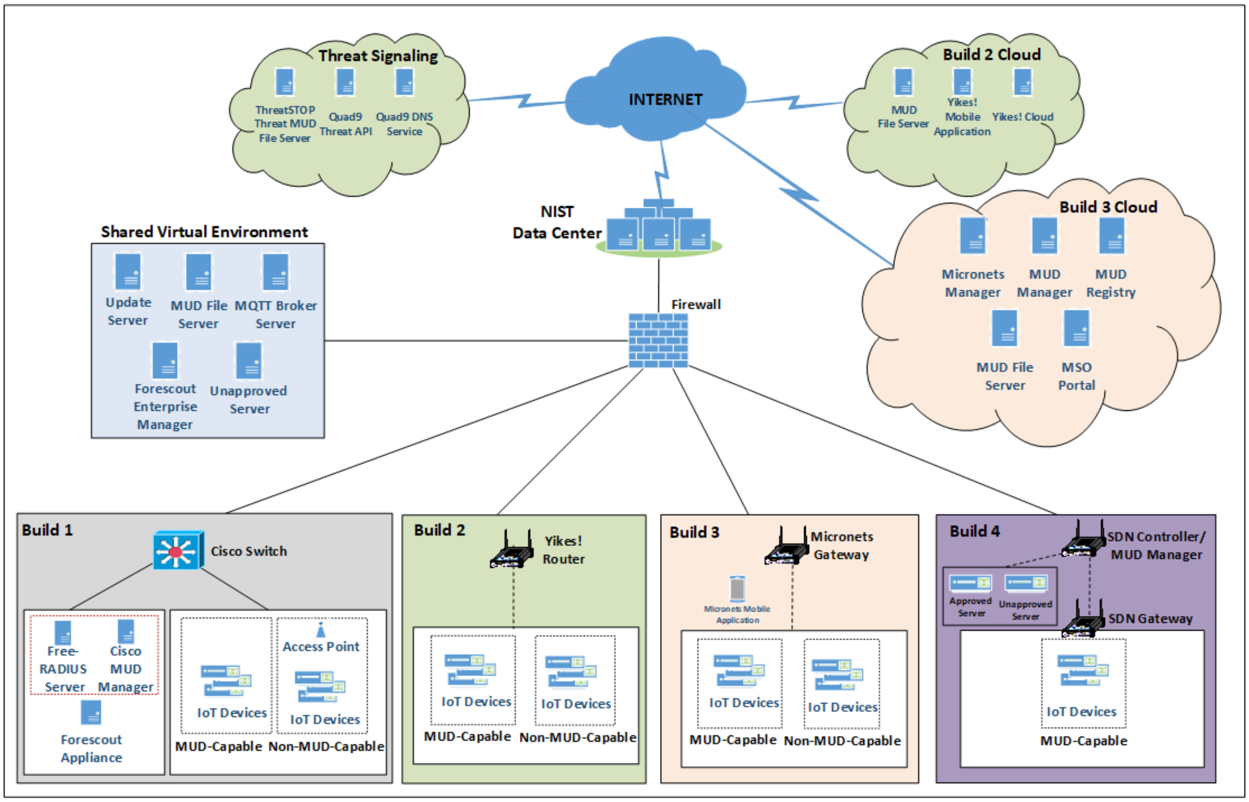
Figure 4-2 depicts the high-level physical architecture of the NCCoE laboratory environment. As depicted, the NCCoE laboratory network is connected to the internet via the NIST data center. Access to and from the NCCoE network is protected by a firewall. The NCCoE network includes a shared virtual environment that houses an update server, a MUD file server, an unapproved server (i.e., a server that is not listed as a permissible communications source or destination in any MUD file), a Message Queuing Telemetry Transport (MQTT) broker server, and a Forescout enterprise manager. These components are hosted at the NCCoE and are used across builds where applicable. DigiCert provided the Transport Layer Security (TLS) certificate and Premium Certificate used by the MUD file server.

All four builds, as depicted in the diagram, have been implemented:

- Build 1 network components consist of a Cisco Catalyst 3850-S switch, a Cisco MUD manager, a FreeRADIUS server, and a virtualized Forescout appliance on the local network. Build 1 also requires support from all components that are in the shared virtual environment, including the Forescout enterprise manager.
- Build 2 network components consist of a MasterPeace Solutions, Ltd. Yikes! router on the local network. Build 2 requires support from the MUD file server, Yikes! cloud, and a Yikes! mobile application that are resident on the Build 2 cloud. The Yikes! router includes threat-signaling capabilities (not depicted) that have been integrated with it. Build 2 also requires support from threat-signaling cloud services that consist of the ThreatSTOP threat MUD file server, Quad9 threat application programming interface (API), and Quad9 DNS service. Build 2 uses only the update server and unapproved server components that are in the shared virtual environment.
- Build 3 network components consist of a CableLabs Micronets Gateway/wireless access point (AP) that resides on the local network and that operates in conjunction with various service provider components and partner/service provider offerings that reside in the Micronets virtual environment in the Build 3 cloud. The Micronets Gateway is controlled by a Micronets Manager that resides in the Build 3 cloud and that coordinates a number of cloud-based Micronets micro-services, some of which are depicted. Build 3 also includes a Micronets mobile Application that provides the user and device interfaces for performing device onboarding.
- Build 4 network components consist of a software-defined networking (SDN)-capable gateway/switch on the local network, an SDN controller/MUD manager, and approved and unapproved servers that are located remotely from the local network. Build 4 also uses the MUD file server that is resident in the shared virtual environment.

IoT devices used in all four builds include those that are both MUD-capable and non-MUD-capable. The MUD-capable IoT devices used, which vary across builds, include Raspberry Pi, ARTIK, u-blox, Intel UP Squared, BeagleBone Black, NXP i.MX 8M (devkit), and the Molex Light Engine controlled by PoE Gateway. Non-MUD-capable devices used, which also vary across builds, include a wireless access point, cameras, a printer, mobile phones, lighting devices, a connected assistant device, a baby monitor, and a digital video recorder. Each of the completed builds and the roles that their components play in their architectures are explained in more detail in Section 6 (Build 1), Section 7 (Build 2), Section 8 (Build 3), and Section 9 (Build 4).

Figure 4-2 Physical Architecture



5 Security Characteristic Analysis

The purpose of the security characteristic analysis is to understand the extent to which the project meets its objective of demonstrating the ability to identify IoT components to MUD managers and manage access to those components while limiting unauthorized access to and from the components. In addition, it seeks to understand the security benefits of the demonstrated approach.

5.1 Assumptions and Limitations

The security characteristic analysis has the following limitations:

- It is neither a comprehensive test of all security components nor a red-team exercise.
- It cannot identify all weaknesses.
- It does not include the lab infrastructure. It is assumed that devices are hardened. Testing these devices would reveal only weaknesses in implementation that would not be relevant to those adopting this reference architecture.

5.2 Security Control Map

One aspect of our security characteristic analysis involved assessing how well the reference design addresses the security characteristics that it was intended to support. The Cybersecurity Framework Subcategories were used to provide structure to the security assessment by consulting the specific sections of each standard that are cited in reference to a Subcategory. The cited sections provide validation points that an example solution would be expected to exhibit. Using the Cybersecurity Framework Subcategories as a basis for organizing our analysis allowed us to systematically consider how well the reference design supports the intended security characteristics.

The characteristic analysis was conducted in the context of home network and small-business usage scenarios.

The capabilities demonstrated by the architectural elements described in Section 4 and used in the home networks and small-business environments are primarily intended to address requirements, best practices, and capabilities described in the following NIST documents: *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework), *Security and Privacy Controls for Federal Information Systems and Organizations* (NIST SP 800-53), and *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks* (NIST Interagency or Internal Report 8228). NISTIR 8228 identifies a set of 25 security and privacy expectations for IoT devices and subsystems. These include expectations regarding meeting device protection, data protection, and privacy protection goals. The reference architecture directly addresses the PR.AC-1, PR.AC-2, PR.AC-3, PR.AC-7, and PR.PT-3 Cybersecurity Framework Subcategories and supports activities addressing the ID.AM-1, ID.AM-2, ID.AM-3, ID.RA-2, ID.RA-3, PR.AC-5, PR.AC-4, PR.DS-5, PR.DS-6, PR.IP-1, PR.IP-3, and DE.CM-8 Subcategories. Also, the reference architecture directly addresses NIST SP 800-53 controls AC-3, AC-18, CM-7, IA-6, SC-5, SC-7, SC-23, and SI-2, and it supports activities addressing NIST SP 800-53 controls AC-4, AC-6, AC-24, CM-7, CM-8, IA-2, IA-5, IA-8, PA-4, PM-5, RA-5, SC-8, and SI-5. In addition, the reference architecture addresses eight of the NISTIR 8228 expectations. Table 5-1 describes how MUD-specific example implementation characteristics address NISTIR 8228 expectations, NIST SP 800-53 controls, and NIST Cybersecurity Framework Subcategories.

Table 5-1 Mapping Characteristics of the Demonstrated Approach, as Instantiated in at Least One of Builds 1-4, to NISTIR 8228 Expectations, NIST SP 800-53 Controls, and NIST Cybersecurity Framework Subcategories

Applicable Project Description Element that Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
There exists some mechanism for associating each device with a URL that can be used to identify and locate its MUD file. The device itself may emit the MUD file URL in one of three ways: • IoT devices insert the MUD URL into DHCP address requests when the device attaches to the network (e.g., powers on) (Build 1, Build 2, and Build 4). • MUD URL is provided in LLDP (Build 1). • MUD URL is included in X.509 certificate. However, a MUD URL may be learned by a network by other means, and the MUD specification is designed to allow flexibility in this regard. (In Build 3, the information required to retrieve the MUD URL from the MUD registry is conveyed using two fields in the device bootstrapping information, which is encoded in the device's Wi-Fi Easy Connect protocol QR code.)	Device has a built-in identifier.	Supports CM-8 System Component Inventory PM-5 System Inventory	Supports ID.AM-1 Physical devices and systems within the organization are inventoried.
Devices that support the Wi-Fi Easy Connect protocol have been preconfigured with their own unique bootstrapping public/private key pair before they initiate onboarding. Although the private key is not actually a device identifier, the device's possession of this unique private key is what enables the device to be authenticated as part of the onboarding protocol. (Build 3)	Device has a built-in unique identifier.	Supports CM-8 System Component Inventory PM-5 System Inventory	Supports ID.AM-1 Physical devices and systems within the organization are inventoried.
The MUD file URL, which identifies the device type, among other things, is passed to the MUD manager, which retrieves a MUD file by	Device can interface with	Provides AC-3 Access Enforcement	Provides PR.PT-3 The principle of least functionality

Applicable Project Description Element that Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
using https. The MUD file describes the communications requirements for this device. The MUD manager converts the requirements into access control information for enforcement by the router or switch. (all builds)	enterprise asset management systems.	AC-18 Wireless Access CM-7 Least Functionality SC-5 Denial of Service Protection SC-7 Boundary Protection Supports AC-4 Information Flow Enforcement AC-6 Least Privilege AC-24 Access Control Decisions CM-8 System Component Inventory PM-5 System Inventory	is incorporated by configuring systems to provide only essential capabilities. Supports ID.AM-1 Physical devices and systems within the organization are inventoried. ID.AM-2 Software platforms and applications within the organization are inventoried. ID.AM-3 Organizational communication and data flows are mapped. PR.AC-4 Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. PR.AC-5 Network integrity is protected (e.g.,

Applicable Project Description Element that Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
			network segregation, network segmentation). PR.DS-5 Protections against data leaks are implemented. DE.AE-1 A baseline of network operations and expected data flows for users and systems is established and managed.
IoT devices periodically contact the appropriate update server to download and apply security patches. (all builds)	The manufacturer will provide patches or upgrades for all software and firmware throughout each device's life span.	Provides SI-2 Flaw Remediation	Supports PR.IP-1 A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). PR.IP-3 Configuration change control processes are in place.
The router or switch receives threat feeds from the threat-signaling server to use as a basis for restricting certain types of network traffic. (Build 2)	The device either supports the use of vulnerability	Supports AC-24 Access Control Decisions	Supports ID.RA-2 Cyber threat intelligence is received

Applicable Project Description Element that Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
	scanners or provides built-in vulnerability identification and reporting capabilities.	RA-5 Vulnerability Scanning SI-5 Security Alerts, Advisories, and Directives	from information-sharing forums and sources. ID.RA-3 Threats, both internal and external, are identified and documented. DE.CM-8 Vulnerability scans are performed.
Using the Wi-Fi Easy Connect protocol to onboard devices ensures that there is no need for anyone to be privy to the device's network credentials. The onboarding protocol provisions the network credentials onto the device automatically, using a secure channel, and the device is then able to present its credentials to the network as part of the standard Wi-Fi network connection handshake. There is no need for the device's network password to be input by a human, and the credentials are never displayed, so presentation of the device's network credentials to the network does not pose any risk that the credentials will be viewed and thereby disclosed. (Build 3)	The device can conceal password characters from display when a person enters a password for a device, such as on a keyboard or touchscreen.	Supports IA-6 Authenticator Feedback	Provides PR.AC-7 Users, devices, and other assets are authenticated commensurate with the risk of the transaction.
The MUD file URL is passed to the MUD manager, which retrieves a MUD file from the designated website (denoted as the MUD file server) by using https. The MUD file server must have a valid TLS certificate, and the MUD file itself must have a valid signature. The MUD file describes the communications requirements for this device. The MUD manager converts the requirements into access	The device can use existing enterprise authenticators and authentication mechanisms.	Supports IA-2 Identification and Authentication (Organizational Users) IA-5 Authenticator Management	Provides PR.AC-1 Identities and credentials are issued, managed, verified, revoked, and audited for

Applicable Project Description Element that Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
control information for enforcement by the router or switch. (all builds)		IA-8 Identification and Authentication (Non-Organizational Users)	authorized devices, users, and processes. PR.AC-3 Remote access is managed. PR.AC-7 Users, devices, and other assets are authenticated commensurate with the risk of the transaction.
Each device that is onboarded using the Wi-Fi Easy Connect protocol is provisioned with unique network credentials that enable the device to authenticate to the network as part of the standard Wi-Fi network connection handshake. (Build 3)	The device can use existing enterprise authenticators and authentication mechanisms.	Supports IA-2 Identification and Authentication (Organizational Users) IA-5 Authenticator Management IA-8 Identification and Authentication (Non-Organizational Users)	Provides PR.AC-1 Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users, and processes. PR.AC-3 Remote access is managed. PR.AC-7 Users, devices, and other assets are authenticated commensurate with the risk of the transaction.
There exists some mechanism for associating each device with a URL that can identify and locate its MUD file. The MUD file URL is passed to the MUD manager, which retrieves	Device can prevent unauthorized ac-	Provides SC-23 Session Authenticity	Provides PR.PT-3 The principle of least functionality

Applicable Project Description Element that Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
a MUD file from the designated website (denoted as the MUD file server) by using https. The MUD file describes the communications requirements for this device. The MUD manager converts the requirements into access control information for enforcement by the router or switch. (all builds)	cess to all sensitive data transmitted from it over networks.	Supports AC-18 Wireless Access SC-8 Transmission Confidentiality and Integrity	is incorporated by configuring systems to provide only essential capabilities. Supports PR.DS-5 Protections against data leaks are implemented. PR.DS-6 Integrity-checking mechanisms are used to verify software, firmware, and information integrity.
There exists some mechanism for associating each device with a URL that can identify and locate its MUD file. The MUD file URL is passed to the MUD manager, which retrieves a MUD file from the designated website (denoted as the MUD file server) by using https. The MUD file describes the communications requirements for this device. The MUD manager converts the requirements into access control information for enforcement by the router or switch. (all builds) The router or switch periodically receives threat feeds from the threat-signaling server to use as a basis for restricting certain types of network traffic. (Build 2)	There is sufficient centralized control to apply policy or regulatory requirements to personally identifiable information.	Supports PA-4 Information Sharing with External Parties	None

886 Table 5-2 details Cybersecurity Framework Identify, Protect, and Detect Categories and Subcategories
887 that the example implementations directly address or for which the example implementations may

serve a supporting role. Entries in the Cybersecurity Framework Subcategory column that are directly addressed are highlighted in green. Informative references are made for each Subcategory. The following sources are used for informative references: Center for Internet Security (CIS), Control Objectives for Information and Related Technology (COBIT), International Society of Automation (ISA), International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), and NIST SP 800-53. While some of the references provide general guidance that informs implementation of referenced Cybersecurity Framework Core Functions, the NIST SP and Federal Information Processing Standard (FIPS) references provide specific recommendations that should be considered when composing and configuring security platforms. (Note that not all of the informative references apply to this example implementation.)

Table 5-2 Mapping Project Objectives to the Cybersecurity Framework and Informative Security Control References

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
Asset Management (ID.AM): The data, personnel, devices, systems, and facilities that enable the organization to achieve business purposes are identified and managed consistent with their relative importance to business objectives and the organization's risk strategy.	**ID.AM-1:** Physical devices and systems within the organization are inventoried.	**CIS** CSC 1 **COBIT 5** BAI09.01, BAI09.02 **ISA 62443-2-1:2009** 4.2.3.4 **ISA 62443-3-3:2013** SR 7.8 **ISO/IEC 27001:2013** A.8.1.1, A.8.1.2 **NIST SP 800-53 Rev. 4** CM-8, PM-5
	ID.AM-2: Software platforms and applications within the organization are inventoried.	**CIS** CSC 2 **COBIT 5** BAI09.01, BAI09.02, BAI09.05 **ISA 62443-2-1:2009** 4.2.3.4 **ISA 62443-3-3:2013** SR 7.8 **ISO/IEC 27001:2013** A.8.1.1, A.8.1.2, A.12.5.1 **NIST SP 800-53 Rev. 4** CM-8, PM-5
	ID.AM-3: Organizational communication and data flows are mapped.	**CIS** CSC 12 **COBIT 5** DSS05.02 **ISA 62443-2-1:2009** 4.2.3.4 **ISA 62443-3-3:2013** SR 7.8 **ISO/IEC 27001:2013** A.8.1.1, A.8.1.2, A.12.5.1 **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8
Risk Assessment (ID.RA): The organization understands the	**ID.RA-2:** Cyber threat intelligence is received from information-sharing forums and sources.	**CIS** CSC 4 **COBIT 5** BAI08.01 **ISA 62443-2-1:2009** 4.2.3, 4.2.3.9,

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
cybersecurity risk to organizational operations (including mission, functions, image, or reputation), organizational assets, and individuals.		4.2.3.12 **ISO/IEC 27001:2013** A.6.1.4 **NIST SP 800-53 Rev. 4** SI-5, PM-15, PM-16
	ID.RA-3: Threats, both internal and external, are identified and documented.	**CIS** CSC 4 **COBIT 5** APO12.01, APO12.02, APO12.03, APO12.04 **ISA 62443-2-1:2009** 4.2.3, 4.2.3.9, 4.2.3.12 **ISO/IEC 27001:2013** Clause 6.1.2 **NIST SP 800-53 Rev. 4** RA-3, SI-5, PM-12, PM-16
Identity Management, Authentication, and Access Control (PR.AC): Access to physical and logical assets and associated facilities is limited to authorized users, processes, and devices and is managed consistent with the assessed risk of unauthorized access to authorized activities and transactions.	**PR.AC-1:** Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users, and processes.	**CIS** CSC 1, 5, 15, 16 **COBIT 5** DSS05.04, DSS06.03 **ISA 62443-2-1:2009** 4.3.3.5.1 **ISA 62443-3-3:2013** SR 1.1, SR 1.2, SR 1.3, SR 1.4, SR 1.5, SR 1.7, SR 1.8, SR 1.9 **ISO/IEC 27001:2013** A.9.2.1, A.9.2.2, A.9.2.3, A.9.2.4, A.9.2.6, A.9.3.1, A.9.4.2, A.9.4.3 **NIST SP 800-53 Rev. 4** AC-1, AC-2, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11
	PR.AC-3: Remote access is managed.	**CIS** CSC 12 **COBIT 5** APO13.01, DSS01.04, DSS05.03 **ISA 62443-2-1:2009** 4.3.3.6.6 **ISA 62443-3-3:2013** SR 1.13, SR 2.6 **ISO/IEC 27001:2013** A.6.2.1, A.6.2.2, A.11.2.6, A.13.1.1, A.13.2.1 **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15
	PR.AC-4: Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties.	**CIS** CSC 3, 5, 12, 14, 15, 16, 18 **COBIT 5** DSS05.04 **ISA 62443-2-1:2009** 4.3.3.7.3 **ISA 62443-3-3:2013** SR 2.1 **ISO/IEC 27001:2013** A.6.1.2, A.9.1.2,

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
		A.9.2.3, A.9.4.1, A.9.4.4, A.9.4.5 **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24
	PR.AC-5: Network integrity is protected, incorporating network segregation where appropriate.	**CIS** CSC 9, 14, 15, 18 **COBIT 5** DSS01.05, DSS05.02 **ISA 62443-2-1:2009** 4.3.3.4 **ISA 62443-3-3:2013** SR 3.1, SR 3.8 **ISO/IEC 27001:2013** A.13.1.1, A.13.1.3, A.13.2.1, A.14.1.2, A.14.1.3 **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7
	PR.AC-7: Users, devices, and other assets are authenticated (e.g., single-factor, multifactor) commensurate with the risk of the transaction (e.g., individuals' security and privacy risks and other organizational risks).	**CIS** CSC 1, 12, 15, 16 **COBIT 5** DSS05.04, DSS05.10, DSS06.10 **ISA 62443-2-1:2009** 4.3.3.6.1, 4.3.3.6.2, 4.3.3.6.3, 4.3.3.6.4, 4.3.3.6.5, 4.3.3.6.6, 4.3.3.6.7, 4.3.3.6.8, 4.3.3.6.9 **ISA 62443-3-3:2013** SR 1.1, SR 1.2, SR 1.5, SR 1.7, SR 1.8, SR 1.9, SR 1.10 **ISO/IEC 27001:2013** A.9.2.1, A.9.2.4, A.9.3.1, A.9.4.2, A.9.4.3, A.18.1.4 **NIST SP 800-53 Rev. 4** AC-7, AC-8, AC-9, AC-11, AC-12, AC-14, IA-1, IA-2, IA-3, IA-4, IA-5, IA-8, IA-9, IA-10, IA-11
Data Security (PR.DS): Information and records (data) are managed consistent with the organization's risk strategy to protect the confidentiality, integrity, and availability of information.	**PR.DS-5:** Protections against data leaks are implemented.	**CIS** CSC 13 **COBIT 5** APO01.06, DSS05.04, DSS05.07, DSS06.02 **ISA 62443-3-3:2013** SR 5.2 **ISO/IEC 27001:2013** A.6.1.2, A.7.1.1, A.7.1.2, A.7.3.1, A.8.2.2, A.8.2.3, A.9.1.1, A.9.1.2, A.9.2.3, A.9.4.1, A.9.4.4, A.9.4.5, A.10.1.1, A.11.1.4, A.11.1.5, A.11.2.1, A.13.1.1, A.13.1.3, A.13.2.1, A.13.2.3, A.13.2.4, A.14.1.2, A.14.1.3 **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
		6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4
	PR.DS-6: Integrity-checking mechanisms are used to verify software, firmware, and information integrity.	**ISA 62443-3-3:2013** SR 3.1, SR 3.3, SR 3.4, SR 3.8 **ISO/IEC 27001:2013** A.12.2.1, A.12.5.1, A.14.1.2, A.14.1.3 **FIPS 140-2** Sec. 4 **NIST SP 800-45 Ver. 2** 2.4.2, 3, 4.2.3, 4.3, 5.1, 6.1, 7.2.2, 8.2, 9.2 **NIST SP 800-49** 2.2.1, 2.3.2, 3.4 **NIST SP 800-52 Rev. 1** 3, 4, D1.4 **NIST SP 800-53 Rev. 4** SI-7 **NIST SP 800-57 Part 1 Rev. 4** 5.5, 6.1, 8.1.5.1, B.3.2, B.5 **NIST SP 800-57 Part 2** 1, 3.1.2.1.2, 4.1, 4.2, 4.3, A.2.2, A.3.2, C.2.2 **NIST SP 800-81-2** All **NIST SP 800-130** 2.2, 4.3, 6.2.1, 6.3, 6.4, 6.5, 6.6.1 **NIST SP 800-152** 6.1.3, 6.2.1, 8.2.1, 8.2.4, 9.4 **NIST SP 800-177** 2.2, 4.1, 4.4, 4.5, 4.7, 5.2, 5.3
Information Protection Processes and Procedures (PR.IP): Security policies (that address purpose, scope, roles, responsibilities, management commitment, and coordination among organizational entities), processes, and procedures are maintained and used to manage protection of information systems and assets.	**PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality).	**CIS** CSC 1 **COBIT 5** BAI10.01, BAI10.02, BAI10.03, BAI10.05 **ISA 62443-2-1:2009** 4.3.4.3.2, 4.3.4.3.3 **ISA 62443-3-3:2013** SR 7.6 **ISO/IEC 27001:2013** A.12.1.2, A.12.5.1, A.12.6.2, A.14.2.2, A.14.2.3, A.14.2.4 **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10
	PR.IP-3: Configuration change control processes are in place.	**CIS** CSC 3, 11 **COBIT 5** BAI01.06, BAI06.01 **ISA 62443-2-1:2009** 4.3.4.3.2,

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
		4.3.4.3.3 **ISA 62443-3-3:2013** SR 7.6 **ISO/IEC 27001:2013** A.12.1.2, A.12.5.1, A.12.6.2, A.14.2.2, A.14.2.3, A.14.2.4 **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10
Protective Technology (PR.PT): Technical security solutions are managed to ensure the security and resilience of systems and assets, consistent with related policies, procedures, and agreements.	**PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities.	**CIS** CSC 3, 11, 14 **COBIT 5** DSS05.02, DSS05.05, DSS06.06 **ISA 62443-2-1:2009** 4.3.3.5.1, 4.3.3.5.2, 4.3.3.5.3, 4.3.3.5.4, 4.3.3.5.5, 4.3.3.5.6, 4.3.3.5.7, 4.3.3.5.8, 4.3.3.6.1, 4.3.3.6.2, 4.3.3.6.3, 4.3.3.6.4, 4.3.3.6.5, 4.3.3.6.6, 4.3.3.6.7, 4.3.3.6.8, 4.3.3.6.9, 4.3.3.7.1, 4.3.3.7.2, 4.3.3.7.3, 4.3.3.7.4 **ISA 62443-3-3:2013** SR 1.1, SR 1.2, SR 1.3, SR 1.4, SR 1.5, SR 1.6, SR 1.7, SR 1.8, SR 1.9, SR 1.10, SR 1.11, SR 1.12, SR 1.13, SR 2.1, SR 2.2, SR 2.3, SR 2.4, SR 2.5, SR 2.6, SR 2.7 **ISO/IEC 27001:2013** A.9.1.2 **NIST SP 800-53 Rev. 4** AC-3, CM-7
Security Continuous Monitoring (DE.CM): The information system and assets are monitored to identify cybersecurity events and verify the effectiveness of protective measures.	**DE.CM-8:** Vulnerability scans are performed.	**CIS** CSC 4, 20 **COBIT 5** BAI03.10, DSS05.01 **ISA 62443-2-1:2009** 4.2.3.1, 4.2.3.7 **ISO/IEC 27001:2013** A.12.6.1 **NIST SP 800-53 Rev. 4** RA-5

Additional resources required to develop this solution are identified in Appendix C. The core standards, secure update standards, industry best practices for software quality, and best practices for identification and authentication are generally stable, well understood, and available in the commercial off-the-shelf market. Standards associated with the MUD protocol are in an advanced level of development by the IETF.

5.3 Scenarios

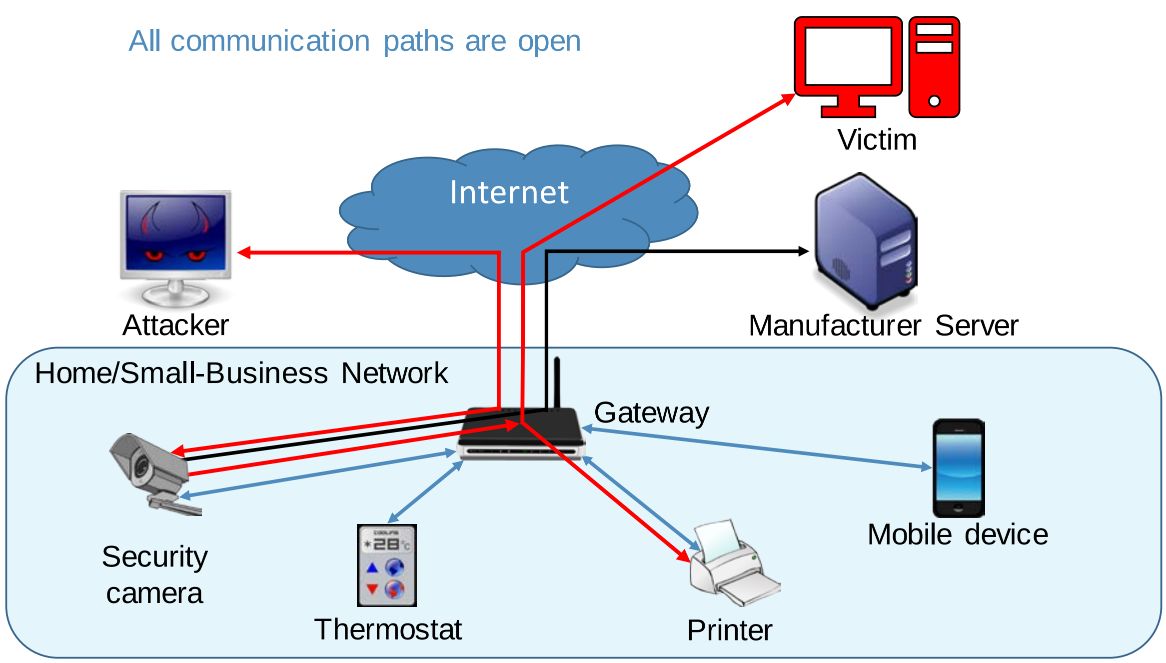
This section presents two scenarios involving home and small-business networks that have IoT devices. In the first scenario, MUD is not deployed on the network, so IoT devices are vulnerable to being port scanned and are not restricted from exchanging traffic with either external sites or other devices on the local network. IoT devices in this first scenario are highly vulnerable to attack. Threat signaling is not deployed either, so none of the devices on the local network are being protected from traffic sent from known malicious actors.

In the second scenario, both MUD and threat signaling are deployed on the network. The MUD files are being used to restrict traffic from being sent between the local IoT devices and some external internet domains (i.e., north/south traffic) as well as traffic among the local IoT devices themselves (i.e., east/west traffic). MUD ensures that each IoT device is permitted to exchange traffic with only external domains and internal devices that are explicitly specified in its MUD file. Threat signaling protects all devices, not just IoT devices, from communicating with sites that are known to be malicious.

5.3.1 Scenario 1: No MUD or Threat-Signaling Protection

In the No MUD or Threat-Signaling Protection scenario, as shown in Figure 5-1, the home/small-business network (depicted by the light blue rectangular box) does not have MUD deployed to provide security for its IoT devices, nor does it use threat signaling.

Figure 5-1 No MUD or Threat-Signaling Protection

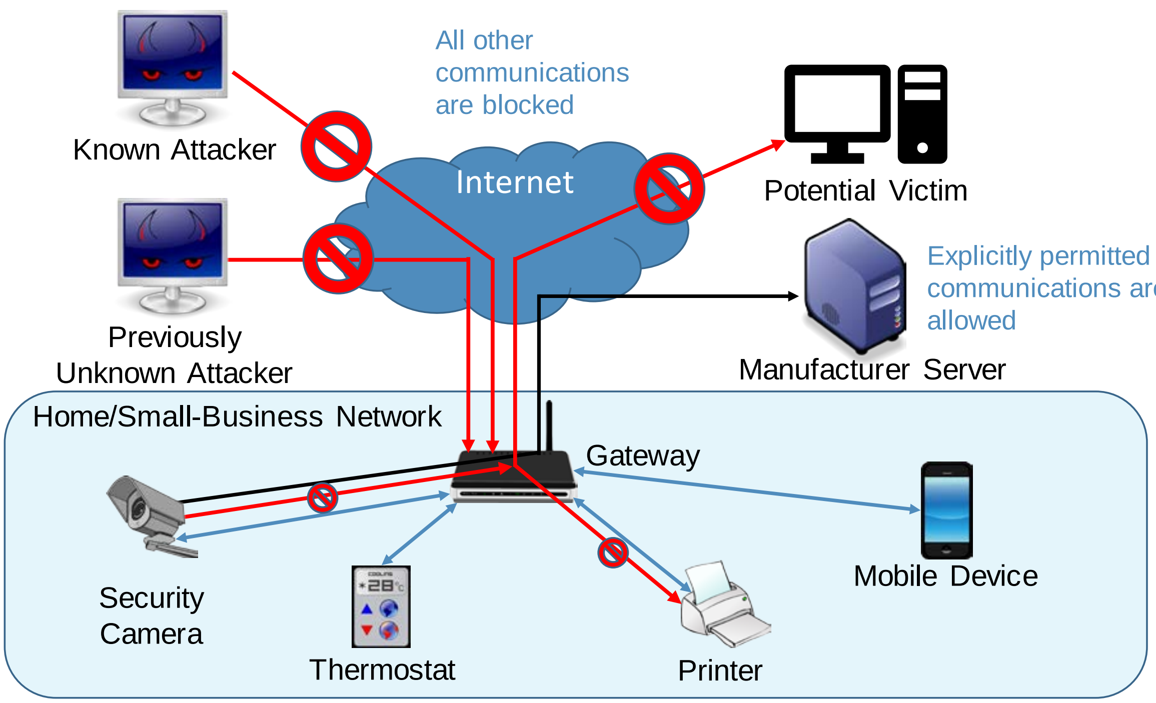


All communication paths are open. The IoT devices on the network can be port scanned (and perhaps hijacked) by an attacker on the internet. IoT devices are permitted to communicate to and from intended services, such as a manufacturer update server, as desired. However, the IoT devices are also reachable by malicious external devices and by compromised devices that are on their local network, making them vulnerable to attacks from these malicious and compromised devices. In addition, if an IoT device on the local network becomes compromised, there are no protections in place to stop it from launching an attack on outside or local devices, creating additional potential victims. As shown in Figure 5-1, an external malicious actor can attack a security camera on the local network, compromise that camera, and use it to launch additional attacks on both local and remote targets.

5.3.2 Scenario 2: MUD and Threat-Signaling Protection

In the MUD and Threat-Signaling Protection scenario, as shown in Figure 5-2, the home/small-business network (depicted by the light blue rectangle) has both MUD and threat signaling deployed. (For simplicity, the components of the MUD deployment such as the MUD manager and MUD file server are not depicted, nor are the components of the threat-signaling deployment.) The MUD file for each MUD-capable IoT device lists the domains of all external services with which the MUD-capable device is permitted to exchange traffic. All external domains that are not explicitly permitted in the MUD file are denied. Therefore, each MUD-capable IoT device on the network can freely communicate with its intended external services, but all other attempted communications between that MUD-capable IoT device and external sites are blocked. The MUD-capable IoT device cannot be port scanned or receive traffic from external malicious domains if communication with those domains is not explicitly permitted in the IoT device’s MUD file, even if those domains are not known to be malicious. Furthermore, even if the MUD-capable IoT device is compromised in some way after it has connected to the local network, it will not be permitted to attack any external domains if communication with those domains is not explicitly permitted in the MUD-capable IoT device’s MUD file.

Figure 5-2 MUD and Threat-Signaling Protection



In Figure 5-2, the symbol prohibiting traffic sent from the previously unknown attacker depicts the fact that MUD prevents MUD-capable devices from receiving traffic from external sites that are not listed in those devices' MUD files. The symbol prohibiting traffic sent from the security camera to the potential external victim depicts the fact that MUD prevents MUD-capable devices from sending traffic to external targets that are not explicitly permitted in their MUD files.

One of the external sites with which a MUD-capable IoT device is permitted to communicate is a manufacturer update server, from which the IoT device receives regular software updates to ensure that it installs the most recent security patches as needed.

In addition to listing external domains with which each MUD-capable device is permitted to communicate, the MUD file for each MUD-capable device restricts the local devices that each MUD-capable IoT device is permitted to exchange traffic with based on characteristics such as those devices' manufacturer or model or whether those other devices are controllers for the IoT device in question. If a local device is not from the specified manufacturer, for example, it will not be permitted to exchange traffic with the MUD-capable IoT device. So, if a device on the local network attempts to attack another device on the local network that is MUD-capable, the traffic will not be received by that MUD-capable device if the attacking device is not from a manufacturer specified in the MUD-capable device's MUD file. Conversely, if a MUD-capable IoT device becomes compromised, it will not be permitted to attack any local devices that are not from a manufacturer specified in the MUD-capable IoT device's MUD file.

In Figure 5-2, the symbol prohibiting traffic received at the printer depicts the fact that MUD prevents MUD-capable devices from receiving traffic from all local devices that are not permitted in their MUD files. The symbol prohibiting traffic sent from the security camera to the printer depicts the fact that MUD prevents MUD-capable devices from sending traffic to other local devices that are not explicitly permitted in their MUD files.

In addition to MUD, threat signaling is deployed. Threat signaling prevents all devices on the local network from communicating with external domains that are known to be malicious. It protects not just MUD-capable IoT devices but also non-MUD-capable IoT devices and fully functional devices such as cell phones and laptops. This protection is depicted in Figure 5-2 by the symbol prohibiting receipt of traffic sent from the known attacker.

6 Build 1

The Build 1 implementation uses products from Cisco Systems, DigiCert, Forescout, and Molex. Cisco equipment supports MUD. Build 1 uses the Cisco MUD manager, which is available as open-source software; and the Cisco Catalyst 3850-S switch, which has been customized to work with the MUD manager, to provide switching, DHCP, and LLDP services. Build 1 also uses the Forescout virtual appliances and enterprise manager to perform discovery of all types of devices on the network—both MUD-capable and non-MUD-capable. Build 1 uses Molex PoE Gateway and Light Engine as MUD-capable IoT devices. Build 1 also uses certificates from DigiCert.

6.1 Collaborators

Collaborators that participated in this build are described briefly in the subsections below.

6.1.1 Cisco Systems

Cisco Systems is a provider of enterprise, telecommunications, and industrial networking solutions. The work in this project was undertaken within Cisco's Enterprise Central Software Group with an eye toward improving the product offering over time. Cisco provided a proof-of-concept MUD manager as well as a Catalyst 3850-S switch with Power over Ethernet. Learn more about Cisco Systems at https://www.cisco.com.

6.1.2 DigiCert

DigiCert is a major provider of scalable TLS/secure sockets layer (SSL), and public key infrastructure (PKI) solutions for identity and encryption. The company is known for its expertise in identity and encryption for web servers and Internet of Things devices. DigiCert supports TLS/SSL and other digital certificates for PKI deployments at any scale through its certificate life-cycle management platform, CertCentral®. The company provides enterprise-grade certificate management platforms, responsive customer support, and advanced security solutions. Learn more about DigiCert at https://www.digicert.com.

6.1.3 Forescout

Forescout Technologies is an industry leader in device visibility and control. Forescout's unified security platform enables enterprises and government agencies to gain complete situational awareness of their extended enterprise environment and to orchestrate actions to reduce cyber and operational risk. Forescout products deploy quickly with agentless, real-time discovery and classification of every connected device, as well as with continuous posture assessment. As of December 31, 2019, more than 3,700 customers in over 90 countries rely on Forescout's infrastructure-agnostic solution to reduce the risk of business disruption from security incidents or breaches, to demonstrate security compliance, and to increase security operations productivity. Learn more about Forescout at https://www.forescout.com.

6.1.4 Molex

Molex brings together innovation and technology to deliver electronic solutions to customers worldwide. With a presence in more than 40 countries, Molex offers a full suite of solutions and services for many markets, including data communications, consumer electronics, industrial, automotive, commercial vehicle, and medical. Learn more about Molex at https://www.molex.com.

6.2 Technologies

Table 6-1 lists all of the products and technologies used in Build 1 and provides a mapping among the generic component term, the specific product used to implement that component, and the security Function Subcategories that the product provides. When applicable, both the Function Subcategories that a component provides directly and those that it supports but does not provide directly are listed and labeled as such. For rows in which the provides/supports distinction is not noted, the component directly provides all listed Categories. Refer to Table 5-1 for an explanation of the NIST Cybersecurity Framework Subcategory codes.

1022 **Table 6-1 Products and Technologies Used in Build 1**

Component	Product	Function	Cybersecurity Framework Subcategories
MUD manager	Cisco MUD manager (open source) and a FreeRADIUS server	Fetches, verifies, and processes MUD files from the MUD file server; configures router or switch with traffic filters to enforce access control based on the MUD file	Provides PR.PT-3 Supports ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 DE.AE-1
MUD file server	NCCoE-hosted Apache server	Hosts MUD files; serves MUD files to the MUD manager by using https	ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
MUD file maker	MUD file maker (https://www.mud-maker.org/)	Yet Another Next Generation (YANG) script graphical user interface (GUI) used to create MUD files	ID.AM-1
MUD file	A YANG model instance that has been serialized in JavaScript Object Notation (JSON) (RFC 7951). The manufacturer of a MUD-capable device creates that device's MUD file. MUD file maker (see previous row) can create MUD files. Each MUD file is also associated with a separate MUD signature file.	Specifies the communications that are permitted to and from a given device	Provides PR.PT-3 Supports ID.AM-1 ID.AM-2 ID.AM-3

Component	Product	Function	Cybersecurity Framework Subcategories
DHCP server	Cisco Internetwork Operating System (IOS) (Catalyst 3850-S)	Dynamically assigns IP addresses; recognizes MUD URL in DHCP DISCOVER message; should notify MUD manager if the device's IP address lease expires or has been released	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
LLDP	Cisco IOS (Catalyst 3850-S)	Supports capability for devices to advertise their identity and capabilities to neighbors on a local area network segment; provides capability to receive MUD URL in IoT device LLDP type-length-value (TLV) frame as an extension	ID.AM-1
Router or switch	Cisco Catalyst 3850-S (IOS XE software version 16.09.02)	Provides MUD URL to MUD manager; gets configured by the MUD manager to enforce the IoT device's communication profile; performs per-device access control	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
Certificates	DigiCert certificates (TLS and premium)	Authenticates MUD file server and secures TLS connection between MUD manager and MUD file server; signs MUD files and generates corresponding signature file	PR.AC-1 PR.AC-3 PR.AC-5 PR.AC-7

Component	Product	Function	Cybersecurity Framework Subcategories
MUD-capable IoT device	Raspberry Pi Model 3B (devkit) u-blox C027-G35 (devkit) Samsung ARTIK 520 (devkit) Intel UP Squared Grove (devkit) Molex PoE Gateway and Light Engine	Emits a MUD URL as part of its DHCP DISCOVER message; requests and applies software updates	ID.AM-1
Non-MUD-capable IoT device	Camera Mobile phones Connected lighting devices Connected assistant Printer Baby monitor Wireless access point Digital video recorder	Acts as typical IoT device on a network; creates network connections to cloud services	ID.AM-1
Update server	NCCoE-hosted Apache server Molex update agent	Acts as a device manufacturer's update server that would communicate with IoT devices to provide patches and other software updates	PR.IP-1 PR.IP-3
Unapproved server	NCCoE-hosted Apache server	Acts as an internet host that has not been explicitly approved in a MUD file	DE.DP-3 DE.AM-1
MQTT broker server	NCCoE-hosted MQTT server	Receives and publishes messages to/from clients	ID.AM-3 DE.AE-3
IoT device discovery	Forescout virtual appliances and enterprise manager	Discover IoT devices on network	ID.AM-1 PR.IP-1 DE.AM-1

1023 Each of these components is described more fully in the following sections.

6.2.1 MUD Manager

The MUD manager is a key component of the architecture. It fetches, verifies, and processes MUD files from the MUD file server. It then configures the router or switch with an access list to control communications based on the contents of the MUD files.

The Cisco MUD manager is an open-source implementation. For this project, we used the Cisco MUD manager to support IoT devices that emit their MUD URLs via DHCP messages and to support other IoT devices that emit their MUD URLs via the Institute of Electrical and Electronics Engineers (IEEE) 802.1AB LLDP. The Cisco MUD manager is supported by an open-source implementation of an authentication, authorization, and accounting (AAA) server that communicates by using the Remote Authentication Dial-In User Service (RADIUS) protocol (i.e., a RADIUS server) called FreeRADIUS. When the MUD URL is emitted via DHCP or LLDP, it is extracted from the corresponding message, and the switch thereafter provides these MUD URLs to the MUD manager via RADIUS messages. The MUD manager then retrieves MUD files associated with those URLs and configures the Catalyst 3850-S switch to enforce the IoT devices' communication profiles based on these MUD files. The switch implements an IP access control list-based policy for src-dnsname, dst-dnsname, my-controller, and controller constructs that are specified in the MUD file, and it uses virtual local area networks (VLANs) to enforce same-manufacturer, manufacturer, and local-networks constructs that are specified in the MUD file. The system supports both lateral east/west protection and appropriate access to internet sites (north/south protection).

When supporting MUD URL emission by LLDP TLV, LLDP TLV must be enabled on both the Cisco switch and the IoT device. A policy-map configuration and a corresponding template are used to cause media access control (MAC) authentication bypass to happen. This will trigger an access-session attribute that will cause LLDP TLVs (including the MUD URL) to be forwarded in an accounting message to the RADIUS server.

Some manual preconfiguration of VLANs on the switch is required. The Cisco MUD manager supports a default policy for IPv4. It implements a static mapping between domain names and IP addresses inside a configuration file.

The version of the Cisco MUD manager used in this project is a proof-of-concept implementation that is intended to introduce advanced users and engineers to the MUD concept. It is not a fully automated MUD manager implementation, and some protocol features are not present. These are described in Section 10.1, Findings.

6.2.2 MUD File Server

In the absence of a commercial MUD file server for this project, the NCCoE implemented its own MUD file server by using an Apache web server. This file server signs and stores the MUD files along with their corresponding signature files for the IoT devices used in the project. Upon receiving a GET request for the MUD files and signatures, it serves the request to the MUD manager by using https.

6.2.3 MUD File

Using the MUD file maker component referenced above in Table 6-1, it is possible to create a MUD file with the following contents:

- internet communication class—access to cloud services and other specific internet hosts:
  - host: updateserver (hosted internally at the NCCoE)
    - protocol: TCP
    - direction-initiated: from IoT device
    - source port: any
    - destination port: 80
- controller class—access to **classes** of devices that are known to be controllers (could describe well-known services such as DNS or NTP):
  - host: mqttbroker (hosted internally at the NCCoE)
    - protocol: TCP
    - direction-initiated: from IoT device
    - source port: any
    - destination port: 1883
- local-networks class—access to/from **any** local host for specific services (e.g., Hypertext Transfer Protocol [http] or Hypertext Transfer Protocol Secure [https]):
  - host: any
    - protocol: TCP
    - direction-initiated: from IoT device
    - source port: any
    - destination port: 80
- my-controller class—access to controllers specific to this device:
  - controllers: null (to be filled in by the network administrator)
    - protocol: TCP
    - direction-initiated: from IoT device
    - source port: any
    - destination port: 80
- same-manufacturer class—access to devices of the same manufacturer:

    - same-manufacturer: null (to be filled in by the MUD manager]
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any
        - destination port: 80
- manufacturer class—access to devices of a specific manufacturer (identified by MUD URL):
    - manufacturer: devicetype (URL decided by the device manufacturer)
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any
        - destination port: 80

6.2.4 Signature File

According to the IETF MUD specification, “a MUD file MUST be signed using Cryptographic Message Syntax (CMS) as an opaque binary object.” The MUD file *(ciscopi2.json)* was signed with the OpenSSL tool by using the command described in the specification (which is in Volume C of this publication). A Premium Certificate, requested from DigiCert, was leveraged to generate the signature file *(ciscopi2.p7s).* Once created, the signature file is stored on the MUD file server.

6.2.5 DHCP Server

The DHCP server in the architecture is MUD-capable. In addition to dynamically assigning IP addresses, it recognizes the DHCP option (161) and extracts the MUD URL from the IoT device’s DHCP message. The MUD URL is provided to the MUD manager. The DHCP server is typically embedded in a router/switch. This project uses the DHCP server that is embedded in the Cisco Catalyst 3850-S.

Cisco IOS provides a basic DHCP server that is useful in small-/medium-business and home network environments, where centralized address management is not required. As described in the previous section, the DHCP server in this case is configured to allocate addresses for the test network, provide a default router, and configure a domain name server. It is **not** used to deliver MUD URLs to the MUD manager.

6.2.6 Link Layer Discovery Protocol

The Cisco Catalyst 3850-S switch also supports a MUD-capable version of the LLDP that provides the MUD URL in the LLDP TLV frame as an extension. When a MUD-capable IoT device uses LLDP to convey its MUD URL, the Cisco Catalyst 3850-S extracts the MUD URL from the LLDP frame and provides it to the MUD manager via a RADIUS message.

6.2.7 Router/Switch

This project uses the Cisco Catalyst 3850-S switch. The Cisco Catalyst 3850-S is an enterprise-class layer 3 switch capable of Universal PoE for digital building solutions. The optional PoE feature means it can be configured to supply power to capable devices over Ethernet through its ports. In addition to providing DHCP services, the switch acts as a broker for connected IoT devices for AAA through the FreeRADIUS server. The LLDP is enabled on ports that MUD-capable devices are plugged into to help facilitate recognition of connected IoT device features, capabilities, and neighbor relationships at layer 2. Additionally, an access session policy is configured on the switch to enable port control for multihost authentication and port monitoring. The combined effect of these switch configurations is a dynamic access list, which has been generated by the MUD manager, being active on the switch to permit or deny access to and from MUD-capable IoT devices. The version of the Cisco Catalyst switch used in this project is a proof-of-concept implementation that is intended to introduce advanced users and engineers to the MUD concept. Some protocol features are not present. These are described in Section 10.1, Findings.

6.2.8 Certificates

DigiCert’s CertCentral web-based platform allows provisioning and managing publicly trusted X.509 certificates for TLS and code signing as well as a variety of other purposes. After establishing an account, clients can log in, request, renew, and revoke certificates by using only a browser. Multiple roles can be assigned within an account, and a discovery tool can inventory all certificates within the enterprise. In addition to certificate-specific features, the platform offers baseline enterprise software-as-a-service capabilities, including role-based access control, Security Assertion Markup Language, single sign-on, and security policy management and enforcement. All account features come with full parity between the web portal and a publicly available API. For this implementation, two certificates were provisioned: a private TLS certificate for the MUD file server to support the https connection from the MUD manager to the MUD file server, and a Premium Certificate for signing the MUD files.

6.2.9 IoT Devices

This section describes the IoT devices used in the laboratory implementation. There are two distinct categories of devices: devices that can emit a MUD URL in compliance with the MUD specification, i.e., MUD-capable IoT devices; and devices that are not capable of emitting a MUD URL in compliance with the MUD specification, i.e., non-MUD-capable IoT devices.

6.2.9.1 MUD-Capable IoT Devices

The project used several MUD-capable IoT devices: NCCoE Raspberry Pi (devkit), u-blox C027-G35 (devkit), Samsung ARTIK 520 (devkit), Intel UP Squared Grove (devkit), Molex PoE Gateway, and Molex Light Engine. The NCCoE modified the devkits to simulate IoT devices. All of the MUD-capable IoT devices demonstrate the ability to emit a MUD URL as part of a DHCP transaction or LLDP message and to request and apply software updates.

6.2.9.1.1 Molex PoE Gateway and Light Engine

Molex developed this set of IoT devices. The PoE Gateway acts as a network end point and manages lights, sensors, and other devices. One of the devices managed by the PoE Gateway is a light engine that Molex provided.

6.2.9.1.2 NCCoE Raspberry Pi (Devkit)

The Raspberry Pi devkit runs the Raspbian 9 operating system. It is configured to include a MUD URL that it emits during a typical DHCP transaction. The NCCoE developed a Python script that allowed the Raspberry Pi to receive and process on and off commands by using the MQTT protocol, which were sent to the light-emitting diode (LED) bulb connected to the Raspberry Pi.

6.2.9.1.3 NCCoE u-blox C027-G35 (Devkit)

The u-blox C027-G35 devkit runs the Arm Mbed operating system. The NCCoE modified several of the Mbed-OS libraries to configure the devkit to include a MUD URL that it emits during a typical DHCP transaction. The u-blox devkit is also configured to initiate network connections to test network traffic throughout the MUD process.

6.2.9.1.4 NCCoE Samsung ARTIK 520 (Devkit)

The Samsung ARTIK 520 devkit runs the Fedora 24 operating system. It is configured to include a MUD URL that it emits during a typical DHCP transaction. The same Python script mentioned earlier was used to simulate a connected lock. This Python script allowed the ARTIK devkit to receive on and off commands by using the MQTT protocol.

6.2.9.1.5 NCCoE Intel UP Squared Grove (Devkit)

The Intel UP Squared Grove devkit runs the Ubuntu 16.04 LTS operating system. It is configured to include a MUD URL that it emits during a typical DHCP transaction. The same Python script mentioned earlier was used to simulate a connected lighting device. This allowed the UP Squared Grove devkit to receive on and off commands by using the MQTT protocol.

6.2.9.2 Non-MUD-Capable IoT Devices

The laboratory implementation also includes a variety of legacy, non-MUD-capable IoT devices that are not capable of emitting a MUD URL. These include cameras, mobile phones, lighting, a connected assistant, a printer, a baby monitor, a wireless access point, and a digital video recorder (DVR).

6.2.9.2.1 Cameras
The three cameras utilized in the laboratory implementation are produced by two different manufacturers. They stream video and audio either to another device on the network or to a cloud service. These cameras are controlled and managed by a mobile phone.

6.2.9.2.2 Mobile Phones
Two types of mobile phones are used for setting up, interacting with, and controlling IoT devices.

6.2.9.2.3 Lighting
Two types of connected lighting devices are used in the laboratory implementation. These connected lighting components are controlled and managed by a mobile phone.

6.2.9.2.4 Connected Assistant
A connected assistant is utilized in the laboratory implementation. The device demonstrates and tests the wide range of network traffic generated by a connected assistant.

6.2.9.2.5 Printer
A connected printer is connected to the laboratory network wirelessly to demonstrate connected printer usage.

6.2.9.2.6 Baby Monitor
A baby monitor with remote control plus video and audio capabilities is connected wirelessly to the laboratory network. This baby monitor is controlled and managed by a mobile phone.

6.2.9.2.7 Wireless Access Point
A connected wireless access point is used in the laboratory implementation to demonstrate the network activity and functionality of this type of device.

6.2.9.2.8 Digital Video Recorder
A connected DVR is connected to the laboratory implementation network. This is also controlled and managed by a mobile phone.

6.2.10 Update Server

The update server is designed to represent a device manufacturer or trusted third-party server that provides patches and other software updates to the IoT devices. This project used an NCCoE-hosted update server that provides faux software update files.

6.2.10.1 NCCoE Update Server

The NCCoE implemented its own update server by using an Apache web server. This file server hosts faux software update files to be served as software updates to the IoT device devkits. When the server receives an http request, it sends the corresponding faux update file.

6.2.10.2 Molex Update Agent

The process for updating the firmware on a Molex PoE Gateway is currently manual, with the firmware update taking place over the Constrained Application Protocol, UDP, and Trivial File transfer Protocol. The update process is initiated by an update agent on the local network connecting to the PoE Gateway and sending the firmware update information.

6.2.11 Unapproved Server

The NCCoE implemented its own unapproved server by using an Apache web server. This web server acts as an unapproved internet host, i.e., an internet host that is not explicitly approved in the MUD file. This was created to test the communication between a MUD-capable IoT device and an internet host that is not included in the MUD file and should thus be denied. To verify that the traffic filters were applied as expected, we tested communication to and from the unapproved server and the MUD-capable IoT device.

6.2.12 MQTT Broker Server

The NCCoE implemented an MQTT broker server by using the open-source tool Mosquitto. The server communicates messages among multiple clients. For this project, it allows mobile devices to set up with the appropriate application to communicate with the MQTT-enabled IoT devices in the build. The messages exchanged by the devices are on and off messages, which allow the mobile device to control the LED light on the IoT device.

6.2.13 IoT Device Discovery

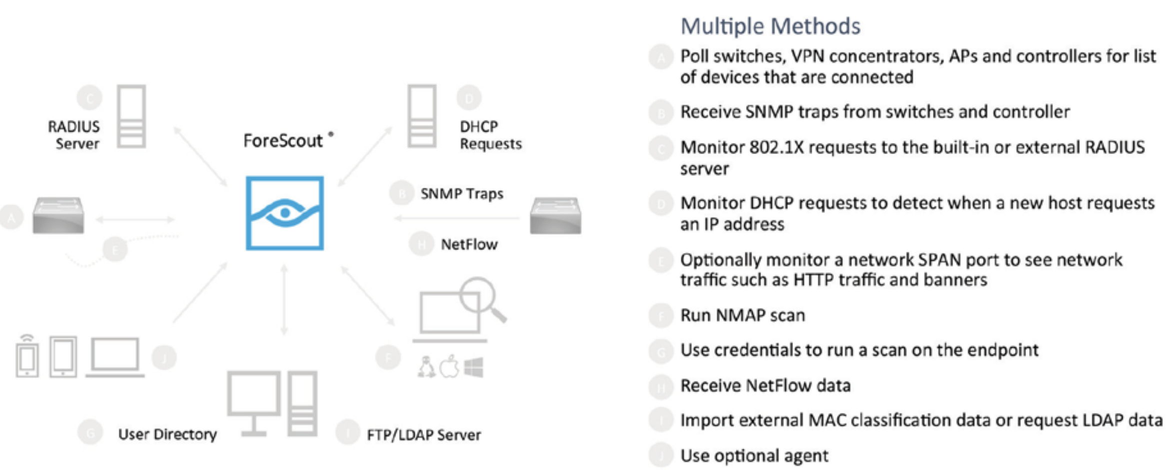
This project uses Forescout appliance and enterprise manager to provide an IoT device discovery service for the demonstration network. The Forescout appliance can discover, inventory, profile, and classify all attached devices to validate that the access that is being granted to each device is consistent with that device's type. Forescout can also continuously monitor the actions of these assets as they join and leave the network. While Forescout provides a wide range of data collection capabilities, items this project focuses on include:

- device information
  - device type
  - manufacturer
  - connection type
  - hardware information
  - MAC and IP addresses
  - operating system

    - network services
- network configuration
    - wired or wireless

The Forescout appliance detects IoT devices in real time as they connect to the network. It uses both passive monitoring and integration with the network infrastructure. As a device connects to the network, Forescout may learn about that device via a variety of different techniques to discover and classify it without requiring agents, as shown in Figure 6-1. The methods demonstrated in this project include Forescout passive discovery of devices by using switch polling, importation of MAC classification data, and TCP fingerprinting. Due to the passive nature of the device discovery, neither performance nor reliability of the IoT devices is impacted.

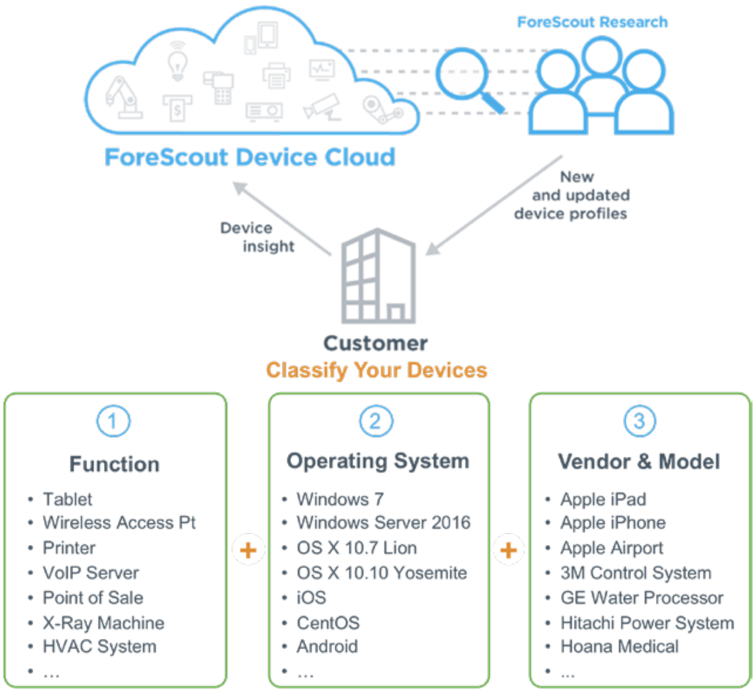
Figure 6-1 Methods the Forescout Platform Can Use to Discover and Classify IP-Connected Devices



Forescout is deployed as virtual appliances on the NCCoE laboratory network and managed by a single enterprise manager. After discovering IoT devices and collecting relevant information, classification is the next step.

To automatically classify discovered devices, the Forescout platform includes Forescout Device Cloud. Device Cloud allows users to benefit from crowdsourced device insight to auto-classify their devices, as shown in Figure 6-2. It also auto-classifies the devices by their type and function, operating system and version, and manufacturer and model. Users can leverage new and updated auto-classification profiles published by Forescout. In addition, they can create custom classification policies to auto-classify devices unique to their environments. At this writing, the Forescout appliance cannot identify whether an IoT device on the network is MUD-capable.

Figure 6-2 Classify IoT Devices by Using the Forescout Platform



6.3 Build Architecture

In this section we present the logical architecture of Build 1 relative to how it instantiates the reference architecture depicted in Figure 4-1. We also describe Build 1's physical architecture and present message flow diagrams for some of its processes.

6.3.1 Logical Architecture

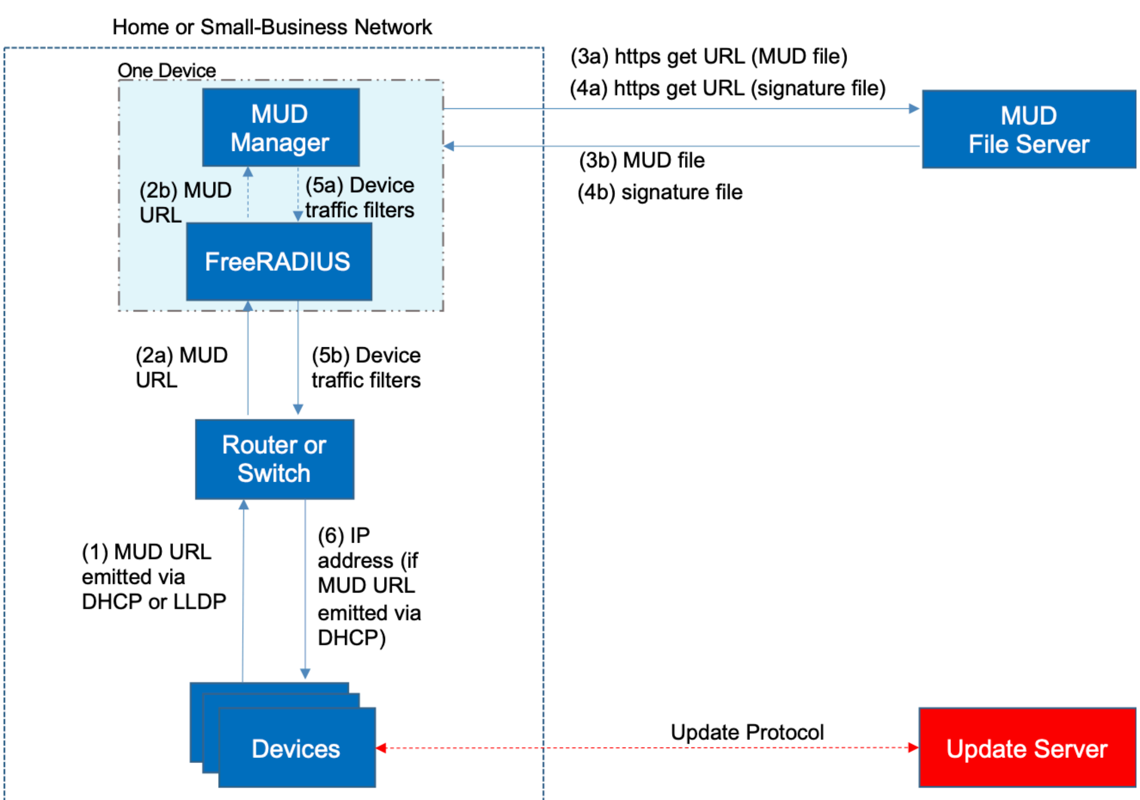
Figure 6-3 depicts the logical architecture of Build 1. Figure 6-3 uses numbered arrows to depict in detail the flow of messages needed to support installation of MUD-based access control rules for a MUD-capable device. Build 1 was designed with a single device serving as the MUD manager and FreeRADIUS server that interfaces with the Catalyst 3850-S switch over TCP/IP. It supports two mechanisms for MUD URL emission: DHCP and LLDP. Figure 6-3 depicts only the steps performed when using DHCP emission. The Catalyst 3850-S switch contains a DHCP server that is configured to extract MUD URLs from IPv4 DHCP transactions.

- Upon connecting a MUD-capable device, the MUD URL is emitted via either DHCP or LLDP (step 1).
- The Catalyst 3850-S switch sends the MUD URL to the FreeRADIUS server (step 2a); this is passed from the FreeRADIUS server to the MUD manager (step 2b).

- Once the MUD URL is received, the MUD manager uses this URL to fetch the MUD file from the MUD file server (step 3a); if successful, the MUD file server at the specified location will serve the MUD file (step 3b).
- Next, the MUD manager requests the signature file associated with the MUD file (step 4a) and upon receipt (step 4b) verifies the MUD file by using its signature file.
- Once the MUD file has been verified successfully, the MUD manager passes the device's traffic filters to the FreeRADIUS server (step 5a), which in turn sends the device's traffic filters to the router or switch, where they are applied (step 5b).
- The device is finally assigned an IP address (step 6).

Once the device's traffic filters are applied to the router or switch, the MUD-capable IoT device will be able to communicate with approved local hosts and internet hosts as defined in the MUD file, and any unapproved communication attempts will be blocked.

Figure 6-3 Logical Architecture–Build 1

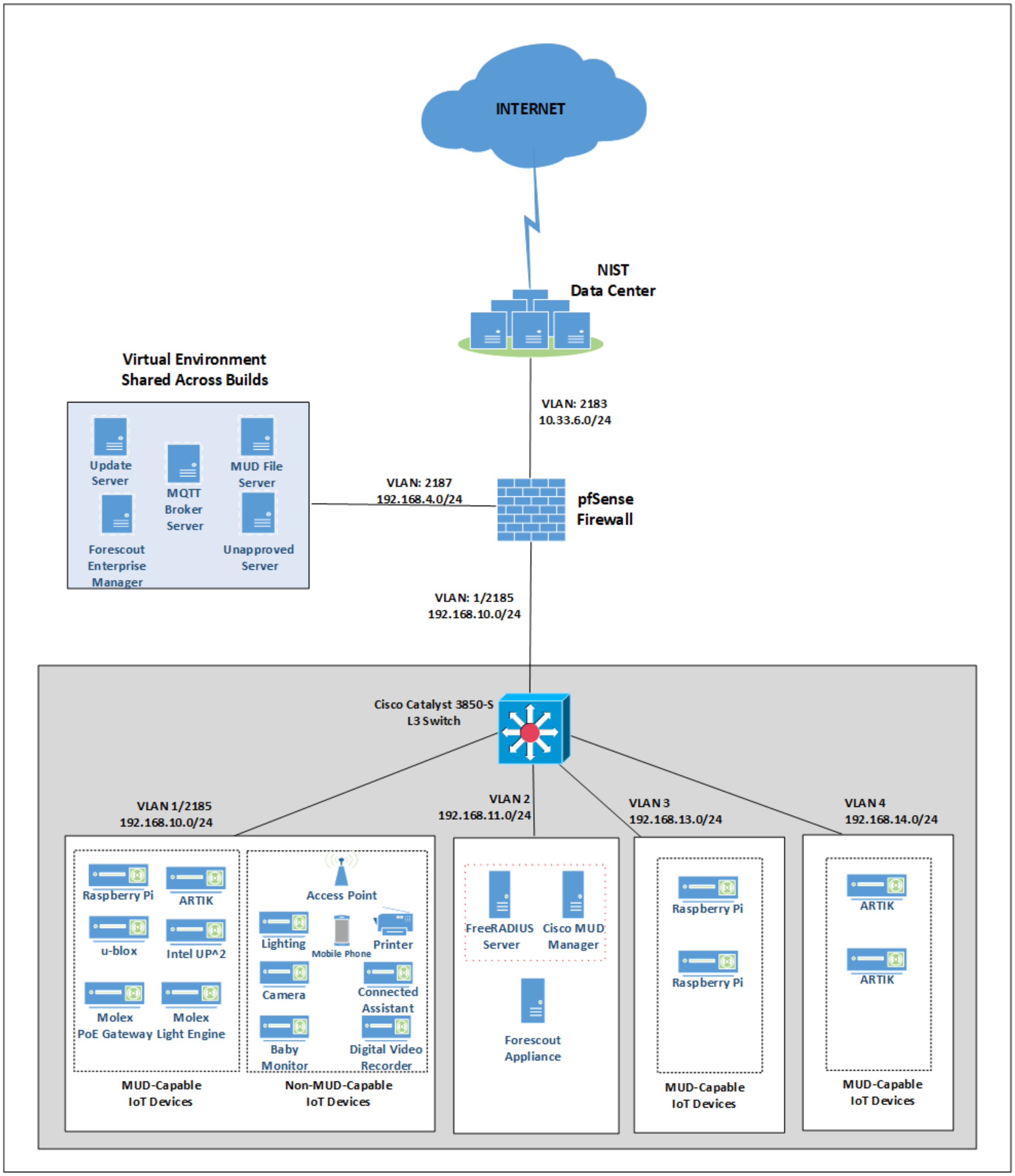


6.3.2 Physical Architecture

Figure 6-4 describes the physical architecture of Build 1. The Catalyst 3850-S switch is configured to host four VLANs. The first VLAN, VLAN 1, hosts many IoT devices. Three separate instances of DHCP servers are configured for VLANs 1, 3, and 4 to dynamically assign IPv4 addresses to each IoT device that connects to the switch on each of these VLANs. VLAN 2 is configured on the Catalyst switch to host the Cisco MUD manager, the FreeRADIUS server, and the Forescout appliance. VLAN 3 and VLAN 4 are configured to host IoT devices from the same manufacturer. Specifically, VLAN 3 hosts two Raspberry Pi devices, while VLAN 4 hosts two u-blox devices. The network infrastructure as configured utilizes the IPv4 protocol for communication both internally and to the internet.

In addition, Build 1 utilized a portion of the virtual environment that was shared across builds. Services hosted in this environment included an update server, MUD file server, MQTT broker, Forescout enterprise manager, and unapproved server.

1312 **Figure 6-4 Physical Architecture–Build 1**



A full description of Cisco's proof-of-concept MUD manager implementation is at https://github.com/CiscoDevNet/MUD-Manager. The Cisco MUD manager is built as a callout from FreeRADIUS and uses MongoDB to store policy information. The MUD manager is configured from a JSON file that will vary slightly based on the installation. This configuration file provides several static bindings and directives as to whether both egress and ingress ACLs should be applied, and it identifies the definition of the local network class on the network.

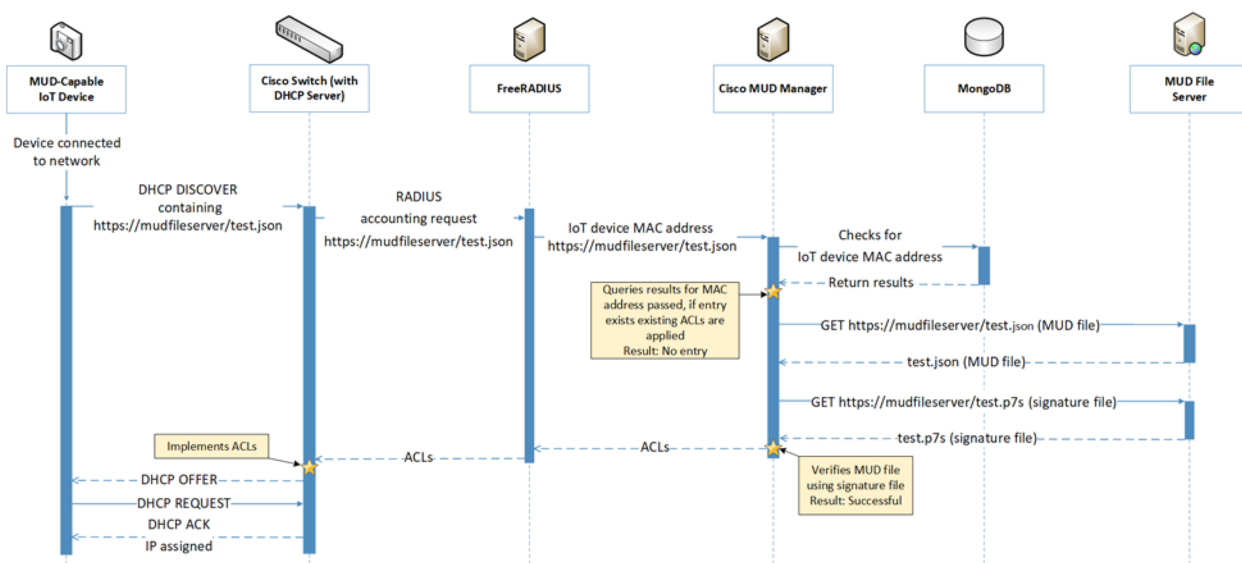
6.3.3 Message Flow

This section presents the message flows used in Build 1 during several different processes of note.

6.3.3.1 Installation of MUD-Based Access Control Rules for MUD-Capable Devices

Figure 6-5 shows the message flow of the process of installing access control rules for a MUD-capable IoT device that emits a MUD URL via DHCPv4.

Figure 6-5 MUD-Capable IoT Device MUD-Based ACL Installation Message Flow–Build 1



As shown in Figure 6-5, the message flow is as follows:

- A MUD-capable IoT device is connected to the network.
- The MUD-capable IoT device begins a DHCPv4 transaction in which DHCP option 161, the Internet Assigned Numbers Authority (IANA)-assigned value for MUD, is transmitted as part of a DHCP DISCOVER message. It is possible to transmit the option in both DISCOVER and REQUEST messages.

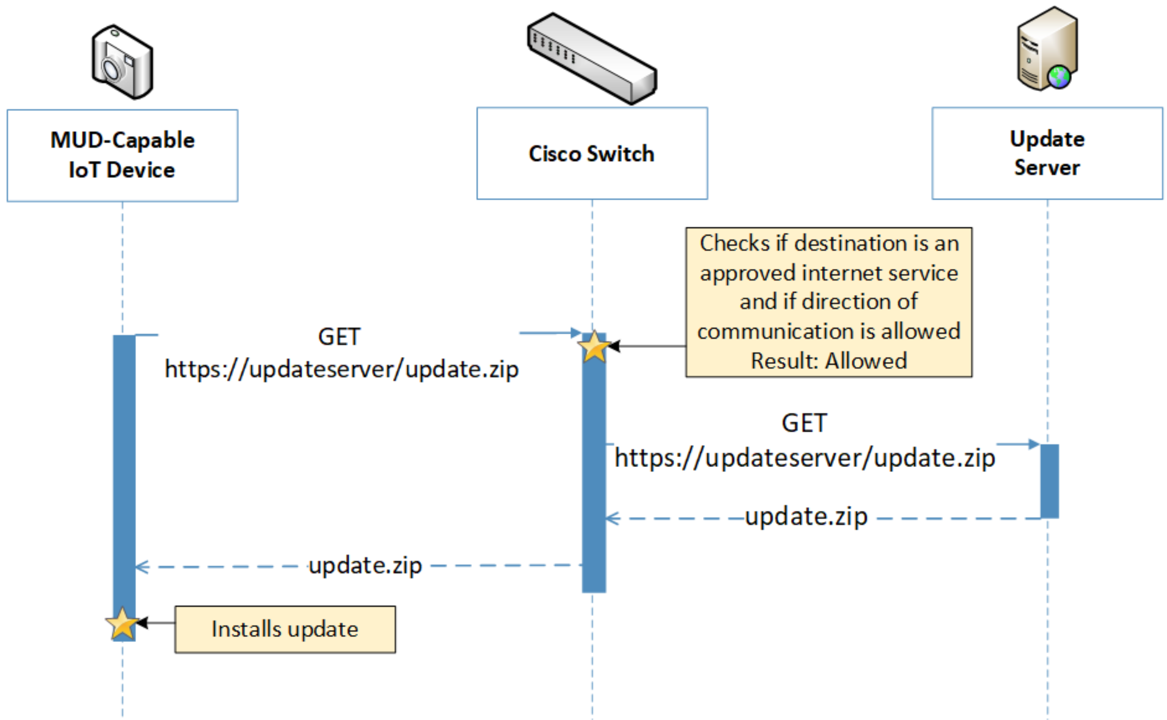
- The DHCP server on the Cisco switch recognizes that option and extracts the MUD URL from the DHCP message, which is sent from the switch to the FreeRADIUS server in the associated accounting request. From this point, the FreeRADIUS server sends the MAC address and MUD URL for the newly connected device to the MUD manager.
- Next, the MUD manager does a query for the MAC address in its database, searching for any cached MUD files associated with the MAC address and MUD URL. If an entry does not exist, as depicted in the figure, the MUD manager fetches the MUD file and signature file from the MUD file server.
- The MUD manager verifies the MUD file with the corresponding signature file and translates the contents into ACLs, which are passed through the FreeRADIUS server to the Cisco switch, where they are applied.
- The MUD-capable IoT device is assigned an IP address and is ready to be used on the network. When the MUD-capable IoT device is in use, access of all traffic to and from the IoT device is controlled by the Cisco switch, which will enforce the MUD ACLs for that device.

As an example, the subsections below address several different types of traffic that might apply to an IoT device. The message flow diagram in each subsection shows how this traffic would interact with Build 1’s infrastructure.

6.3.3.2 Updates

After a device has been permitted to connect to the home/small-business network, it should periodically check for updates. The message flow for updating the IoT device is shown in Figure 6-6 Update Process Message Flow–Build 1.

Figure 6-6 Update Process Message Flow–Build 1



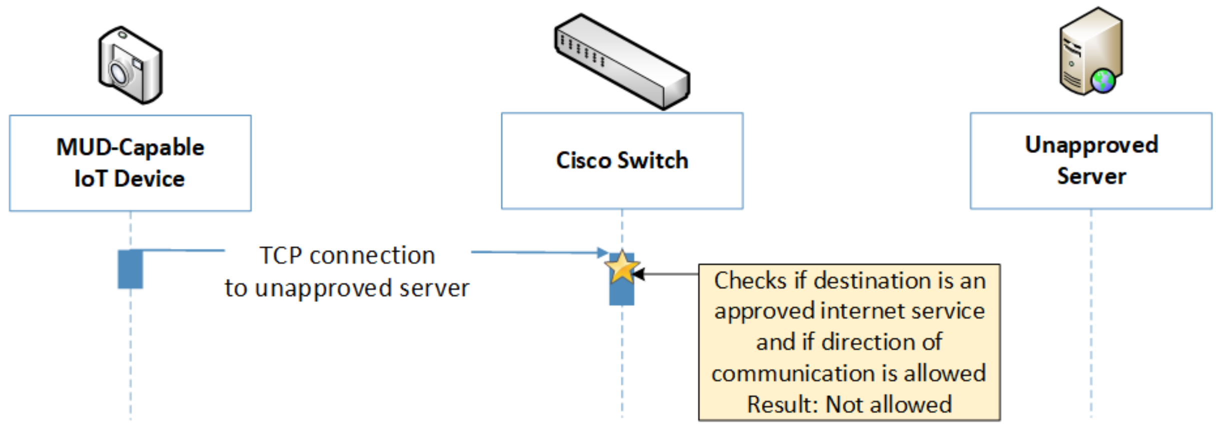
As shown in Figure 6-6 Update Process Message Flow–Build 1, the message flow is as follows:

- A MUD-capable IoT device initiates an https request to the update server.
- The Cisco switch checks its ACLs to determine if the destination and direction of communication should be allowed for the IoT device, and the switch allows the request after verification.
- The update server completes the process by sending the requested update package to the IoT device.

6.3.3.3 Prohibited Traffic

Figure 6-7 shows the message flows used to handle prohibited traffic in Build 1's infrastructure.

Figure 6-7 Prohibited Traffic Message Flow–Build 1



As shown in Figure 6-7, when an IoT device attempts to send traffic to an external domain, the message flow is as follows:

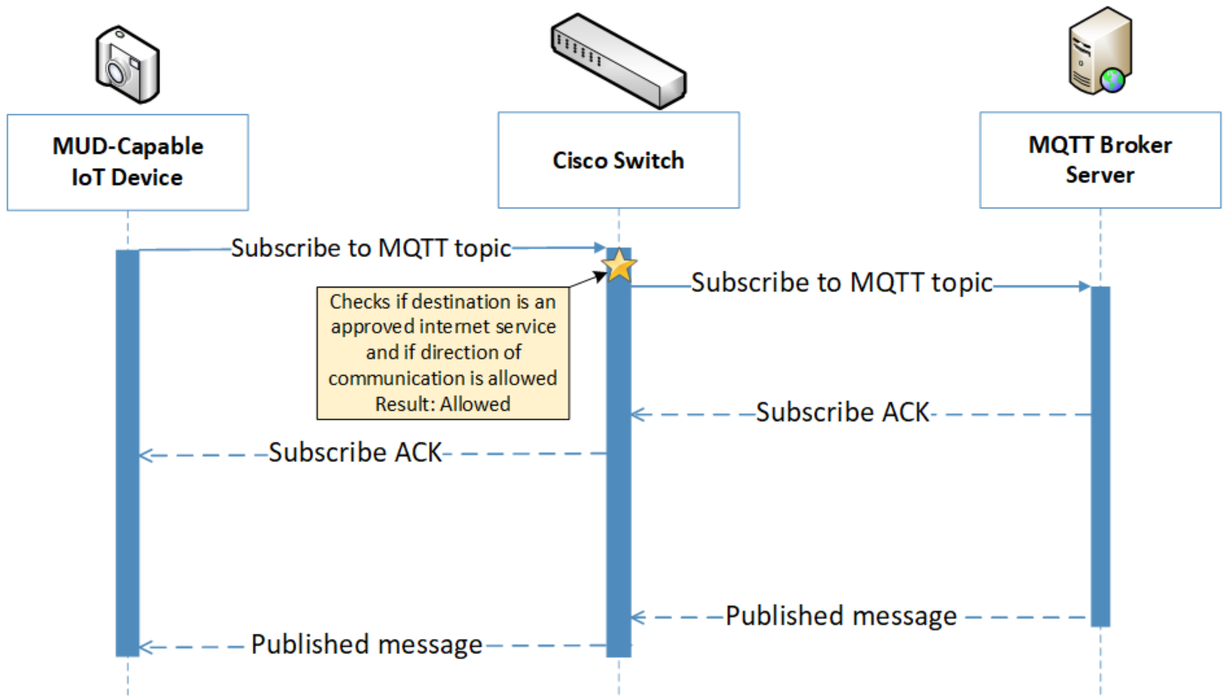
- The MUD-capable IoT device initiates a TCP request to an unapproved server.
- The Cisco switch checks its ACLs to determine if the destination and direction of communication should be allowed for the IoT device, and the switch blocks the unapproved communication.

At publication time, ingress access control was not yet supported in Build 1. That is, if an unapproved server attempts to send traffic to an IoT device on the local network, this traffic will currently not be blocked. However, responses from the IoT device will still be blocked. Specifics are in Section 10.1, Findings.

6.3.3.4 MQTT Protocol Example

Figure 6-8 shows the message flows used to handle MQTT communication in Build 1’s infrastructure.

Figure 6-8 MQTT Protocol Process Message Flow–Build 1



As shown in Figure 6-8, the message flow is as follows:

- The MUD-capable IoT device initiates a Subscribe message to the MQTT broker.
- The Cisco switch checks its ACLs to determine if the destination and direction of communication should be allowed for the IoT device, and the switch allows the Subscribe message after verification.
- The MQTT broker server sends a Subscribe Acknowledgement (ACK) to the IoT device.
- The MQTT broker server sends a Published message to the IoT device.

6.4 Functional Demonstration

A functional evaluation and a demonstration of Build 1 were conducted that involved two types of activities:

- Evaluation of conformance to the MUD RFC. We tested Build 1 to determine the extent to which it correctly implements basic functionality defined within the MUD RFC.
- Demonstration of additional (non-MUD-related) capabilities. It did not verify the example implementation's behavior for conformance to a standard or specification or any other expected set of capabilities; rather, it demonstrated advertised capabilities of the example implementation related to its ability to increase device and network security in ways that are

independent of the MUD RFC. These capabilities may provide security for both non-MUD-capable and MUD-capable devices. Examples of this type of activity include device discovery, attribute identification, and monitoring.

Table 6-2 summarizes the tests that we performed to evaluate Build 1's MUD-related capabilities, and Table 6-3 summarizes the exercises that we performed to demonstrate Build 1's non-MUD-related capabilities. Both tables list each test or exercise identifier, the test or exercise's expected and observed outcomes, and the applicable Cybersecurity Framework Subcategories and NIST SP 800-53 controls for which each test or exercise was designed to verify support. We detailed the tests and exercises listed in the table in a separate supplement for functional demonstration results. Boldface text highlights the gist of the information being conveyed.

Table 6-2 Summary of Build 1 MUD-Related Functional Tests

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
IoT-1	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed.	A **MUD-capable IoT device is configured to emit a MUD URL within a DHCP message.** The DHCP server extracts the MUD URL, which is sent to the MUD manager. The MUD manager requests the MUD file and signature from the MUD file server, and the MUD file server serves the MUD file to the MUD manager. The MUD file explicitly permits traffic to/from some internet services and hosts and implicitly denies traffic to/from all other internet services. **The MUD manager translates the MUD file information**	Upon connection to the network, the MUD-capable IoT device has its MUD **policy enforcement point (PEP) router/switch automatically configured according to the MUD file's route-filtering policies.**	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.AC-4: Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.	**into local network configurations that it installs on the router or switch that is serving as the MUD PEP for the IoT device.**		
IoT-2	**PR.AC-7:** Users, devices, and other assets are authenticated (e.g., single-factor, multifactor)	A MUD-capable IoT device is configured to emit a URL for a MUD	When the MUD-capable IoT device is connected	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	commensurate with the risk of the transaction (e.g., individuals' security and privacy risks and other organizational risks). **NIST SP 800-53 Rev. 4** AC-7, AC-8, AC-9, AC-11, AC-12, AC-14, IA-1, IA-2, IA-3, IA-4, IA-5, IA-8, IA-9, IA-10, IA-11	file, but the **MUD file server that is hosting that file does not have a valid TLS certificate. Local policy has been configured to ensure that if the MUD file for an IoT device is located on a server with an invalid certificate, the router/switch will be configured to deny all communication to/from the device.**	to the network, the MUD manager sends locally defined policy to the router/switch that handles whether to allow or block traffic to the MUD-capable IoT device. Therefore, the **MUD PEP router/switch will be configured to block all traffic to and from the IoT device.**	
IoT-3	**PR.DS-6:** Integrity-checking mechanisms are used to verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-capable IoT device is configured to emit a URL for a MUD file, but the **certificate that was used to sign the MUD file had already expired at signing. Local policy has been configured to ensure that if the MUD file for a device has a signature that was signed by a certificate that had already expired at the time of signature, the device's MUD PEP router/switch will be configured to deny all**	When the MUD-capable IoT device is connected to the network and the MUD file and signature are fetched, the MUD manager will detect that the MUD file's signature was created by using a certificate that had already expired at signing. According to local policy, the **MUD PEP will be configured to block all traffic**	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		communication to/from the device.	**to/from the device.**	
IoT-4	**PR.DS-6:** Integrity-checking mechanisms are used to verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-capable IoT device is configured to emit a URL for a MUD file, but the **signature of the MUD file is invalid. Local policy has been configured to ensure that if the MUD file for a device is invalid, the router/switch will be configured to deny all communication to/from the IoT device.**	When the MUD-capable IoT device is connected to the network, the MUD manager sends locally defined policy to the router/switch that handles whether to allow or block traffic to the MUD-capable IoT device. Therefore, the **MUD PEP router/switch will be configured to block all traffic to and from the IoT device.**	Pass
IoT-5	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating	Test IoT-1 has run successfully, meaning that the MUD PEP router/switch has been configured based on a **MUD file that permits traffic to/from some internet locations and implicitly denies traffic to/from all other internet locations.**	When the MUD-capable IoT device is connected to the network, its MUD PEP **router/switch will be configured to enforce the route filtering that is described in the device's MUD file** with respect to traffic being	Pass (for testable procedure, ingress cannot be tested)

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7		permitted to/from some internet locations, and traffic being implicitly blocked to/from all remaining internet locations.	
IoT-6	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by	Test IoT-1 has run successfully, meaning that the MUD PEP router/switch has been configured based on a **MUD file that permits traffic to/from some lateral hosts and implicitly denies traffic to/from all other lateral hosts.** (The MUD file does not explicitly identify the hosts as lateral hosts; it identifies classes of hosts to/from which traffic should be denied, where one or more hosts of this class happen to be lateral hosts.)	When the MUD-capable IoT device is connected to the network, its MUD PEP **router/switch will be configured to enforce the access control information that is described in the device's MUD file** with respect to traffic being permitted to/from some lateral hosts, and traffic being implicitly blocked to/from all remaining lateral hosts.	Pass (for testable procedure, ingress cannot be tested)

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.IP-3:** Configuration change control processes are in place. **PR.DS-3:** Assets are formally managed throughout removal, transfers, and disposition.			
IoT-7	**PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.DS-3:** Assets are formally managed throughout removal, transfers, and disposition. **NIST SP 800-53 Rev. 4** CM-8, MP-6	Test IoT-1 has run successfully, meaning that the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question. Next, have **the IoT device change DHCP state by explicitly releasing its IP address lease, causing the device's policy configuration to be removed from the MUD PEP router/switch.**	When the MUD-capable **IoT device explicitly releases its IP address lease,** the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch.	Failed
IoT-8	**PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.DS-3:** Assets are formally managed throughout removal, transfers, and disposition. **NIST SP 800-53 Rev. 4** CM-8, MP-6	Test IoT-1 has run successfully, meaning that the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question. Next, have **the IoT device change DHCP state by**	When the MUD-capable **IoT device's IP address lease expires,** the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch.	Failed (not supported)

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		waiting until the IoT device's address lease expires, causing the device's policy configuration to be removed from the MUD PEP router/switch.		
IoT-9	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CM-2, SI-4 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of	Test IoT-1 has run successfully, meaning the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question. The MUD file contains domains that resolve to multiple IP addresses. The MUD PEP router/switch should be configured to permit communication to or from all IP addresses for the domain.	A domain in the MUD file resolves to two different IP addresses. The MUD manager will create ACLs that permit the MUD-capable device to send traffic to both IP addresses. The MUD-capable device attempts to send traffic to each of the IP addresses, and the MUD PEP router/switch permits the traffic to be sent in both cases.	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-8, MP-6 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-8, MP-6 **PR.DS-2:** Data in transit is protected. **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10			
IoT-10	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5	A MUD-capable IoT device is configured to emit a MUD URL. Upon being connected to the network, its MUD file is retrieved, and the PEP is configured to enforce the policies specified in that MUD URL for that device. **Within 24**	Upon reconnection of the IoT device to the network, **the MUD manager does not contact the MUD file server. Instead, it uses the cached MUD file.** It translates this	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	ID.AM-3: Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10	**hours (i.e., within the cache-validity period for that MUD file), the IoT device is reconnected to the network.** After 24 hours have elapsed, the same device is reconnected to the network.	MUD file's contents into appropriate route-filtering rules and installs these rules onto the PEP for the IoT device. Upon reconnection of the IoT device to the network, after 24 hours have elapsed, the MUD manager does fetch a new MUD file.	

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.IP-3: Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.			
IoT-11	**ID.AM-1:** Physical devices and systems within the organization are inventoried.	A **MUD-capable IoT device can emit a MUD URL.** The device should leverage one of the specified manners for emitting a MUD URL.	Upon initialization, the MUD-capable IoT device broadcasts a DHCP message on the network, including at most one **MUD URL, in https scheme, within the DHCP transaction.** OR Upon initialization, the MUD-capable IoT device **emits a MUD URL as an LLDP extension.**	Pass

1401 In addition to supporting MUD, Build 1 demonstrates capabilities with respect to device discovery,
1402 attribute identification, and monitoring, as shown in Table 6-3.

Table 6-3 Non-MUD-Related Functional Capabilities Demonstrated

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
CnMUD-1	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CM-2, SI-4 **DE.CM-1:** The network is monitored to detect potential cybersecurity events. **NIST SP 800-53 Rev. 4** AC-2, AU-12, CA-7, CM-3, SC-5, SC-7, SI-4	A **visibility/monitoring component** is connected to the local IoT network. It **is configured to detect all devices connected to the network, discover attributes of these devices, categorize the devices, and monitor the devices** for any change of status.	Upon being connected to the network, the **visibility/monitoring component detects all connected devices, identifies their attributes** (e.g., type, IP address, OS), and categorizes them. **When an additional device is powered on, it is also detected, and its attributes identified. When a device is powered off, its change of status is detected.**	As expected

6.5 Observations

We observed the following limitations to Build 1 that are informing improvements to its current proof-of-concept implementation:

- MUD manager (version 3.0.1):
    - In previous versions (version 1.0), DNS resolution of internet host names in the MUD file was performed manually and remained static. Dynamic resolution of Fully Qualified Domain Names has since been added and is currently supported.
    - Translation and implementation of the model construct from the MUD file was not supported at testing time. However, this should be addressed in newer versions.
- Catalyst 3850-S switch (IOS version 16.09.02):
    - The MUD URL cannot be extracted when emitted via DHCPv6. Hence, the switch is only able to support MUD-capable IoT devices that use DHCPv4 and IPv4. This version of the switch does not yet support MUD-capable IoT devices when they are configured to use IPv6. IPv6 functionality is expected to be supported in the future.
    - The DHCP server does not notify the MUD manager of changes in DHCP state for MUD-capable IoT devices on the network. According to the MUD specification, the DHCP server should notify the MUD manager if the MUD-capable IoT device's IP address lease expires or has been released. However, this version of the DHCP server does not do so at testing time. This is expected to be addressed in the future.
    - Ingress dynamic ACLs (DACLs) (i.e., DACLs that pertain to traffic that is received from sources external to the network and directed to local IoT devices) are not supported with this version. Consequently, even if a MUD-capable IoT device's MUD file indicates that the IoT device is not authorized to receive traffic from an external domain, the DACL that is needed to prohibit that ingress traffic will not be configured on the switch. As a result, unless there is some other layer of security in place, such as a firewall that is configured to block this incoming traffic, the IoT device will still be able to receive incoming packets from that unauthorized external domain, which means it will still be vulnerable to attacks originating from that domain, despite the fact that the device's MUD file makes it clear that the device is not authorized to receive traffic from that domain. Because egress DACLs (i.e., DACLs that pertain to traffic that is sent from IoT devices to an external domain) are supported, however, even though packets that are sent from an outside domain are not stopped from being received at the IoT device, return traffic from the device to the external domain will be stopped. This means, for example, that if an attacker is able to get packets to an IoT device from an outside domain, it will not be possible for the attacker to establish a TCP connection with the device from that outside domain, thereby limiting the range of attacks that can be launched against the IoT device. This is expected to be addressed in the future.

7 Build 2

The Build 2 implementation uses a product from MasterPeace Solutions called Yikes! to support MUD. Yikes! is a commercial router/cloud service solution focused on consumer and small-business markets. It consists of a Yikes! router, a cloud service, and a mobile application that interfaces with the cloud service. In addition to supporting MUD, the Yikes! router and cloud service perform device discovery on the network and apply additional traffic rules to both MUD-capable and non-MUD-capable devices based on device manufacturer and model.

Also integrated with the Yikes! router in Build 2 is open-source software called Quad9 Active Threat Response (Q9Thrt), which builds on the Quad9 DNS service provided by Global Cyber Alliance. Q9Thrt enables the Yikes! router to take advantage of threat-signaling intelligence that is available through the Quad9 DNS service. Build 2 can use this information to block access, first to domains and, subsequently, to related IP addresses, that have been determined to be dangerous. This threat-signaling capability can protect both MUD-capable and non-MUD-capable devices. Build 2 also uses certificates from DigiCert.

7.1 Collaborators

Collaborators that participated in this build are described briefly in the subsections below.

7.1.1 MasterPeace Solutions

MasterPeace Solutions, Ltd. is a cybersecurity company in Columbia, Maryland, that focuses on serving federal intelligence community agencies. MasterPeace also operates the MasterPeace LaunchPad start-up studio, chartered with launching cyber-oriented technology product companies. A current LaunchPad start-up portfolio company, Yikes!, has developed a solution that includes both a MUD manager and cloud-based support for non-MUD IoT device security. Yikes! was created to bring automated enterprise-level security to consumer and small-business networks. Those networks are typically flat (unsegmented), predominantly connected via Wi-Fi-enabled devices, and managed by individuals who possess relatively little IT or cyber background compared with enterprise IT and cyber teams. Learn more about MasterPeace at https://www.masterpeaceltd.com.

7.1.2 Global Cyber Alliance

GCA is an international, cross-sector effort dedicated to eradicating cyber risk and improving our connected world. It achieves its mission by uniting global communities, implementing concrete solutions, and measuring the effect. GCA, a 501(c)3, was founded in September 2015 by the Manhattan District Attorney's Office, the City of London Police, and the Center for Internet Security. Learn more about GCA at https://www.globalcyberalliance.org.

7.1.3 DigiCert

See Section 6.1.2 for a description of DigiCert.

7.2 Technologies

Table 7-1 lists all of the products and technologies used in Build 2 and provides a mapping among the generic component term, the specific product used to implement that component, and the security Function Subcategories that the product provides. When applicable, both the Function Subcategories that a component provides directly and those that it supports but does not provide directly are listed and labeled as such. For rows in which the provides/supports distinction is not noted, the component directly provides all listed Categories. Refer to Table 5-1 for an explanation of the NIST Cybersecurity Framework Subcategory codes.

Table 7-1 Products and Technologies

Component	Product	Function	Cybersecurity Framework Subcategories
MUD manager	MasterPeace Yikes! router	Fetches, verifies, and processes MUD files from the MUD file server; configures router or switch with traffic filters to enforce firewall rules based on the MUD file	Provides PR.PT-3 Supports ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 DE.AE-1
MUD file server	MasterPeace-hosted Apache server	Hosts MUD files; serves MUD files to the MUD manager by using https	ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
MUD file maker	MUD file maker (https://www.mud-maker.org/)	YANG script GUI used to create MUD files	ID.AM-1
MUD file	A YANG model instance that has been serialized in JSON (RFC 7951). The manufacturer of a MUD-capable device creates that device's MUD file.	Specifies the communications that are permitted to and from a given device	Provides PR.PT-3 Supports ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
	MUD file maker (see previous row) can create MUD files. Each MUD file is also associated with a separate MUD signature file.		ID.AM-2 ID.AM-3
DHCP server	MasterPeace Yikes! router (Linksys WRT 3200ACM)	Dynamically assigns IP addresses; recognizes MUD URL in DHCP DISCOVER message; should notify MUD manager if the device's IP address lease expires or has been released	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
Router or switch	MasterPeace Yikes! router (Linksys WRT 3200ACM)	Provides MUD URL to MUD manager; gets configured by the MUD manager to enforce the IoT device's communication profile; performs per-device firewall rule enforcement	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
Certificates	DigiCert Premium Certificate	Used to sign MUD files and generate corresponding signature file	PR.AC-1 PR.AC-3 PR.AC-5 PR.AC-7
MUD-capable IoT device	Raspberry Pi Model 3B (devkit) Samsung ARTIK 520 (devkit) BeagleBone Black (devkit) NXP i.MX 8M (devkit)	Emits a MUD URL as part of its DHCP DISCOVER message; requests and applies software updates	ID.AM-1
Non-MUD-capable IoT device	Camera Mobile phones Connected lighting devices Connected assistant Printer	Acts as typical IoT devices on a network; creates network connections to cloud services	ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
	Digital video recorder		
Update server	NCCoE-hosted Apache server	Acts as a device manufacturer's update server that would communicate with IoT devices to provide patches and other software updates	PR.IP-1 PR.IP-3
Unapproved server	NCCoE-hosted Apache server	Acts as an internet host that has not been explicitly approved in a MUD file	DE.DP-3 DE.AM-1
IoT device discovery, categorization, and traffic policy enforcement	MasterPeace Yikes! router (Linksys WRT 3200ACM) and Yikes! cloud service	Discovers, classifies, and constrains traffic to/from IoT devices on network based on information such as DHCP header, MAC address, operating system, manufacturer, and model	ID.AM-1 PR.IP-1 DE.AM-1
Display and configuration of device information and traffic policies	MasterPeace Yikes! mobile application	Interacts with the Yikes! cloud to receive, display, and change information about the Yikes! router traffic policies and identification and categorization information about connected devices	ID.AM-1 PR.IP-1 DE.AM-1
Threat agent	GCA Quad9 threat agent, which is part of the open-source software Q9Thrt and is integrated into the Yikes! router	Monitors DNS traffic to/from devices on the local network and detects when domains are not resolved. When domains are not resolved, it queries the	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		Quad9 threat API regarding whether the domain is dangerous and, if so, what threat intelligence provider has flagged it as such. If a domain is determined to be dangerous, it notifies the Quad9 MUD manager of this threat.	
Threat-signaling MUD manager	GCA Quad9 MUD manager, which is part of the open-source software Q9Thrt and is integrated into the Yikes! router	Requests, receives, and parses the threat MUD file provided by the threat-signaling service's threat MUD file server, and applies its rules to create configurations to the Yikes! router's DNS service and its firewall rules that prohibit all devices from accessing the locations listed in the threat MUD file	ID.RA-1 ID.RA-2 ID.RA-3
Threat-signaling DNS services	GCA Quad9 DNS service	Receives input from several threat intelligence providers (including ThreatSTOP). Receives DNS resolution queries from local DNS service. For domains that are not known to be a threat, it simply resolves those domains to their IP address and provides this address to the request-	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		ing device. For domains that have been flagged as dangerous, it does not perform address resolution and instead returns a NULL response.	
Threat-signaling API	GCA Quad9 threat API	Receives queries from the threat-signaling agent on the local network regarding domains that were not resolved. If a domain was not resolved because it had been flagged as dangerous, it responds with the name of the threat intelligence provider that had flagged the domain as dangerous.	ID.RA-1 ID.RA-2 ID.RA-3
Threat MUD file server	ThreatSTOP threat MUD file server	Receives requests from the threat-signaling MUD manager on the local network for the threat MUD file corresponding to a domain that has been flagged as dangerous. Responds by providing the threat MUD file (and the MUD file's signature file) that is associated with the threat that has made this domain dangerous. This threat file will contain not just the domain and IP address	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		of the domain that the router had tried, unsuccessfully, to resolve; it will also include the list of all domains and IP addresses that are associated with the threat in question, i.e., all domains and IP addresses that are associated with this threat campaign.	
Threat MUD File	Threat file in MUD file format provided by ThreatSTOP listing all dangerous domains and IP addresses associated with any given threat	This is a file that has the exact same format as a MUD file, thus providing a standardized format for conveying the domains and IP addresses of all dangerous sites that are associated with a given threat and should therefore be blocked. Unlike a typical MUD file, however, this file does not contain usage description information regarding the permitted communication profile of some specific type of device. Instead, the information in this file is intended to be applied to the entire network (both MUD-capable and non-MUD-capable devices). Furthermore, it will list only external	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		sites to and from which traffic should be prohibited because the sites are associated with a given threat, not sites with which communication should be permitted, and it will not provide any rules regarding local network traffic that should be permitted or prohibited. Also, any given threat may be associated with a number of different domains and/or IP addresses. This threat file is designed to list all domains and IP addresses that are associated with any given threat that should be blocked. The file will also differ from a typical MUD file insofar as its mfg-name field will contain the name of the threat intelligence provider rather than the name of a device manufacturer, and its model-name field will typically contain the name of the threat that the file is associated with rather than model information about any IoT device.	

Each of these components is described more fully in the following sections.

7.2.1 MUD Manager

The MUD manager is a key component of the architecture. It fetches, verifies, and processes MUD files from the MUD file server. It then configures the router with firewall rules to control communications based on the contents of the MUD files. The Yikes! MUD manager is a logical component within the physical Yikes! router. The Yikes! router supports IoT devices that emit their MUD URLs via DHCP messages. When the MUD URL is emitted via DHCP, it is extracted from the DHCP message and provided to the MUD manager, which then retrieves the MUD file and signature file associated with that URL and configures the Yikes! router to enforce the IoT device's communication profile based on the MUD file. The router implements firewall rules for src-dnsname, dst-dnsname, my-controller, controller, same-manufacturer, manufacturer, and local-networks constructs that are specified in the MUD file. The system supports both lateral east/west protection and appropriate access to internet sites (north/south protection).

By default, Yikes! prohibits each device on the network from communicating with all other devices on the network unless explicitly permitted either by the MUD file or by local policy rules that are configurable within the Yikes! router.

The version of the Yikes! MUD manager used in this project is a prerelease implementation that is intended to introduce home and small-business network users to the MUD concept. It is intended to be a fully automated MUD manager implementation that includes all MUD protocol features.

7.2.2 MUD File Server

In the absence of a commercial MUD file server for use in this project, the NCCoE used a MUD file server hosted by MasterPeace that is accessible via the internet. This file server stores the MUD files along with their corresponding signature files for the IoT devices used in the project. Upon receiving a GET request for the MUD files and signatures, it serves the request to the MUD manager by using https.

7.2.3 MUD File

Using the MUD file maker component referenced above in Table 7-1, it is possible to create a MUD file with the following contents:

- internet communication class—access to cloud services and other specific internet hosts:
    - host: www.osmud.org
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any

        - destination port: 443
- controller class—access to **classes** of devices that are known to be controllers (could describe well-known services such as DNS or NTP):
    - host: www.getyikes.com
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any
        - destination port: 443
- local-networks class—access to/from **any** local host for specific services (e.g., http or https):
    - host: any
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any
        - destination port: 80
- my-controller class—access to controllers specific to this device:
    - controllers: null (to be filled in by the network administrator)
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any
        - destination port: 80
- same-manufacturer class—access to devices of the same manufacturer:
    - same-manufacturer: null (to be filled in by the MUD manager)
        - protocol: TCP
        - direction-initiated: from IoT device
        - source port: any
        - destination port: 80
- manufacturer class—access to devices of a specific manufacturer (identified by MUD URL):
    - manufacturer: Google (URL decided by the device manufacturer)
        - protocol: TCP
        - direction-initiated: from IoT device

- source port: any
- destination port: 80

7.2.4 Signature File

According to the IETF MUD specification, "a MUD file MUST be signed using CMS as an opaque binary object." All the MUD files in use (e.g., *yikesmain.json*) were signed with the OpenSSL tool by using the command described in the specification (detailed in Volume C of this publication). A Premium Certificate, requested from DigiCert, was leveraged to generate the signature file (e.g., *yikesmain.p7s*). Once created, the signature file is stored on the MUD file server.

7.2.5 DHCP Server

The DHCP server in the architecture is MUD-capable and, like the MUD manager, is a logical component within the Yikes! router. In addition to dynamically assigning IP addresses, it recognizes the DHCP option (161) and extracts the MUD URL from the IoT device's DHCP message. It then provides the MUD URL to the MUD manager. The DHCP server provided by the Yikes! router is useful in small-/medium-business and home network environments where centralized address management is not required.

7.2.6 Router/Switch

This build uses the MasterPeace Yikes! router. The Yikes! router is a customized original equipment manufacturer product, which at the time of this implementation is a preproduction product developed on a Linksys WRT 3200ACM router. It is a self-contained router, Wi-Fi access point, and firewall that communicates locally with Wi-Fi devices and wired devices. The Yikes! router initially isolates all devices connected to the router from one another. When devices connect to the router, the Yikes! router provides the device's DHCP header, MAC address, operating system, and connection characteristics to the Yikes! cloud service, which attempts to identify and categorize each device based on this information. The Yikes! router receives from the Yikes! cloud service rules for north/south and east/west filtering based on the Yikes! cloud processing (see Section 7.2.11) and any custom user settings that may have been configured in the Yikes! mobile application (see Section 7.2.12). These rules may apply to both MUD-capable and non-MUD-capable devices.

In addition to this category-based traffic policy enforcement that the Yikes! router provides for all devices, the Yikes! router also provides MUD support for MUD-capable IoT devices that emit MUD URLs via DHCP. Future work may be done to support MUD-capable devices that emit MUD URLs via X.509 or LLDP. The Yikes! router receives the MUD URL emitted by the device, retrieves the MUD file associated with that URL, and configures traffic filters (firewall rules) on the router to enforce the communication limitations specified in the MUD file for each device. The Yikes! router requires access to the internet to support secure API access to the Yikes! cloud service.

Last, the Yikes! router also provides integrated support for threat signaling by incorporating GCA Quad9 threat agent (see Section 7.2.13) and GCA Quad9 MUD manager (see Section 7.2.14) capabilities. Both the Quad9 threat agent and the Quad9 MUD manager are components of the open-source software Q9Thrt. See Section 7.3.1.3 for a description of Build 2's threat-signaling architecture and more information on Q9Thrt.

7.2.7 Certificates

DigiCert provisioned a Premium Certificate for signing the MUD files. The Premium Certificate supports the key extensions required to sign and verify Cryptographic Message Syntax (CMS) structures as required in the MUD specification. Further information about DigiCert's CertCentral web-based platform, which allows provisioning and managing publicly trusted X.509 certificates, is in Section 6.2.8.

7.2.8 IoT Devices

This section describes the IoT devices used in the laboratory implementation. There are two distinct categories of devices: devices that can emit a MUD URL in compliance with the MUD specification, i.e., MUD-capable IoT devices; and devices that are not capable of emitting a MUD URL in compliance with the MUD specification, i.e., non-MUD-capable IoT devices.

7.2.8.1 MUD-Capable IoT Devices

The project used several MUD-capable IoT devices: NCCoE Raspberry Pi (devkit), Samsung ARTIK 520 (devkit), BeagleBone Black (devkit), and NXP i.MX 8m (devkit). The NCCoE team modified the devkits to simulate MUD capability within IoT devices. All of the MUD-capable IoT devices demonstrate the ability to emit a MUD URL as part of a DHCP transaction and to request and apply software updates.

7.2.8.1.1 NCCoE Raspberry Pi (Devkit)

The Raspberry Pi devkit runs the Raspbian 9 operating system. It is configured to include a MUD URL that it emits during a typical DHCP transaction.

7.2.8.1.2 NCCoE Samsung ARTIK 520 (Devkit)

The Samsung ARTIK 520 devkit runs the Fedora 24 operating system. It is configured to include a MUD URL that it emits during a typical DHCP transaction.

7.2.8.1.3 NCCoE BeagleBone Black (Devkit)

The BeagleBone Black devkit runs the Debian 9.5 operating system. It is configured to include a MUD URL that it emits during a typical DHCP transaction.

7.2.8.1.4 NCCoE NXP i.MX 8m (Devkit)

The NXP i.MX 8m devkit runs the Yocto Linux operating system. The NCCoE modified a Wi-Fi start-up script on the device to configure it to emit a MUD URL during a typical DHCP transaction.

7.2.8.2 Non-MUD-Capable IoT Devices

The laboratory implementation also includes a variety of legacy, non-MUD-capable IoT devices that are not capable of emitting a MUD URL. These include cameras, mobile phones, connected lighting, a connected assistant, a printer, and a DVR.

7.2.8.2.1 Cameras

The three cameras utilized in the laboratory implementation are produced by two different manufacturers. They stream video and audio either to another device on the network or to a cloud service. These cameras are controlled and managed by a mobile phone.

7.2.8.2.2 Mobile Phones

Two types of mobile phones are used for setting up, interacting with, and controlling IoT devices.

7.2.8.2.3 Lighting

Two types of connected lighting devices are used in the laboratory implementation. These connected lighting components are controlled and managed by a mobile phone.

7.2.8.2.4 Connected Assistant

A connected assistant is utilized in the laboratory implementation. The device demonstrates and tests the wide range of network traffic generated by a connected assistant.

7.2.8.2.5 Printer

A connected printer is connected to the laboratory network wirelessly to demonstrate connected printer usage.

7.2.8.2.6 Digital Video Recorder

A connected DVR is connected to the laboratory implementation network. This is also controlled and managed by a mobile phone.

7.2.9 Update Server

The update server is designed to represent a device manufacturer or trusted third-party server that provides patches and other software updates to the IoT devices. This project used an NCCoE-hosted update server that provides faux software update files.

7.2.9.1 NCCoE Update Server

The NCCoE implemented its own update server by using an Apache web server. This file server hosts faux software update files to be served as software updates to the IoT device devkits. When the server receives an http request, it sends the corresponding faux update file.

7.2.10 Unapproved Server

As with Build 1, the NCCoE implemented and used its own unapproved server for Build 2. Details are in Section 6.2.11.

7.2.11 IoT Device Discovery, Categorization, and Traffic Policy Enforcement–Yikes! Cloud

The Yikes! cloud uses proprietary techniques and machine learning to analyze information about each device that is provided to it by the Yikes! router. The Yikes! cloud uses the DHCP header, MAC address, operating system, and connection characteristics of devices to automatically classify each device, including make, model, and Yikes! device category. Yikes! has a comprehensive list of categories that includes these examples:

- mobile: phone, tablet, e-book, connected watch, wearable, car
- home and office: computer, laptop, printer, IP phone, scanner
- connected home: IP camera, connected device, connected plug, light, voice assistant, thermostat, doorbell, baby monitor
- network: router, Wi-Fi extender
- server: network attached storage, server
- engineering: Raspberry Pi, Arduino

The Yikes! cloud then uses the Yikes! category to define specific east/west rules for that device and every other device on the Yikes! router's network. It also looks up the device in the Yikes! proprietary IoT device library, and, if available, provides specialized north/south filtering rules for that device. The east/west and north/south rules are then configured on the Yikes! router for local enforcement.

The Yikes! cloud also provides information about the device, whether it is MUD-capable, its categorization, and filtering rules to the Yikes! mobile application (see Section 7.2.12). This information is presented to the user in a graphical user interface, and the user can make specific changes. These changes are also configured on the Yikes! router for enforcement.

7.2.12 Display and Configuration of Device Information and Traffic Policies–Yikes! Mobile Application

Yikes! also provides a mobile application for additional capabilities, which at publication time was accessed through a web user interface (UI). The Yikes! mobile application allows users further fine-grained device-filtering control. The Yikes! mobile application interacts with the Yikes! cloud to receive and display information about the traffic policies that are configured on the Yikes! router as well as identification and categorization information about devices connected to the network. The Yikes!

mobile application enables device information that is populated automatically by the Yikes! cloud to be overridden, and it enables users to configure traffic policies to be enforced by the router.

7.2.13 Threat Agent

Build 2 has a threat-signaling agent integrated into the Yikes! router. This threat-signaling agent is part of the open-source software called Q9Thrt, which builds on and extends the Quad9 DNS service provided by GCA. More information on Q9Thrt is at https://github.com/osmud/q9thrt.

7.2.13.1 GCA Quad9 Threat Agent

The GCA Quad9 threat agent monitors DNS traffic to/from devices on the local network and detects when domains are not resolved by the Quad9 DNS service. When a domain is not resolved, it could mean one of two things: Either the domain has been flagged as potentially unsafe, or the domain does not exist (perhaps because it was mistyped, for example). The Quad9 threat agent eavesdrops on DNS responses that are sent from the Quad9 DNS service in the cloud to the Yikes! router's local DNS services. If the Quad9 threat agent detects a null response, it queries the Quad9 threat API to inquire as to whether the domain is dangerous and, if so, what threat intelligence provider has flagged it as such. If it receives a response indicating that a domain has been determined to be unsafe, it informs the Quad9 MUD manager (see Section 7.2.18) component (which is also integrated into the Yikes! router).

7.2.14 Threat-Signaling MUD Manager

Build 2 has a second MUD manager integrated into the Yikes! router that is designed to retrieve and parse the threat MUD file (see Section 7.2.18) retrieved from the threat intelligence provider. This threat-signaling MUD manager is part of the open-source software called GCA Q9Thrt, which builds on and extends the Quad9 DNS service provided by GCA. More information on Q9Thrt may be found at https://github.com/osmud/q9thrt.

7.2.14.1 GCA Quad9 MUD Manager

The GCA Quad9 MUD manager retrieves and parses threat MUD files. Threat MUD files are files that are written in MUD file format that list the domains and IP addresses of locations on the internet that have been determined to be unsafe and should be blocked because they are associated with a known threat. When the Quad9 threat agent (which is also integrated into the Yikes! router) learns that a threat has been found, it informs the Quad9 MUD manager and provides the Quad9 MUD manager with the URL of the threat MUD file. The Quad9 MUD manager uses https to request the threat MUD file and the threat MUD file's signature file. Assuming the signature file indicates that the threat MUD file is valid, the Quad9 MUD manager parses the threat MUD file and uses the threat MUD file rules to configure both the firewall and the local DNS services in the Yikes! router. It configures the firewall to prohibit all devices from accessing the domains and IP addresses listed in the threat MUD file, and it configures the local DNS services to return null responses when asked to resolve domain names listed in the threat MUD file.

7.2.15 Threat-Signaling DNS Services

Build 2 accesses external DNS services that receive input from several internet threat intelligence providers and are thus able to respond to domain name resolution requests for unsafe domains by signaling that the requested domain is potentially unsafe. These DNS services are provided by GCA.

7.2.15.1 GCA Quad9 DNS Service

GCA Quad9 DNS service receives input from several threat intelligence providers, making them aware of which domains have been determined to be unsafe. One of the threat intelligence providers that provides input to Quad9 DNS service is ThreatSTOP. For domains that are not known to be a threat, Quad9 DNS service behaves like any other DNS service would by resolving those domain names to their IP address(es) and providing those addresses to the requesting device. For domains that have been flagged as dangerous, however, Quad9 DNS service does not perform domain name resolution; instead, it returns a null response to the requesting device.

7.2.16 Threat-Signaling API

Build 2 accesses an external threat-signaling API that, when queried regarding specific domain names, responds by indicating whether the domain has been determined to be unsafe and, if so, the name of the threat intelligence provider responsible for the threat information. This threat-signaling API is provided by GCA.

7.2.16.1 GCA Quad9 Threat API

When a device on the local network makes a DNS request for a domain that does not get resolved, this means either that the domain does not exist or that it is unsafe. To determine which is the case for any given domain, the Quad9 threat agent on the Yikes! router queries the Quad 9 Threat API regarding that domain. If the domain is considered unsafe, the Quad9 threat API responds with the name of the threat intelligence provider that had flagged the domain as dangerous and other information that is needed to retrieve the associated threat MUD file.

7.2.17 Threat MUD File Server

Build 2 accesses an external threat MUD file server containing threat MUD files (see Section 7.2.18) for threats that a threat intelligence provider has identified and documented. The threat MUD file server used in Build 2 hosts threat MUD files provided by the threat intelligence provider ThreatSTOP.

7.2.17.1 ThreatSTOP Threat MUD File Server

When the Quad9 MUD manager on the Yikes! router is informed by the Quad9 threat agent that a threat has been found, the Quad9 MUD manager contacts the ThreatSTOP threat MUD file server to retrieve the threat MUD file associated with that threat. This threat MUD file server hosts threat MUD files (see Section 7.2.18) for threats that ThreatSTOP has identified and documented. When it receives a request from the Quad9 MUD manager for a threat file corresponding to a domain, the ThreatSTOP threat MUD file server responds by providing the threat file that is associated with the threat that has made this domain unsafe. This threat file will contain not just the domain and IP address of the domain that the router had tried unsuccessfully to resolve; it will also include all domains and IP addresses that are associated with the threat in question.

7.2.18 Threat MUD File

Build 2 uses threat MUD files provided by the threat intelligence provider ThreatSTOP. Threat MUD files have the same format as MUD files, thus providing a standardized format for conveying the domains and IP addresses of all dangerous sites that are associated with a given threat and should therefore be blocked. Unlike a typical MUD file, however, a threat MUD file does not contain manufacturer usage description information regarding the communication profile of some specific type of device. Instead, the information in this file is intended to be applied to the entire network (both MUD-capable and non-MUD-capable devices). Furthermore, the threat MUD file will list only external sites to and from which traffic should be prohibited because the sites are associated with a given threat, not sites with which communication should be permitted, and it will not provide any rules regarding local network traffic that should be permitted or prohibited. Also, any given threat may be associated with several different domains and/or IP addresses. The threat MUD file is designed to list all domains and IP addresses that are associated with any given threat that should be blocked. The file will also differ from a typical MUD file insofar as its mfg-name field will typically contain the name of the threat intelligence provider rather than the name of a device manufacturer, and its model-name field will typically contain the name of the threat that the file is associated with rather than model information about a particular IoT device.

7.3 Build Architecture

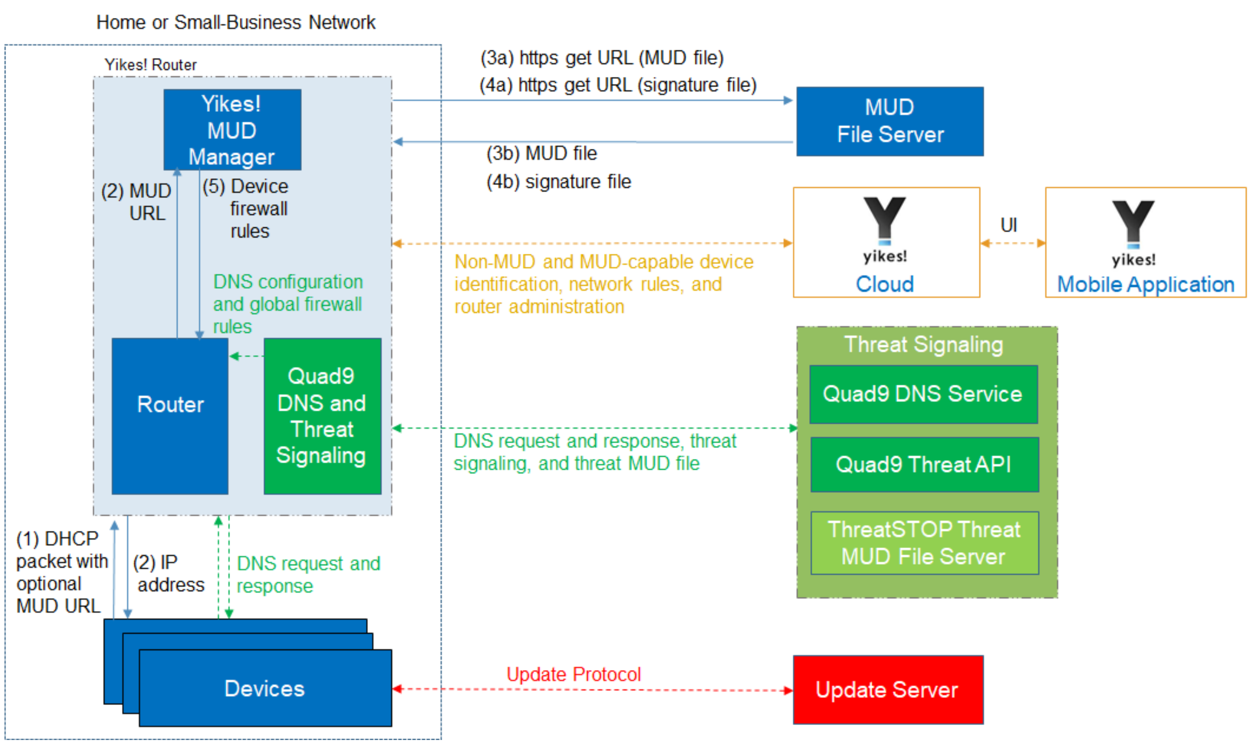
In this section we present the logical architecture of Build 2 relative to how it instantiates the reference architecture depicted in Figure 4-1. We also describe Build 2’s physical architecture and present message flow diagrams for some of its processes.

7.3.1 Logical Architecture

Figure 7-1 depicts the logical architecture of Build 2. Figure 7-1 uses numbered arrows to depict in detail the flow of messages needed to support installation of MUD-based access control rules for a MUD-capable device. The other key aspects of the Build 2 architecture (i.e., the Yikes! cloud, the Yikes! mobile application, threat signaling, and the update server) are depicted but not described in the same depth as MUD.

Yikes! is designed to run as a router with a connection to the Yikes! cloud and to be managed via the Yikes! mobile application. The Yikes! cloud provides traffic rules to the Yikes! router that apply to devices based on device category. The Yikes! router also supports threat-signaling capabilities that enable it to refrain from connecting to domains that threat intelligence services have flagged as potentially dangerous. The logical architecture for Build 2 also includes the notion of ensuring that all IoT devices can access update servers so they can remain up-to-date with the latest security patches. MUD, Yikes! cloud, and threat-signaling support are each described in their respective subsections below.

Figure 7-1 Logical Architecture—Build 2



7.3.1.1 MUD Capability

As shown in Figure 7-1, the Yikes! router includes integrated support for MUD in the form of a Yikes! MUD manager component and a MUD-capable DHCP server (not depicted). Support for MUD also requires access to a MUD file server that hosts MUD files for the MUD-capable IoT devices being connected to the network.

The Yikes! router currently supports DHCP as the mechanism for MUD URL emission. It contains a DHCP server that is configured to extract MUD URLs from IPv4 DHCP transactions.

As shown in Figure 7-1, the flow of messages needed to support installation of MUD-based access control rules for a MUD-capable device is as follows:

- Upon connecting a MUD-capable device, the MUD URL is emitted via DHCP (step 1).
- The Yikes! DHCP server on the router receives the request from the device and assigns it an IP address (step 2).
- At the same time, the DHCP server sends the MUD URL to the Yikes! MUD manager (step 2).
- Once the MUD URL is received, the MUD manager uses it to fetch the MUD file from the MUD file server (step 3a); if successful, the MUD file server at the specified location will serve the MUD file (step 3b).
- Next, the MUD manager requests the signature file associated with the MUD file (step 4a) and upon receipt (step 4b) verifies the MUD file by using its signature file.
- Assuming the MUD file has been verified successfully, the MUD manager translates the traffic rules that are in the MUD file into firewall rules that it installs onto the Yikes! router (step 5). Once the firewall rules are installed on the router, the MUD-capable IoT device will be able to communicate with approved local hosts and internet hosts as defined in the MUD file, and any unapproved communication attempts will be blocked.

7.3.1.2 Yikes! Cloud Capability

The Yikes! cloud includes the ability to identify and categorize both MUD-capable and non-MUD-capable devices that join the network, and it serves as the repository of traffic policies that can be applied to categories of devices regardless of whether those devices are MUD-capable. The Yikes! router communicates with the Yikes! cloud via a secure API. This communication is required for the router to send information related to the network to the Yikes! cloud service as well as to receive network rules and router administration from the Yikes! cloud. Network rules and router administration are configured through the Yikes! mobile application.

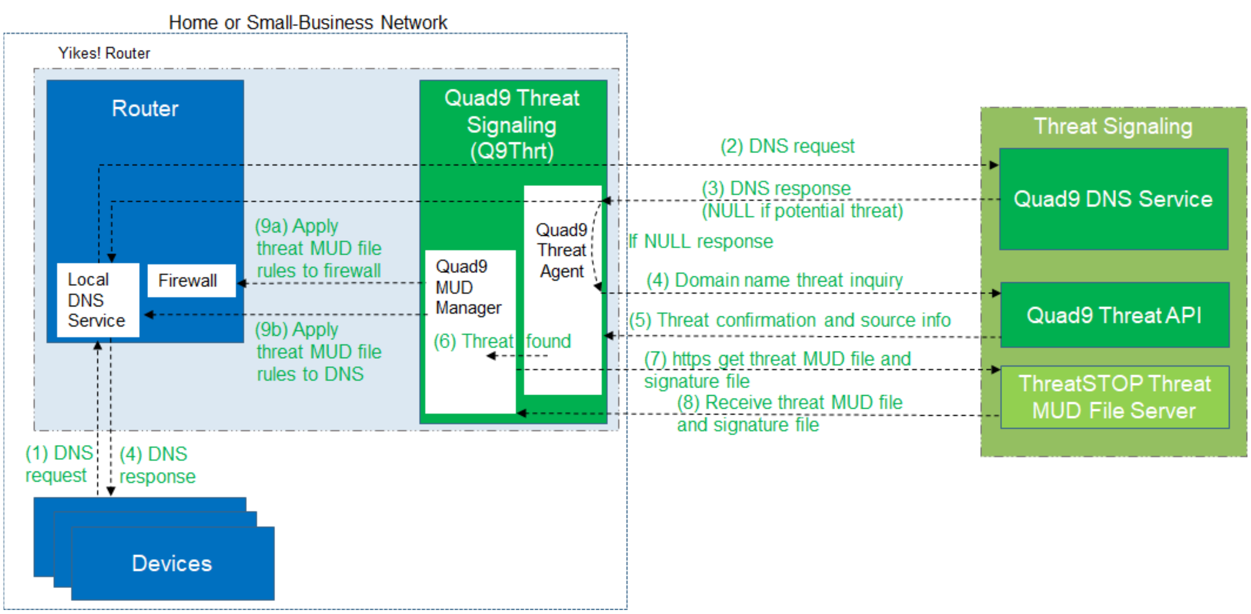
It is possible that both Yikes! cloud traffic policies and MUD file traffic policies could both apply to any given device in the network. For any given device, if these policies conflict, MUD file policies are given precedence over Yikes! traffic policies. If the policies do not conflict, they are both applied to the device. If a device is not MUD-capable, the Yikes! cloud policies that apply to it will be applied. If a device is MUD-capable but its MUD file is not applied (because, for example, the TLS certificate of the MUD file server is not valid or the MUD file is determined to be invalid), the Yikes! cloud rules that apply to the MUD-capable device will still be applied.

7.3.1.3 Threat-Signaling Capability

Build 2 integrates a threat-signaling capability that protects both MUD-capable and non-MUD-capable devices from the latest cybersecurity threats that have been detected by threat intelligence services. It prevents devices from accessing external domains and IP addresses that are associated with known current cybersecurity threats.

Figure 7-2 depicts a detailed view of Build 2's threat-signaling architecture. As shown, GCA's Quad9 threat agent and Quad9 MUD manager (which are both part of Q9Thrt) are integrated into the Yikes! router to support threat signaling. Additionally, the Yikes! router requires the use of several external components to support threat signaling: Quad9 DNS service, which receives threat information feeds from a variety of threat intelligence services; Quad9 threat API, which confirms a threat as well as information regarding how to find the threat MUD file for that threat; and the ThreatSTOP threat MUD file server, which provides the threat MUD file for the threat.

Figure 7-2 Threat-Signaling Logical Architecture–Build 2



The messages that are exchanged among architectural components to support threat signaling are depicted by arrows and numbered in sequence in Figure 7-2. The result of this message flow is to protect a local device from connecting to a domain that has been identified as unsafe by a threat intelligence service from which Quad9 DNS service receives information which, in this case, is ThreatSTOP.

As depicted in Figure 7-2, the steps are as follows:

- A local device (which may or may not be an IoT device and may or may not be MUD-capable) sends a DNS resolution requests to its local DNS service, which is hosted on the Yikes! router (step 1).
- If the local DNS service cannot resolve the request itself, it will forward the request to the Quad9 DNS service (step 2).
- The Quad9 DNS service will return a DNS response to the Yikes! router's local DNS service. The Quad9 DNS service receives input from several threat intelligence providers (not depicted in the diagram), so it is aware of whether the domain in question has been identified to be unsafe. If the domain has not been identified as unsafe, the Quad9 DNS service will respond with the IP address(es) corresponding to the domain (as would any normal DNS service). If the domain has been flagged as unsafe, however, the Quad9 DNS service will not resolve the domain. Instead, it will return an empty (null) DNS response message to the local DNS service (step 3).
- The local DNS service will forward the DNS response to the device that originally made the DNS resolution request (step 4).
- Meanwhile, the Quad9 Threat Agent that is running on the Yikes! router monitors all DNS requests and responses. When it sees a domain that does not get resolved, it sends a query to the Quad9 Threat API asking whether the domain is dangerous and, if so, what threat intelligence provider had flagged it as such and with what threat it is associated (step 4).
- The Quad9 Threat API responds with this information, which, in this case, informs the threat agent that the domain is indeed dangerous and if it wants more information about the blocked domain, it should contact ThreatSTOP (a threat intelligence provider) and request a particular threat MUD file. This threat MUD file will list domains and IP addresses that should be blocked because they are all associated with the same threat campaign as this threat (step 5).
- The Quad9 threat agent provides this information to the Quad9 MUD manager (step 6).
- The Quad9 MUD manager requests the threat MUD file (and the threat MUD file's signature file) from the ThreatSTOP threat MUD file server (step 7).
- The Quad9 MUD manager receives the threat MUD file (and the threat MUD file's signature file) from the ThreatSTOP threat MUD file server and uses the signature file to verify that the threat MUD file is valid (step 8).
- Assuming the threat MUD file is valid, the Quad9 MUD manager uses the threat MUD file to configure the router's firewall to block all domains and IP addresses listed in this threat MUD file (step 9a).
- The Quad9 MUD manager also configures the router's local DNS services to provide empty responses for DNS requests that are made for all domain names that are listed in the threat MUD file (step 9b).

Threat-signaling rules have higher precedence than MUD rules, which, in turn, have higher precedence than Yikes! category rules. This means that if a domain is flagged as dangerous by threat-signaling intelligence, none of the devices on the local network will be permitted to communicate with it—even MUD-capable devices whose MUD files list that domain as permissible.

Threat-signaling rules time out after 24 hours, at which time the firewall rules associated with those rules are removed from the router. If, after 24 hours, a device tries to connect to that domain but is still considered dangerous, the firewall rules will no longer be in place in the router to prevent access to the domain. However, when the device attempts to access the domain, the same DNS resolution process as depicted in Figure 7-2 will be performed all over again: when the device requests resolution of the domain name, the Quad9 DNS service will return an empty DNS response message, and the threat MUD file for that domain will be retrieved and its rules installed on the router firewall for another 24 hours.

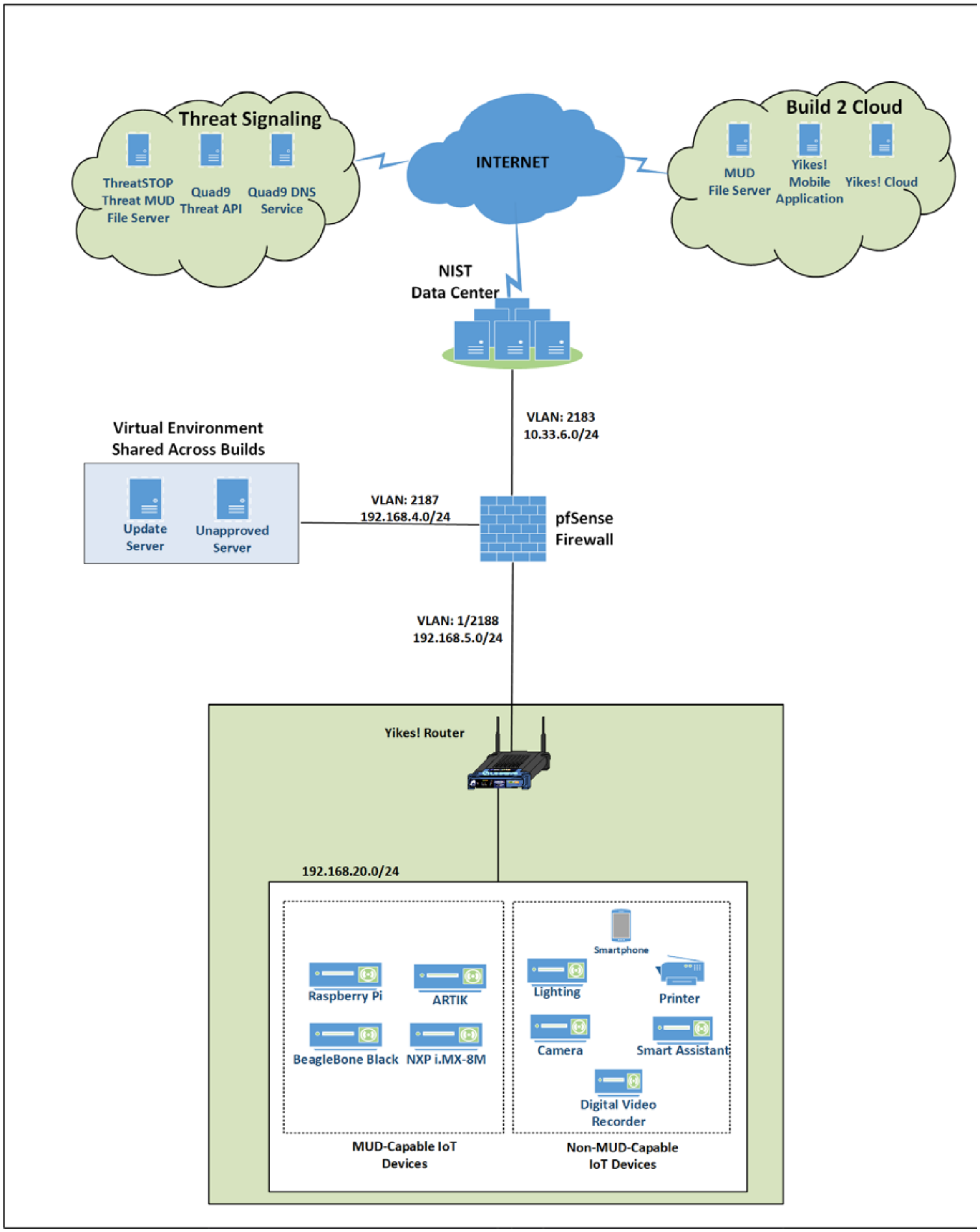
7.3.2 Physical Architecture

Figure 7-3 depicts the physical architecture of Build 2. A single DHCP server instance is configured for the local network to dynamically assign IPv4 addresses to each IoT device that connects to the Yikes! router. This single subnet hosts both MUD-capable and non-MUD-capable IoT devices. The network infrastructure as configured utilizes the IPv4 protocol for communication both internally and to the internet.

In addition, this build uses a portion of the virtual environment that is shared across builds. Services hosted in this environment include an update server and an unapproved server.

Internet-accessible cloud services are also supported in Build 2. This includes a MUD file server and Yikes! cloud services. To support threat-signaling functionality, a ThreatSTOP threat MUD file server, Quad9 threat API, and Quad9 DNS service were utilized.

1894 **Figure 7-3 Physical Architecture—Build 2**



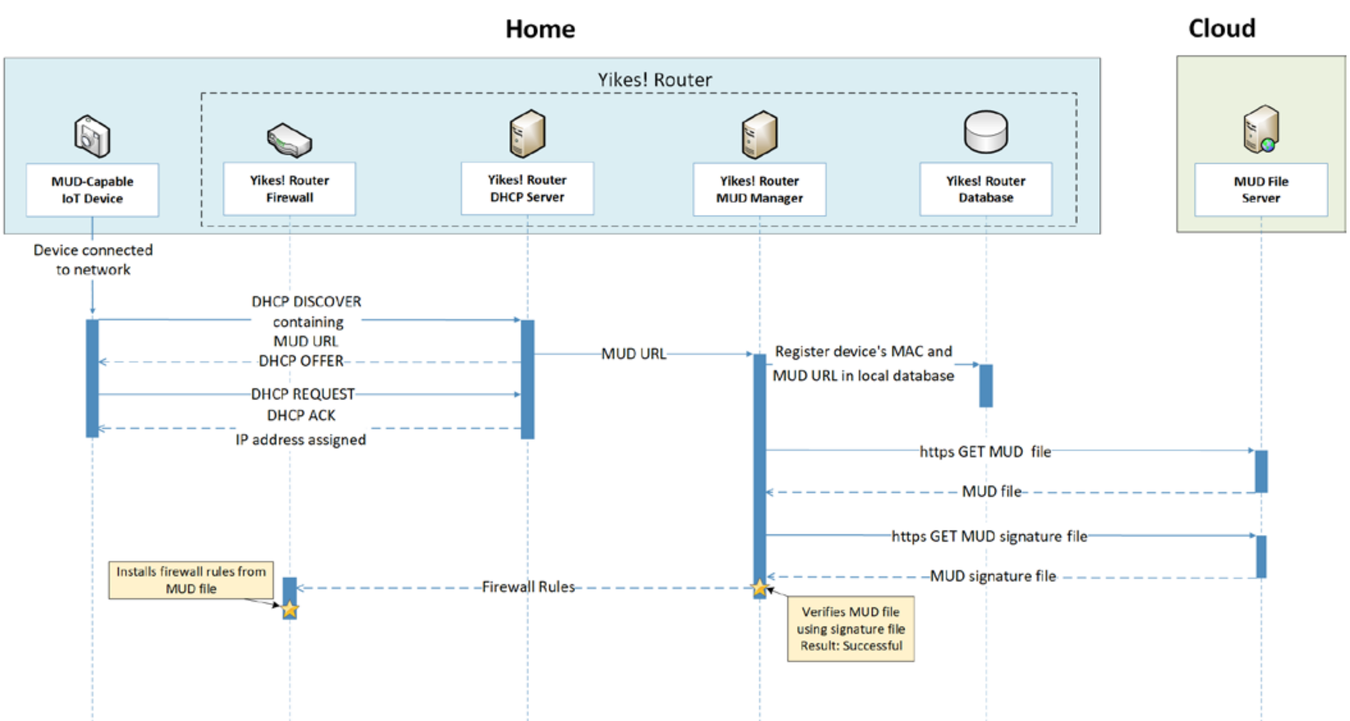
7.3.3 Message Flow

This section presents the message flows used in Build 2 during several different processes of note.

7.3.3.1 Installation of MUD-Based Access Control Rules for MUD-Capable Devices

Figure 7-4 depicts the message flows involved in the process of installing MUD-based access control rules for a MUD-capable IoT device in Build 2.

Figure 7-4 MUD-Capable IoT Device MUD-Based ACL Installation Message Flow—Build 2



The components used to support Build 2 are deployed across the home/small-business network (shown in blue) and the cloud (shown in green). A single device called the Yikes! router on the home/small-business network hosts five logical components: the Yikes! router firewall, the Yikes! router DHCP server, the Yikes! router MUD manager, the Yikes! router database, and the Yikes! router agent. (The Yikes! agent is not depicted in Figure 7-4 because it is not involved in installing MUD-based access control rules for the MUD-capable device.) The MUD file server is in the cloud, as are the device's update server and the Yikes! cloud service. (Again, only the MUD file server is depicted in Figure 7-4 because it is the only cloud component that is involved in installing MUD-based access control rules for the MUD-capable device.)

As shown in Figure 7-4, the message flow is as follows:

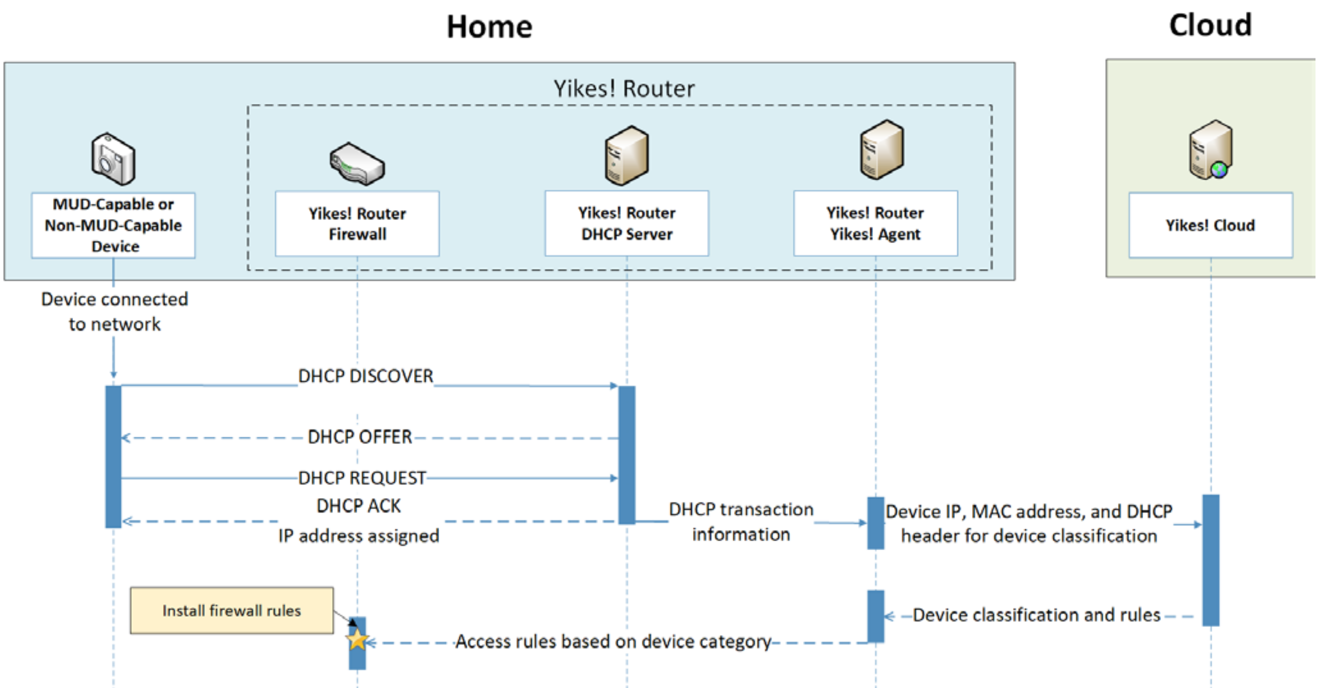
- When a MUD-capable IoT device is connected to the home/small-business network in Build 2, it exchanges DHCP protocol messages with the DHCP server on the router to obtain an IP address. The IoT device provides its MUD file URL within the DHCP DISCOVER message, as specified in the MUD RFC.
- The DHCP server forwards the MUD file URL and the MAC address of the connecting device to the MUD manager.
- The MUD manager registers the MAC address and MUD file URL of the device in the database that is located on the router.
- The MUD manager fetches the MUD file and the MUD file signature file from the MUD file server.
- After verifying that the MUD file is valid, the MUD manager installs the access control rules that correspond to the MUD file rules onto the router's firewall.

7.3.3.2 Installation of Category-Based Access Control Rules for All Devices

Figure 7-5 depicts the message flows involved in the process of installing category-based access control rules for all devices in Build 2 (both MUD-capable and non-MUD-capable devices), which are as follows:

- When a device is connected to the home/small-business network in Build 2, it exchanges DHCP protocol messages with the DHCP server to obtain an IP address. If it is a MUD-capable device, it also includes a MUD URL in this DHCP protocol exchange, and the message flow depicted in Figure 7-4 occurs in addition to the following message flow that is depicted in Figure 7-5. If it is a non-MUD-capable device, it does not include a MUD URL in this DHCP protocol exchange, and only the following message flow occurs.
- The DHCP server forwards information relevant to the connecting device such as IP address, MAC address, and DHCP header to the Yikes! router agent.
- The Yikes! router agent, in turn, forwards this information to the Yikes! cloud so the cloud can try to identify and classify the device.
- The Yikes! cloud sends the Yikes! router agent its determination of the device's category and associated traffic rules.
- The Yikes! router agent then configures the router with firewall rules for the device based on the device's category. Note that for this process to work, it is assumed that the Yikes! cloud has been preconfigured with various categories and traffic profile rules pertaining to each category. These rules can be configured by a user at any time by using the Yikes! mobile application.
- Note that if a device is MUD-capable and its MUD file rules conflict with its Yikes! category rules, both the device MUD rules and Yikes! category rules are installed, but the MUD rules take precedence and are enforced first.

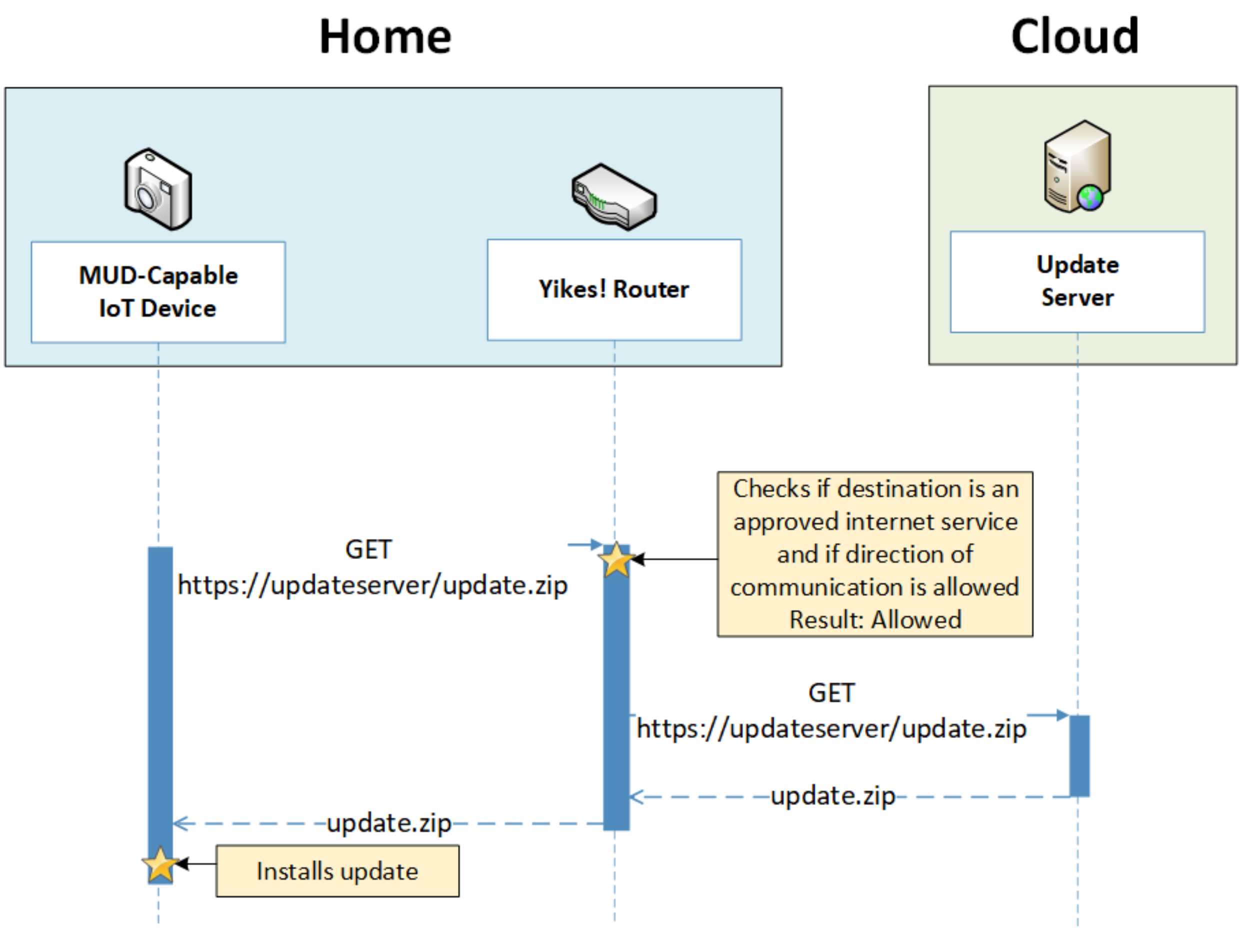
Figure 7-5 All Device Category-Based ACL Installation Message Flow—Build 2



1945 ### *7.3.3.3 Updates*

1946 After a device has been permitted to connect to the home/small-business network, it should
1947 periodically check for updates. The message flow for updating the IoT device is shown in Figure 7-6
1948 Update Process Message Flow—Build 2.

Figure 7-6 Update Process Message Flow—Build 2



As shown in Figure 7-6 Update Process Message Flow—Build 2, the message flow is as follows:

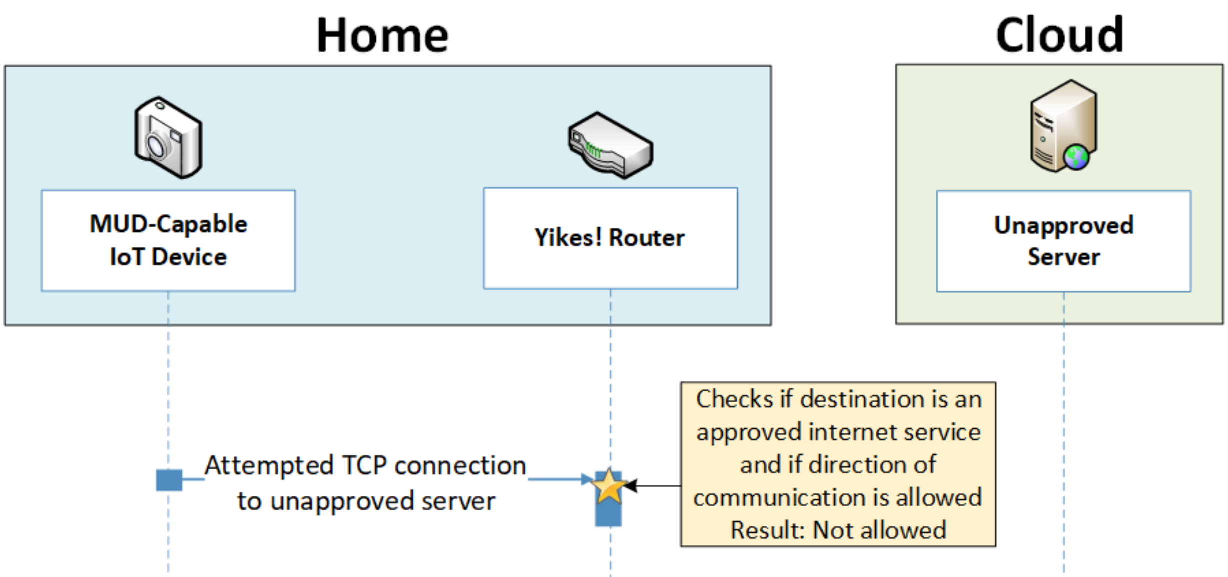
- The device generates an https GET request to its update server.
- The Yikes! router will consult the firewall rules for this device to verify that it is permitted to send traffic to the update server. Assuming there were explicit rules in the device's MUD file enabling it to send messages to this update server, the Yikes! router will forward the request to the update server.
- The update server will respond with a zip file containing the updates.
- The Yikes! router will forward this zip file to the device for installation.

7.3.3.4 Prohibited Traffic

Figure 7-7 shows an attempt to send traffic that is prohibited by the MUD file and so is blocked by the Yikes! router.

- A connection attempt is made from a local IoT device to an unapproved server. (The unapproved server is located at a domain to which the MUD file does not explicitly permit the IoT device to send traffic.)
- This connection attempt is blocked because there is no firewall rule in the Yikes! router that permits traffic from the IoT device to the unapproved server.

Figure 7-7 Unapproved Communications Message Flow—Build 2



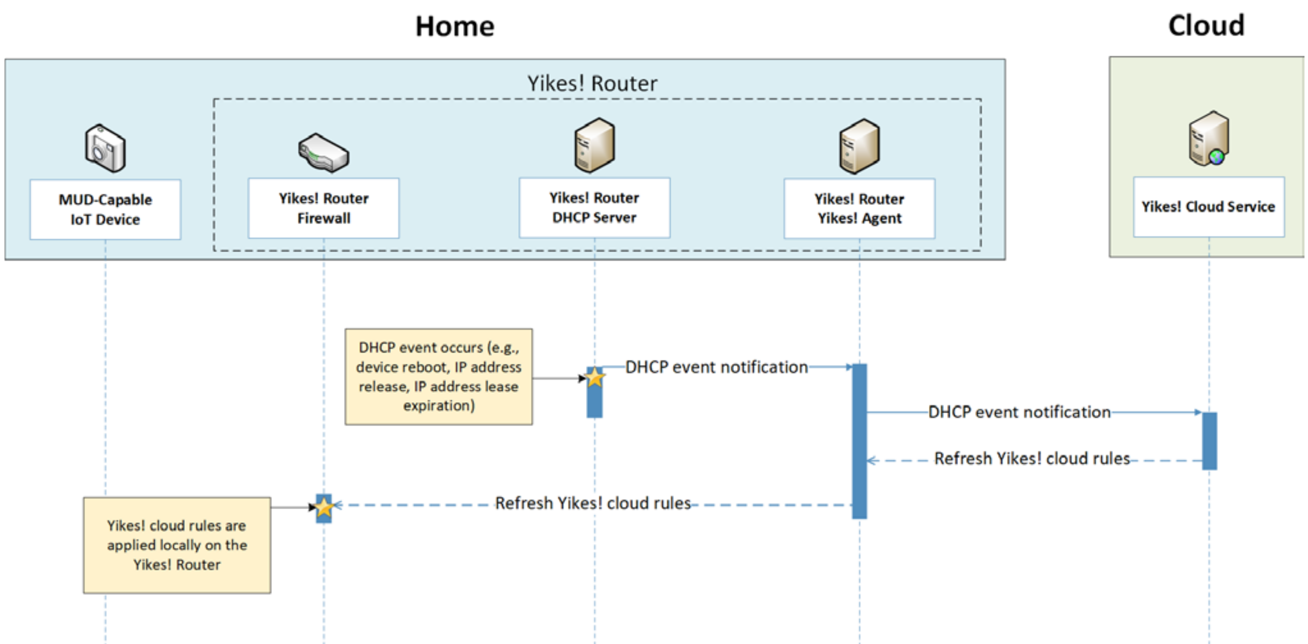
7.3.3.5 DHCP Events

Figure 7-8 shows the message flow when a change of DHCP state occurs, for example, when a device's IP address is assigned to a newly connected device, a lease expires, or a lease is explicitly released by the device. The Yikes! agent is triggered to send a notification to the Yikes! cloud to update or refresh the Yikes! cloud rules on the router when a DHCP event occurs. This update refreshes the firewall rules defined at the device category level that have been configured through the Yikes! cloud to be applied onto the Yikes! router. Figure 7-8 shows the following message flow:

- The DHCP event triggers a notification that is sent to the Yikes! router Yikes! agent.
- The Yikes! router Yikes! agent forwards the notification to the Yikes! cloud service.

- The Yikes! cloud service responds by sending a refresh of all Yikes! cloud rules to the Yikes! router agent.
- The Yikes! router Yikes! agent installs these refreshed rules onto the Yikes! router firewall.

Figure 7-8 DHCP Event Message Flow—Build 2



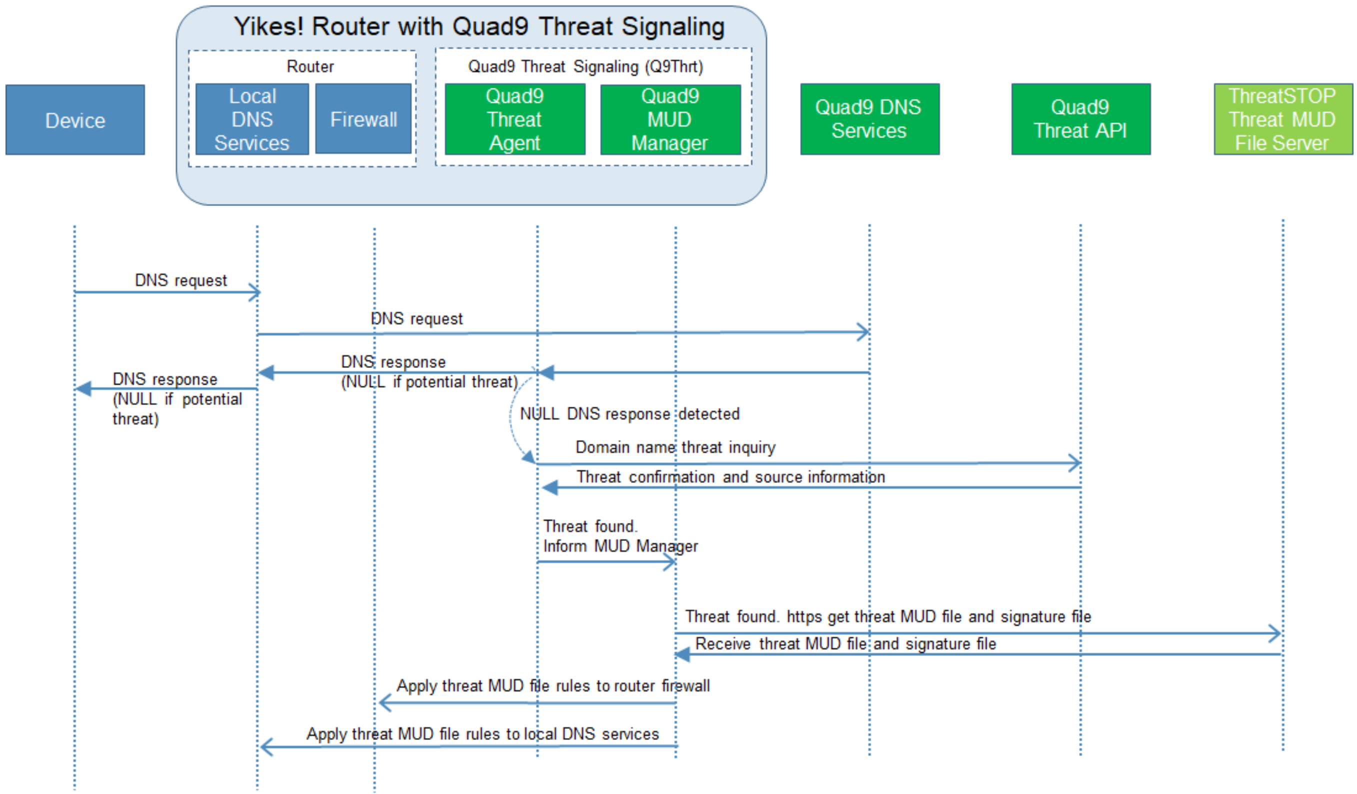
7.3.3.6 Threat Signaling

Figure 7-9 shows the message flow required to support threat signaling in Build 2.

- A local device (which may or may not be an IoT device and may or may not be MUD-capable) sends a DNS resolution request to its local DNS service, which is hosted on the Yikes! router.
- If the local DNS service cannot resolve the request itself, it will forward the request to the Quad9 DNS service.
- The Quad9 DNS service receives input from several threat intelligence providers (not depicted in the diagram) so the providers are aware of whether the domain in question has been identified to be unsafe. If the domain has not been identified as unsafe, the Quad9 DNS service will respond with the IP address(es) corresponding to the domain (as would any normal DNS service). If the domain has been flagged as unsafe, however, the Quad9 DNS service will not resolve the domain. Instead, it will return an empty (null) DNS response message to the local DNS service.

- The local DNS service will forward the DNS response to the device that originally made the DNS resolution request.
- Meanwhile, the Quad9 threat agent that is running on the Yikes! router monitors all DNS requests and responses. When it sees a domain that does not get resolved, it sends a query to the Quad9 threat API asking whether the domain is dangerous and, if so, which threat intelligence provider had flagged it as such and with what threat it is associated (this query is labeled “Domain name threat inquiry” in Figure 7-9).
- The Quad9 threat API responds with this information, which, in this case, informs the threat agent that if it wants more information about the blocked domain, it should contact ThreatSTOP (a threat intelligence provider) and request a threat MUD file. This threat MUD file will list domains and IP addresses that should be blocked because they are all associated with the same threat campaign as this threat.
- Next, the Quad9 threat agent provides this information to the Quad9 MUD manager.
- The Quad9 MUD manager requests and receives this threat MUD file and the threat MUD file signature file from the ThreatSTOP threat MUD file server.
- After ensuring that the threat MUD file is valid, the Quad9 MUD manager uses the threat MUD file to configure the router’s firewall to block all domains and IP addresses listed in this threat MUD file.
- The Quad9 MUD manager also configures the router’s local DNS services to provide empty responses for DNS requests that are made for all domains that are listed in the threat MUD file.

Figure 7-9 Message Flow for Protecting Local Devices Based on Threat Intelligence—Build 2



7.4 Functional Demonstration

A functional evaluation and a demonstration of Build 2 were conducted that involved two types of activities:

- Evaluation of conformance to the MUD RFC—Build 2 was tested to determine the extent to which it correctly implements basic functionality defined within the MUD RFC.
- Demonstration of additional (non-MUD-related) capabilities—It did not verify the example implementation's behavior for conformance to a standard or specification; rather, it demonstrated advertised capabilities of the example implementation related to its ability to increase device and network security in ways that are independent of the MUD RFC. These capabilities may provide security for both non-MUD-capable and MUD-capable devices. Examples of this type of activity include device discovery, identification and classification, and support for threat signaling.

Table 7-2 summarizes the tests used to evaluate Build 2’s MUD-related capabilities, and Table 7-3 summarizes the exercises used to demonstrate Build 2’s non-MUD-related capabilities. Both tables list each test or exercise identifier, a summary of the test or exercise, the test or exercise’s expected and observed outcomes, and the applicable Cybersecurity Framework Subcategories and NIST SP 800-53 controls for which each test or exercise verifies support. The tests and exercises listed in the table are detailed in a separate supplement for functional demonstration results. Boldface text is used to highlight the gist of the information that is being conveyed.

Table 7-2 Summary of Build 2 MUD-Related Functional Tests

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
IoT-1	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties.	A **MUD-capable IoT device is configured to emit a MUD URL within a DHCP message.** The DHCP server assigns its IP address and extracts the MUD URL, which is sent to the MUD manager. The MUD manager requests the MUD file and signature from the MUD file server, and the MUD file server serves the MUD file to the MUD manager. The MUD file explicitly permits traffic to/from some internet services and hosts and implicitly denies traffic to/from all other internet services. **The MUD manager translates the MUD file information into local network configurations that it installs on the router or switch that is serving**	Upon connection to the network, the MUD-capable IoT device has its MUD **PEP router/switch automatically configured according to the MUD file’s route-filtering policies.**	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	NIST SP 800-53 Rev. 4 AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.	**as the MUD PEP for the IoT device.**		
IoT-2	**PR.AC-7:** Users, devices, and other assets are authenticated (e.g., single-factor, multifactor) commensurate with the risk of the transaction (e.g., individuals' security and privacy risks and other organizational risks). **NIST SP 800-53 Rev. 4** AC-7, AC-8, AC-9, AC-11, AC-12, AC-14, IA-1, IA-	A MUD-capable IoT device is configured to emit a URL for a MUD file, but the **MUD file server that is hosting that file does not have a valid TLS certificate. Local policy has been configured to ensure that if the MUD file for**	When the MUD-capable IoT device is connected to the network, the MUD manager sends locally defined policy to the router/switch that handles whether to allow or block traffic to the	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	2, IA-3, IA-4, IA-5, IA-8, IA-9, IA-10, IA-11	**an IoT device is located on a server with an invalid certificate, the router/switch will be configured by local policy to allow all communication to/from the device.**	MUD-capable IoT device. Therefore, the **MUD PEP router/switch will be configured to allow all traffic to and from the IoT device.**	
IoT-3	**PR.DS-6:** Integrity-checking mechanisms are used to verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-capable IoT device is configured to emit a URL for a MUD file, but the **certificate that was used to sign the MUD file had already expired at signing. Local policy has been configured to ensure that if the MUD file for a device has a signature that was signed by a certificate that had already expired at the time of signature, the device's MUD PEP router/switch will be configured by local policy to either allow or deny all communication to/from the device.**	When the MUD-capable IoT device is connected to the network and the MUD file and signature are fetched, the MUD manager will detect that the MUD file's signature was created by using a certificate that had already expired at signing. According to local policy, the **MUD PEP will be configured to either allow or block all traffic to/from the device.**	Pass
IoT-4	**PR.DS-6:** Integrity-checking mechanisms are used to verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-capable IoT device is configured to emit a URL for a MUD file, but the **signature of the MUD file is invalid. Local policy has**	When the MUD-capable IoT device is connected to the network, the MUD manager sends locally defined policy to the router/switch	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		been configured to ensure that if the MUD file for a device is invalid, the router/switch will allow all communication to/from the IoT device.	that handles whether to allow or block traffic to the MUD-capable IoT device. Therefore, the **MUD PEP router/switch will be configured to allow all traffic to and from the IoT device.**	
IoT-5	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7	Test IoT-1 has run successfully, meaning that the MUD PEP router/switch has been configured based on a **MUD file that permits traffic to/from some internet locations and implicitly denies traffic to/from all other internet locations.**	When the MUD-capable IoT device is connected to the network, its MUD PEP **router/switch will be configured to enforce the route filtering that is described in the device's MUD file** with respect to traffic being permitted to/from some internet locations, and traffic being implicitly blocked to/from all remaining internet locations.	Pass (for testable procedure, ingress cannot be tested due to Network Address Translation [NAT])
IoT-6	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8	Test IoT-1 has run successfully, meaning that the MUD PEP router/switch has been	When the MUD-capable IoT device is connected to the network, its MUD	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.DS-3: Assets are formally managed throughout removal, transfers, and disposition. **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7	configured based on a **MUD file that permits traffic to/from some lateral hosts and implicitly denies traffic to/from all other lateral hosts.** (The MUD file does not explicitly identify the hosts as lateral hosts; it identifies classes of hosts to/from which traffic should be denied, where one or more hosts of this class happen to be lateral hosts.)	PEP **router/switch will be configured to enforce the access control information that is described in the device's MUD file** with respect to traffic being permitted to/from some lateral hosts, and traffic being implicitly blocked to/from all remaining lateral hosts.	
IoT-7	**PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.DS-3:** Assets are formally managed throughout removal, transfers, and disposition.	Test IoT-1 has run successfully, meaning that the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question.	When the MUD-capable **IoT device explicitly releases its IP address lease,** the MUD-related configuration for that IoT device will be removed from its	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		Next, have **the IoT device change DHCP state by explicitly releasing its IP address lease, causing the device's policy configuration to be removed from the MUD PEP router/switch.**	MUD PEP router/switch.	
IoT-8	**PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.DS-3:** Assets are formally managed throughout removal, transfers, and disposition.	Test IoT-1 has run successfully, meaning that the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question. Next, have **the IoT device change DHCP state by waiting until the IoT device's address lease expires, causing the device's policy configuration to be removed from the MUD PEP router/switch.**	When the MUD-capable **IoT device's IP address lease expires,** the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch.	Pass
IoT-9	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped.	Test IoT-1 has run successfully, meaning the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question. The MUD file contains domains that resolve to multiple IP addresses.	A domain in the MUD file resolves to two different IP addresses. The MUD manager will create firewall rules that permit the MUD-capable device to send traffic to both IP addresses. The MUD-	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	NIST SP 800-53 Rev. 4 AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CM-2, SI-4 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-2, CM-3, SA-10	The MUD PEP router/switch should be configured to permit communication to or from all IP addresses for the domain.	capable device attempts to send traffic to each of the IP addresses, and the MUD PEP router/switch permits the traffic to be sent in both cases.	

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.DS-2: Data in transit is protected. **NIST SP 800-53 Rev. 4** SC-8, SC-11, SC-12			
IoT-10	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7	A MUD-capable IoT device is configured to emit a MUD URL. Upon being connected to the network, its MUD file is retrieved, and the PEP is configured to enforce the policies specified in that MUD URL for that device. **Within 24 hours (i.e., within the cache-validity period for that MUD file), the IoT device is reconnected to the network.** After 24 hours have elapsed, the same device is reconnected to the network.	Upon reconnection of the IoT device to the network, **the MUD manager does not contact the MUD file server. Instead, it uses the cached MUD file.** It translates this MUD file's contents into appropriate route-filtering rules and installs these rules onto the PEP for the IoT device. Upon reconnection of the IoT device to the network, after 24 hours have elapsed, the MUD manager does fetch a new MUD file.	Not testable in preproduction implementation

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.IP-1: A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.			
IoT-11	**ID.AM-1:** Physical devices and systems within the organization are inventoried.	A **MUD-enabled IoT device can emit a MUD URL.** The device should leverage one of the specified manners for emitting a MUD URL.	Upon initialization, the MUD-enabled IoT device broadcasts a DHCP message on the network, including at most one **MUD URL, in https scheme, within the DHCP transaction.**	Pass

In addition to supporting MUD, Build 2 can identify a device's make (i.e., manufacturer) and model, categorize devices based on their make and model, and associate device categories with traffic policies that affect both internal and external traffic transmissions, as shown in Table 7-3.

Table 7-3 Non-MUD-Related Functional Capabilities Demonstrated

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
YnMUD-1	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CM-2, SI-4 **DE.CM-1:** The network is monitored to detect potential cybersecurity events. **NIST SP 800-53 Rev. 4** AC-2, AU-12, CA-7, CM-3, SC-5, SC-7, SI-4	A device identification and a categorization capability are supported by the router and cloud services. The **router is designed to detect all devices connected to the network and leverage cloud services to identify the devices using attributes associated with them, as well as categorize the devices by type when possible. If unable to identify and categorize them, devices are designated as uncategorized.**	Upon being connected to the network, the **router detects all connected devices and leverages a cloud service, which identifies each device's make and model using attributes** (e.g., type, IP address, OS), and **categorizes them** (e.g., cell phone, printer, smart appliance).	As expected
YnMUD-2	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped.	After executing YnMUD-1 successfully, the **UI is used to modify make, model, and/or category of connected devices.**	Connected devices have been identified and categorized automatically upon being connected to the network. Using the UI, show that the make and model of	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
			a device can be modified, and that the category of the device can be assigned manually.	
YnMUD-3	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **ID.AM-4:** External information systems are catalogued. **NIST SP 800-53 Rev. 4** AC-20, SA-9 **PR.AC-1:** Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users and processes. **NIST SP 800-53 Rev. 4** AC-1, AC-2, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11 **NIST SP 800-53 Rev. 4** PE-2, PE-3, PE-4, PE-5, PE-6, PE-8 **PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected (e.g., network segregation, network segmentation).	**The router can apply traffic policies to categories of devices that restrict initiation of (south-to-north) communications to internet sites** by all devices in the specified category. Communication **can be configured to (a) allow all internet communication, (b) deny all internet communication to devices of a specific make and model, or (c) permit communication only to/from specified internet domains and devices of a specific make and model.**	Through the UI, device category rules can be defined to permit connectivity to every internet location by selecting “Allow All Internet Traffic” or to device-specific sites by selecting “IoT specific sites.” Set rules for the **computer category to permit all internet traffic,** and attempt to initiate communication from laptop to any internet host. **All internet communication from laptop will be approved.** Next, set rules for **Smart Appliance category to permit IoT-specific site,** and attempt to initiate communication to specific sites permitted for the make and model of the device being tested. **All specified sites for device make and model should be**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	NIST SP 800-53 Rev. 4 AC-4, AC-10, SC-7		**permitted, and any other communication outside these specified hosts should be blocked.** Last, set rules for **a third type of device category (cell phone) to permit IoT-specific sites, but do not specify any sites as permissible. The device should not be permitted to initiate communication with any internet sites.**	
YnMUD-4	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **ID.AM-4:** External information systems are catalogued. **NIST SP 800-53 Rev. 4** AC-20, SA-9 **PR.AC-1:** Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users, and processes. **NIST SP 800-53 Rev. 4** AC-1, AC-2, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11 **PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15	The router can apply **policies to categories of devices** (as defined by a user through the UI) to **specify rules regarding initiation of lateral (east/west) communications to other categories of devices on the local network. All traffic is enforced according to** rules associated with **the device's category.**	**Through the UI,** device category rules can be defined to **permit connectivity between categories of devices.** Set rules for **category x to permit communication with category y but not to category z.** After rules have been set, **attempt to communicate from a device in category x to a device in category y;** the router will **permit this communication** to occur. Next, **attempt to communicate from**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	PR.AC-4: Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected (e.g., network segregation, network segmentation). **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7		**a device in category x to a device in category z;** the router **will not permit this communication** to occur.	
YnMUD-5	**ID.RA-2:** Cyber threat intelligence is received from information-sharing forums and sources. **NIST SP 800-53 Rev. 4** SI-5, PM-15, PM-16 **ID.RA-3:** Threats, both internal and external, are identified and documented. **NIST SP 800-53 Rev. 4** RA-3, SI-5, PM-12, PM-16 **ID.RA-5:** Threats, vulnerabilities, likelihoods, and impacts are used to determine risk. **NIST SP 800-53 Rev. 4** RA-2, RA-3, PM-16 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties.	The router can query a threat intelligence provider and receiving threat information related to domains that devices on the network are attempting to access. In **response to threat information, all devices on the local network are prohibited from visiting specific domains and IP addresses.**	A **device on the network sends a DNS request for a malicious domain** to which it is attempting to navigate. The **router receives a response indicating that the domain is potentially malicious. The router queries** threat **services** regarding the domain and receives back the URL for the threat MUD file that is associated with the domain. The router retrieves the threat MUD file and installs its rules as global firewall rules. As a result, the **device that attempted to communicate with the dangerous**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
			domain is blocked from communicating with that domain as well as all other domains associated with that same threat.	
YnMUD-6	**PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected (e.g., network segregation, network segmentation). **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **ID.RA-2:** Cyber threat intelligence is received from information-sharing forums and sources. **NIST SP 800-53 Rev. 4** SI-5, PM-15, PM-16 **ID.RA-3:** Threats, both internal and external, are identified and documented. **NIST SP 800-53 Rev. 4** RA-3, SI-5, PM-12, PM-16	YnMUD-5 was successfully completed, i.e., **in response to threat information received in YnMUD-5, all devices on the local network are prohibited from visiting not only the domains that are associated with the identified threat but also with all IP addresses associated with these domains.**	A **different device on the network attempts to communicate with the malicious domain identified in test YnMUD-5 via its IP address instead of its domain.** Router firewall rules prohibiting access to this IP address should already be present as a result of test **YnMUD-5. As** a result, the **device that attempted to communicate to the IP address is prevented from initiating communication**.	As expected
YnMUD-7	**PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15	YnMUD-5 was successfully completed, resulting in the router being configured with threat	Log in to the router and verify that the firewall rules that	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	PR.AC-4: Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected (e.g., network segregation, network segmentation). **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **ID.RA-2:** Cyber threat intelligence is received from information-sharing forums and sources. **NIST SP 800-53 Rev. 4** SI-5, PM-15, PM-16 **ID.RA-3:** Threats, both internal and external, are identified and documented. **NIST SP 800-53 Rev. 4** RA-3, SI-5, PM-12, PM-16	intelligence rules**. The threat intelligence was received more than 24 hours earlier.** It indicated domains and IP addresses that should not be trusted, and those domains and IP addresses were blocked by firewall rules installed on the router. **After 24 hours, these firewall rules have been removed from the router.**	prohibited communication to malicious domains (and that were verified as present in the previous two tests) are no longer present.	

7.5 Observations

Build 2 was able to successfully permit and block traffic to and from MUD-capable IoT devices as specified in the MUD files for the devices. It was also able to constrain communications to and from all devices (both MUD-capable and non-MUD-capable) based on the traffic profile associated with the device's category in the Yikes! cloud.

We observed the following limitations to Build 2 that are informing improvements to its current proof-of-concept implementation:

- MUD manager (version 1.1.3):
    - MUD file caching is not supported in this version of the MUD manager. The MUD manager fetches a new MUD file for every MUD request that occurs, regardless of the cache-validity of the current MUD file.

- Yikes! cloud:
    - Yikes! performs device identification using data available at the time a device requests an IP address during the network connection process. Future versions of the product may collect additional information about a device to improve the specificity of device identification.
- Yikes! mobile application:
    - At demonstration time, the Yikes! mobile application was under development. For this reason, Yikes! provided a web-hosted replica of the mobile application under development. This was accessible via web browsers on both mobile and computer platforms.
- Yikes! router (version 1.1.3):
    - At demonstration time, DHCP was the only MUD URL emission method supported. LLDP and X.509 MUD URL emission methods are not supported by the current version of the Yikes! router.
    - When MUD-capable devices are first connected and introduced to the network, the default policy in this version of the Yikes! router is to allow communications while the MUD file is being requested and processed. This results in a short period of time during which the device has received an IP address and is able to communicate unconstrained on the network before the MUD rules related to the device are applied.
    - In some situations, when a MUD-capable IoT device is connected to the network, the base router configurations may contend with the MUD rules. This can result in the initial instances of unapproved attempted communication from the MUD-capable device to other devices on the local network being permitted until the router reconciles the configuration. Traffic to or from locations outside the local network is not impacted and only approved traffic is ever allowed.
    - At demonstration time, the automated process to associate the Yikes! router with the Yikes! cloud service was still under development, and association had to be done manually by MasterPeace.
- threat signaling (version 0.4.0):
    - Access to threat-signaling information is triggered when a device on the local network makes a DNS resolution request for a domain that has been flagged as dangerous because it is associated with some known threat. If a device attempts to connect to a dangerous site using that site's IP address rather than its domain name without first attempting to resolve a domain name that is associated with the same threat that is associated with the dangerous site, the threat-signaling mechanism provided in Build 2 will not block access to that IP address. Therefore, users are cautioned to use domain names rather than IP addresses when attempting outbound communication to ensure that they can take full advantage of the threat-signaling protections offered by Build 2.

8 Build 3

The Build 3 implementation uses equipment supplied by CableLabs to onboard devices and to support MUD. Build 3 leverages the Wi-Fi Alliance's Wi-Fi Easy Connect specification to securely onboard devices to the network (i.e., to provision each device with the unique network credentials that it needs to connect to the network). It also uses SDN to create separate trust zones (e.g., network segments) to which devices are assigned according to their intended network function. The Build 3 network platform is called Micronets, and there is an open-source reference implementation of the Micronets platform available on the Micronets project site as well as on GitHub. The Micronets platform is continually evolving with new features and functionality being added to its open-source reference implementation.

Micronets consists of:

- an on-premises Micronets-capable gateway that resides on the home/small-business network. A micronet is a trust zone that is implemented as a network segment and is used to group devices together into trust domains that isolate devices based on their function and access policy. The Micronets Gateway manages and enforces service-specific micronets and customer-defined micronets. Cloud-based microservices layer that hosts various Micronets services (e.g., SDN controller, Micronets Manager, MUD Manager, Configuration microservice, identity server (optional), DHCP/DNS configuration services) that interact with the on-premises Micronets Gateway to manage local devices and network connectivity. The most important of these is the Micronets Manager, which interacts with all of the other microservices to coordinate the state of the Micronets-enabled on-premises network.
- Cloud-based Intelligent Services and Business Logic layer (e.g., machine-learning-based services) that is operated by the service provider.
- Micronets APIs, which allow partners and service providers to interface with a customer's micro-networks environment to provision and deliver specific customer-requested services.
- Micronets Mobile App that supports device onboarding using the Wi-Fi Easy Connect protocol

These various components may be used in combination to onboard devices and leverage MUD, if supported by the device. The on-premises Micronets Gateway supports the Wi-Fi Easy Connect protocol for IoT device onboarding and leverages it to provision the device in the right trust domain. This Micronets Gateway can enforce policy-based flows where the policies can be derived from MUD-based traffic constraints or other policy sources. It also supports dynamic micro-segmentation.
CableLabs provided prototype Micronets platform components in the NCCoE lab based on the open-source reference implementation available on GitHub. A more detailed description of Micronets can be found in CableLabs' Micronets white paper, and the various Micronets components listed above are each described more fully in Section 8.3.1.

8.1 Collaborators

Collaborators that participated in this build are described briefly in the subsections below.

8.1.1 CableLabs

CableLabs is an innovation and R&D laboratory for the cable industry. CableLabs is a not-for-profit global network technologies organization with member companies around the world. CableLabs offers state-of-the-art research and innovation facilities with collaborative ecosystem made up of thousands of vendors. In November 2018, CableLabs publicly announced Micronets, a next-generation on-premise network platform. Micronets provides adaptive security for all devices connecting to residential or small-business networks through dynamic micro-segmentation and management of connectivity to those devices. Micronets is designed to provide seamless and transparent security to users without burdening them with the technical aspects of configuring the network. Micronets incorporates and leverages MUD as one technology component to help identify and manage the connectivity of devices, in support of the broader Micronets platform. It also leverages the Wi-Fi Easy Connect protocol to enable IoT devices to be onboarded easily and securely, and to provide each IoT device with unique network credentials. In addition, Micronets can provide enhanced security for high-value or sensitive devices, further reducing the risk of compromise for these devices and their applications. Learn more about CableLabs at https://www.cablelabs.com.

8.1.2 DigiCert

See Section 6.1.2 for a description of DigiCert.

8.2 Technologies

Table 8-1 lists all the products and technologies used in Build 3 and provides a mapping among the generic component term, the specific product used to implement that component, and the function subcategories that the product provides. When applicable, both the function subcategories that a component provides directly and those that it supports, but does not provide directly, are listed and labeled as such. For rows in which the provides/supports distinction is not noted, all listed categories are directly provided by the component. Refer to Table 5-1 for an explanation of the Cybersecurity Framework's Subcategory codes.

Table 8-1 Products and Technologies

Component	Product	Function	Cybersecurity Framework Subcategories
MUD manager	Service provider’s cloud infrastructure MUD Manager component	Fetches, verifies, caches, and processes MUD files from the MUD file server; provides parsed MUD rules as ACLs to the Micronets Manager that is on the service provider cloud, which will send these ACLs to the home/small-business network Micronets Gateway, which will convert them into traffic flow rules.	Provides: PR.PT-3 Supports: ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 DE.AE-1
MUD file server	A web server that hosts the device’s MUD file	Hosts MUD files; serves MUD files to the MUD manager over https.	ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
MUD file maker	MUD file maker (https://www.mud-maker.org/)	YANG script GUI used to create MUD files	ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
MUD file	A YANG model instance that has been serialized in JSON (RFC 7951). The manufacturer of a MUD-capable device creates that device's MUD file. MUD file maker (see previous row) can create MUD files. Each MUD file is also associated with a separate MUD signature file.	Specifies the communications that are permitted to and from a given device	Provides: PR.PT-3 Supports: ID.AM-1 ID.AM-2 ID.AM-3 PR.DS-5
Router/Switch	Micronets Gateway and access point	An integrated SDN-capable switch/router and Wi-Fi access point that routes traffic on the home/small-business network. During onboarding, receives ACLs that enforce the IoT device's MUD file rules from the Micronets Manager; creates flow rules to enforce these ACLs. Creates micronets (subnetworks) to separate devices into trust zones.	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1

Component	Product	Function	Cybersecurity Framework Sub-categories
Certificates	DigiCert certificates (TLS and Premium)	Authenticate the MUD file server and secure the TLS connection between the MUD manager and the MUD file server and other components of the Micronets platform; sign MUD files and generate a corresponding signature file	PR.AC-1 PR.AC-3 PR.AC-5 PR.AC-7
MUD-capable IoT device	Raspberry Pi Model 3 B+ (devkit)	When put into onboarding mode, it displays a QR code that contains its Wi-Fi Easy Connect bootstrapping information (including elements that identify its MUD file) and begins listening for Wi-Fi Easy Connect protocol messages. After being authenticated by the Easy Connect-capable Micronets Gateway, the gateway provides it with its unique network credentials. Also requests and applies software updates	ID.AM-1

Component	Product	Function	Cybersecurity Framework Sub-categories
Update server	NCCoE-hosted Apache server	Acts as a device manufacturer's update server that would communicate with IoT devices to provide patches and other software updates	PR.IP-1 PR.IP-3
Unapproved server	NCCoE-hosted Apache server	Acts as an internet host that has not been explicitly approved in a MUD file	DE.DP-3 DE.AM-1
MUD registry	Micronets MUD Registry	Provides a service that looks up each MUD-capable device's MUD file URL based on the contents of the information element field and the public key field in the device's Wi-Fi Easy Connect bootstrapping information	Provides: ID.AM-1 Supports: ID.AM-3 PR.DS-5 PR.IP-1

Component	Product	Function	Cybersecurity Framework Sub-categories
SDN controller	Micronets Manager	Although the Micronets Manager does not use the OpenFlow protocol, it functions as an SDN controller by conveying to the Micronets Gateway the ACLs and micronet topology that it wants the gateway to create and enforce. During the onboarding process, it provides the gateway with device-specific configuration, including ACLs to enforce the communications profile specified in the device's MUD file; it also indicates the micronet (trust zone) to which each IoT device should be assigned (as directed by user input to the Micronets mobile application).	Supports: PR.AC-3 PR.AC-5 PR.AC-4 PR.DS-1 PR.DS-2 PR.DS-5 PR-PT-3

Component	Product	Function	Cybersecurity Framework Subcategories
onboarding manager	Micronets Configuration Microservice and Micronets Manager	Resides in the service provider cloud and manages the onboarding process. Receives the onboarding request and device bootstrapping information from the Micronets mobile phone application (via the multiple-system operator (MSO) portal) and provides it to the Micronets Gateway. Looks up the device's MUD file URL in the MUD registry, sends the MUD file URL to the MUD manager, receives back ACLs corresponding to the parsed MUD rules from the MUD manager, and provides these to the Micronets Gateway for enforcement	Supports: PR.AC-3 PR.AC-5 PR.AC-4 PR.DS-1 PR.DS-2 PR.DS-5 PR-PT-3

Component	Product	Function	Cybersecurity Framework Subcategories
User and device interface to the onboarding manager	Micronets mobile Phone application	Acts as both the Micronets Configuration Microservice's user interface and its device bootstrapping information reader. Collects device bootstrapping information from both the QR code and user input and sends this information with the onboarding request to the Micronets Manager's Configuration Microservice via the service provider's MSO portal	Supports: ID.AM-1 ID.AM-3 PR.AC-3 PR.AC-4 PR.AC-5 PR.DS-1 PR.DS-2 PR.IP-1 PR.PT-3
Bootstrapping interface to the onboarding manager	MSO portal	Receives the onboarding request from the Micronets mobile application and forwards it to the Micronets Configuration Microservice that is associated with the specific user/owner of the network and the device	Supports: ID.AM-1 ID.AM-3 PR.AC-3 PR.AC-4 PR.AC-5 PR.DS-1 PR.DS-2 PR.IP-1 PR.PT-3

Component	Product	Function	Cybersecurity Framework Subcategories
Network onboarding component	Wi-Fi Easy Connect-Capable Micronets Gateway	Based on bootstrapping and other information it receives from the onboarding manager (i.e., the Micronets Manager), interacts directly with each IoT device via the Wi-Fi Easy Connect protocol to authenticate the device, establish a secure channel with it, and provide it with its unique network credentials	Provides: PR.AC-3 PR.AC-4 PR.AC-7 Supports: PR.AC-5 PR.DS-1 PR.DS-2 PR.DS-5 PR.DS-6 PR.PT-3

Each of these components is described more fully in the following sections.

8.2.1 MUD Manager

The Micronets MUD manager is a component within the service provider cloud. During the onboarding process, the MUD manager receives the device's MUD URL from the Micronets Manager and checks its cache for the MUD file corresponding to the MUD URL. If the file is not cached or if it is cached but has been there too long, the MUD manager fetches the MUD file that is at this URL and the MUD file's signature file, verifies the MUD file based on this signature file, parses the MUD file, and generates ACLs for the device based on the MUD file. The MUD manager sends these ACLs to the Micronets Manager, which forwards them to the Micronets Gateway so it can create and install traffic flow rules to enforce the MUD file rules. The MUD manager generates ACLs for src-dnsname, dst-dnsname, my-controller, controller, same-manufacturer, manufacturer, and local-networks constructs that are specified in the MUD file. It supports both lateral east/west protection and appropriate access to internet sites (north/south protection).

8.2.2 MUD File Server

In the absence of a commercial MUD file server for this project, the NCCoE used a MUD file server that is hosted on a Linode server that is accessible via the internet. This file server stores the MUD files along with their corresponding signature files for the IoT devices used in the project. Upon receiving a GET request for the MUD files and signatures, it serves the request to the MUD manager by using https.

8.2.3 MUD File

Using the MUD file maker component referenced above in Table 8-1, it is possible to create a MUD file with the following contents:

- Internet communication class–access to cloud services and other specific internet hosts:
  - Host: updateserver (hosted internally at the NCCoE)
    - Protocol: TCP
    - Direction-initiated: from IoT device
    - Source port: any
    - Destination port: 80
- Controller class–access to **classes** of devices that are known to be controllers (could describe well-known services such as DNS or NTP):
  - Host: nccoe-server1.micronets.net
    - Protocol: TCP
    - Direction-initiated: from IoT device
    - Source port: any
    - Destination port: 1883
- Local-networks class–access to/from **any** local host for specific services (e.g., http or https):
  - Host: any
    - Protocol: TCP
    - Direction-initiated: from IoT device
    - Source port: any
    - Destination port: 80
- My-controller class–access to controllers specific to this device:
  - Controllers: mm.micronets.in
    - Protocol: TCP
    - Direction-initiated: from IoT device
    - Source port: any

- Destination port: 80
- Same-manufacturer class–access to devices of the same manufacturer:
  - Same-manufacturer: null (to be filled in by the MUD manager]
    - Protocol: TCP
    - Direction-initiated: from IoT device
    - Source port: any
    - Destination port: 80
- Manufacturer class–access to devices of a specific manufacturer (identified by MUD URL):
  - Manufacturer: devicetype (URL decided by the device manufacturer)
    - Protocol: TCP
    - Direction-initiated: from IoT device
    - Source port: any
    - Destination port: 80

8.2.4 Signature file

According to the IETF MUD specification, "a MUD file MUST be signed using CMS as an opaque binary object." The MUD file (e.g., *nist-model-fe_northsouth.json*) was signed with the OpenSSL tool by using the command described in the specification (detailed in Volume C of this publication). A Premium certificate, requested from DigiCert, was leveraged to generate the signature file (e.g., *nist-model-fe_northsouth.p7s*). Once created, the signature file is stored on the MUD file server.

8.2.5 Router/Switch

This build uses the Micronets Gateway as the router/switch on the home/small-business network. The Micronets Gateway is an SDN-capable switch that interfaces with the Micronets Manager in the service provider cloud via a RESTful interface. The gateway receives ACLs and micronet topology information from the Micronets Manager that the gateway converts to traffic flow rules that it enforces. The gateway is also integrated with a Wi-Fi access point and supports connectivity for both wired and wireless components. In support of MUD, this gateway serves as the policy enforcement point for the access control rules that are defined in each device's MUD file. These access control rules are instantiated on the switch as traffic flow rules.

In support of MUD, the gateway implements north/south IP access control protection based on src-dnsname, dst-dnsname, my-controller, and controller constructs that are specified in the MUD file. The gateway is also responsible for creating and enforcing micronets, which segregate devices. Each micronet represents a distinct trust domain and, at a minimum, represents a distinct IP subnetwork. By definition, devices in the same micronet are permitted to communicate with one another without restrictions. However, devices in different micronets are not permitted to communicate with one another unless such communication is explicitly permitted by local communications rules in the devices' MUD files.

During the onboarding process, devices are manually assigned to micronets by user input that is provided to the Micronets mobile application after the device's QR code is scanned. If devices are assigned to micronets in a way that is consistent with the local communications rules that are in the devices' MUD files, then Micronets can serve as the mechanism to enforce those local communications restrictions for the devices. By default, devices that were onboarded in Build 3 were manually assigned to separate micronets to ensure that only local communications that were explicitly permitted in the devices' MUD files would be permitted.

Devices can talk to other devices in the same micronet without restrictions. (Cross-device communication can be enabled between micronets as needed.) Sorting devices into specific micronets for enforcing local communications restrictions defined in the MUD file cannot currently be performed automatically. However, future versions of the Micronets implementation may support automatic placement of devices into specific micronets based on the local communications rules defined in their MUD files, thereby using the communications constraints imposed by each micronet to enforce same-manufacturer, manufacturer, and local-networks constructs.

8.2.6 Certificates

DigiCert provisioned a Premium Certificate for signing the MUD files. The Premium Certificate supports the key extensions required to sign and verify CMS structures as required in the MUD specification. DigiCert certificates also authenticate the MUD file server; secure the TLS connection between the MUD manager and the MUD file server; and mutually authenticate the connection between the Micronets Manager and the micronets. All of the web services also use web certificates. Further information about DigiCert's CertCentral web-based platform, which allows provisioning and managing publicly trusted X.509 certificates, is in Section 6.2.8.

8.2.7 IoT Devices

This section describes the IoT devices used in the laboratory implementation. There are two distinct categories of devices: devices that support MUD and have a vendor code value in the information element field of their onboarding QR code to indicate the location of the device's MUD file server, i.e., MUD-capable IoT devices; and devices that do not support MUD and do not have a value in the information element field of their onboarding QR code, i.e., non-MUD-capable IoT devices. For more information regarding how the information element field value is used to locate the device's MUD file, see Section 8.3.1.1.

8.2.7.1 MUD-Capable IoT Devices

The project used several MUD-capable IoT devices, all of which were Micronets Raspberry Pi devkits.

8.2.7.1.1 Micronets Raspberry Pi (Devkit)

The Raspberry Pi devkit runs the Raspbian 9 operating system. It was provisioned with one Wi-Fi Easy Connect bootstrapping public/private key pair before it initiates onboarding. This device is capable of being placed in Easy Connect onboarding mode, at which point it begins displaying a QR code and listening for Wi-Fi Easy Connect protocol messages. The QR code that the device displays contain the device's bootstrapping information, which includes:

- the public bootstrapping key of the device. (i.e., the public key from the public/private key pair that has already been stored on the device)
- MAC address of device
- Wi-Fi-channel the device will use
- information element (i.e., a code that identifies a device vendor)

Note that if the information element field is empty, the device is not considered to be MUD-capable. If the information element field contains a value, however, this value identifies the device's manufacturer. It is assumed that each manufacturer has a well-known location for serving MUD files; therefore, the value in the information element field indicates the location of the device's MUD file server. The value in the public key field, in addition to serving as the device's public key, is also used to indicate which of the files on the device's MUD file server is the device's MUD file.

8.2.7.2 Non-MUD-Capable IoT Devices

The laboratory implementation also includes non-MUD-capable IoT devices. In this case, several Raspberry Pi devices running the Raspbian 9 operating system are utilized.

8.2.8 Update Server

The update server is designed to represent a device manufacturer or trusted third-party server that provides patches and other software updates to the IoT devices. This project used an NCCoE-hosted update server that provides faux software update files.

8.2.8.1 NCCoE Update Server

The NCCoE implemented its own update server by using an Apache web server. This file server hosts software update files to be served as software updates to the IoT device devkits. When the server receives an http request, it sends the corresponding update file.

8.2.9 Unapproved Server

As with Builds 1 and 2, the NCCoE implemented and used its own unapproved server for Build 3. Details are in Section 6.2.11.

8.2.10 MUD Registry

The Micronets MUD Registry resides in the service provider cloud. Its purpose is to provide a lookup service for determining the URL of each device's MUD file. Currently, the Wi-Fi Easy Connect bootstrapping information in the QR code does not include an information field that has been designated to explicitly carry the device's MUD file URL. Instead, the device's MUD file URL is determined indirectly by using two elements of the device's bootstrapping information:

- The information element field contains a code that identifies the device's manufacturer, and it is assumed that each manufacturer has a well-known location for serving MUD files.
- The public key field locates the device's MUD file on that manufacturer's well-known MUD file server.

The Micronets Manager extracts these two items from the device's bootstrapping information, sends them to the MUD registry, and, in response, receives back the URL of the device's MUD file. The Micronets Manager then provides this MUD file URL to the MUD manager.

MUD-based ACLs are enforced only for IoT devices that have bootstrapping information with a vendor code value in the information element field to indicate the location of the device's MUD file server. If the information element field in an IoT device's bootstrapping information is empty, it is assumed that the device does not have a MUD file, and the device will be onboarded without any restraints on its communications other than those imposed by its location on a given micronet.

8.2.11 SDN Controller

The Micronets Manager resides in the service provider cloud. It is responsible for coordinating and managing a collection of micro-services, one of which helps manage the traffic flow rules on the home/small-business network's Micronets Gateway. The Micronets Manager does not use the OpenFlow protocol to configure traffic flow rules to the Micronets Gateway. However, the Micronets Manager effectively functions as an SDN controller insofar as it uses a RESTful interface to the Micronets Gateway to convey the micronets topology and express the ACLs that it wants the gateway to convert to traffic flow rules that the gateway will enforce.

During the onboarding process, the Micronets Manager sends ACLs to the Micronets Gateway to enforce the communications profile specified in the device's MUD file. It also tells the gateway what trust zones (e.g., micronets) to create on the Micronets Gateway and assigns IoT devices to these micronets as directed by user input. The intention is for devices to be assigned to micronets according to policies for that device class, the device functionality, and the desired level of device protection.

Although the Micronets Manager is not currently capable of automatically assigning devices to micronets based on the local communications rules in the device's MUD file, the goal is to be able to automate such assignment in the future.

8.2.12 Onboarding Manager

The Micronets Manager resides in the service provider cloud. It is responsible for a collection of micro-services, one of which is the Micronets Configuration Microservice. The Micronets Configuration Microservice is the managing and controlling element of the Micronets onboarding process, and it effectively serves as the device onboarding manager. During the onboarding process, the Micronets Manager receives the onboarding request and bootstrapping information from the Micronets mobile phone application (via the MSO portal), looks up the device's MUD file URL in the MUD registry, sends the MUD file URL to the MUD manager, and receives back the parsed MUD rules as ACLs from the MUD manager that it sends to the Micronets Gateway. The Micronets Manager provisions the device by providing network configuration for the device (e.g., IP address, assigned micronet, Wi-Fi credentials) and also provides the device's bootstrapping information (e.g., the device's public key, its MAC address, the Wi-Fi channel it will use) to the Micronets Gateway so the Wi-Fi Easy Connect capabilities in the Micronets Gateway can interact with the device to onboard it and place it in the appropriate micronet.

8.2.13 User and Device Interface to the Onboarding Manager

The Micronets mobile phone application acts as both the Micronets Configuration Microservice's user interface and its IoT device bootstrapping information reader. When a device is put into onboarding mode, it displays a QR code that contains its bootstrapping information. A user positions the Micronets mobile phone application so the phone's camera can scan the device's QR code, thereby providing the application with the device's bootstrapping information. The application also requests additional user input regarding the device, including its Micronets class and a device name. The application then sends an onboarding request containing this bootstrapping and user-supplied device information to the Micronets Manager's Configuration Microservice via the service provider's MSO portal.

8.2.14 Bootstrapping Interface to the Onboarding Manager

The MSO portal is part of the service provider's general-purpose cloud infrastructure. It serves as the interface through which the Micronets mobile application can send a device bootstrapping request to the configuration micro-service in the cloud. This service request will include the device bootstrapping information that the Micronets mobile application collects from both the device QR code and the user who is performing the onboarding. The MSO portal forwards this onboarding request and its associated bootstrapping information to the configuration micro-service, which manages the onboarding process in the service provider cloud.

8.2.15 Network Onboarding Component

The Micronets Gateway is Wi-Fi Easy Connect-capable and serves as the network onboarding component. The Wi-Fi Easy Connect onboarding capabilities that reside in the Micronets Gateway are responsible for interacting directly with IoT devices to perform device onboarding. The gateway receives the IoT device's bootstrapping information (e.g., MAC address, public key, Wi-Fi frequency the device will use, MUD ACLs [if any], micronet class, and device name) from the Micronets Manager. After creating and installing any necessary MUD-based flow rules pertaining to the device (if the device is MUD-capable), the gateway initiates the onboarding process with the device by using the Wi-Fi Easy Connect protocol. The gateway authenticates the device, establishes a secure channel with the device, and then, using this secure channel, provides the device with the unique credentials that it needs to connect to the network (e.g., the network service set identifier [SSID] and the device's unique pre-shared key [PSK]). Once the device has been provisioned with its network credentials, it can use those credentials in a standard Wi-Fi handshake to connect to the network via the network access point.

8.3 Build Architecture

In this section we present the logical architecture of Build 3 relative to how it instantiates the reference architecture depicted in Figure 4-1. We also describe Build 3's physical architecture and present message flow diagrams for some of its processes.

8.3.1 Logical Architecture

Figure 8-1 depicts the logical architecture of Build 3. Figure 8-1 uses numbered arrows to depict in detail the flow of messages needed to support installation of MUD-based access control rules for a MUD-capable device. In contrast to Builds 1, 2, and 4, installation of the MUD ACLs in Build 3 occurs when the device is onboarded, prior to the device's connection to the network. This onboarding process is accomplished using the Wi-Fi Easy Connect protocol, the general steps of which are also depicted in Figure 8-1. The other key aspects of the Build 3 architecture (i.e., the Micronets micro-services layer, on-premises Micronets components, Intelligent Services and Business Logic layer, and the update server) are depicted but not described in the same depth as MUD-capable onboarding. To avoid excessive complexity in the depiction, the Micronets APIs are not depicted.

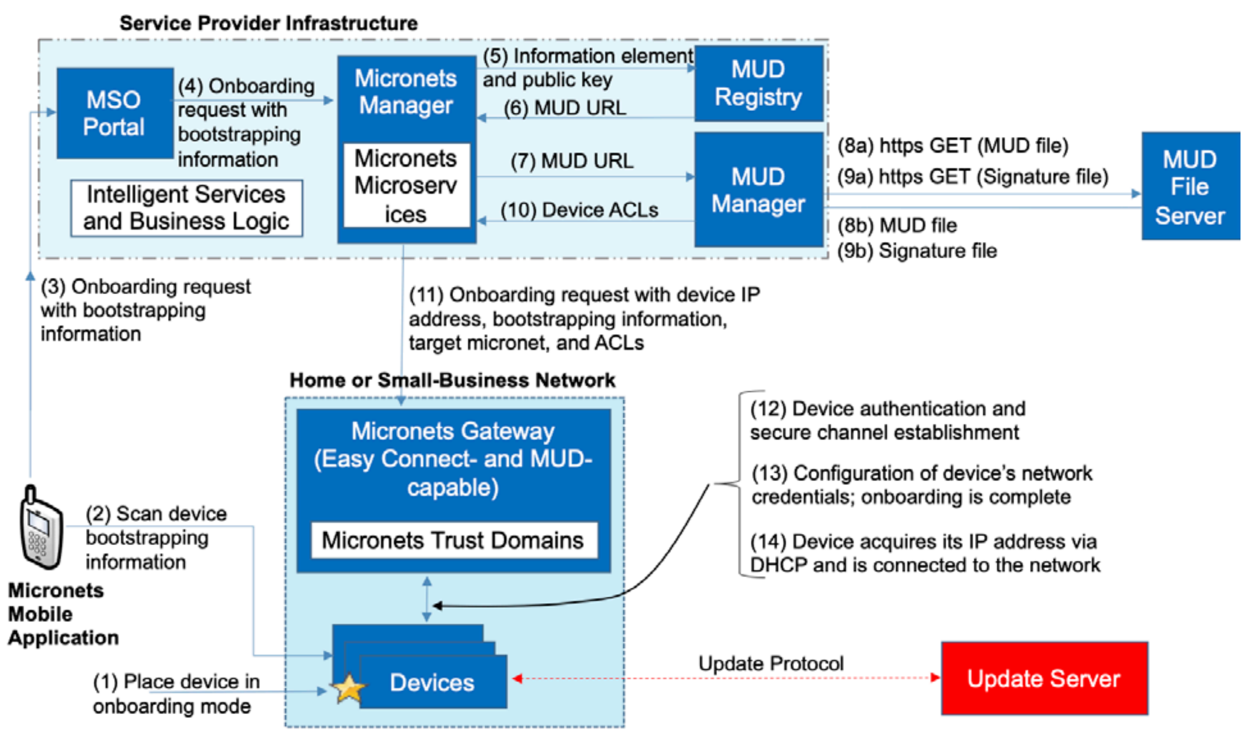
Micronets' logical architecture consists of the following components:

- Micronets mobile application, which supports device onboarding using the Wi-Fi Easy Connect protocol
- On-premises Micronets components, which reside on the home/small-business network. These include the Micronets Gateway, managed services micronets (i.e., micro-networks), and customer micronets. The micro-networks can group devices together into trust domains and isolate them from other devices.

- Cloud-based micro-services layer that hosts various Micronets services. The most important component of this layer is the Micronets Manager, which coordinates the state of the Micronets-enabled on-premises network.
- cloud-based Intelligent Services and Business Logic layer (e.g., machine-learning-based services) that is operated by the service provider
- Micronets API framework, which allows partners and service providers to interface with a customer's micro-networks environment to provision and deliver specific customer-requested services

The logical architecture for Build 3 also includes the notion of ensuring that all IoT devices can access update servers so they can remain up-to-date with the latest security patches. Wi-Fi Easy Connect onboarding of a MUD-capable device using the Micronets mobile application, on-premises Micronets components, the Micronets Microservices layer, the Intelligent Services and Business Logic layer, and the Micronets API framework are each described in their respective subsections below.

Figure 8-1 Logical Architecture—Build 3



8.3.1.1 MUD Capability

As shown in Figure 8-1, Build 3 includes integrated support for MUD in the form of a MUD registry, a MUD manager, a MUD-capable Micronets Manager, and a MUD-capable Micronets Gateway. Support for MUD also requires access to a MUD file server that hosts MUD files for the MUD-capable IoT devices being onboarded.

Build 3 is based on Release 1 of Wi-Fi Easy Connect, which does not include a mechanism for explicitly conveying the device's MUD file URL as part of the device bootstrapping information. To work around this deficiency, Build 3 uses both the information element field and the public key field in the device bootstrapping information to determine the device's MUD file URL. These two fields are used in the following manner:

- The information element field indicates the device's MUD file server. The value in the information element field identifies the device's manufacturer, and it is assumed that each manufacturer has a well-known location for serving MUD files.
- The public key field both conveys the device's public key and identifies the specific file on the manufacturer's MUD file server that is the device's MUD file.
- The Micronets Manager extracts these two values from the bootstrapping information and provides them to the MUD registry lookup service, which in turn responds with the URL of the MUD file associated with an onboarded device. This MUD file URL is then provided to the MUD manager so it can fetch and verify the MUD file.

Future versions of Micronets, subsequent to Build 3, are expected to implement a later version of Wi-Fi Easy Connect (Release 2 or later), which will include a mechanism to optionally and explicitly convey the device's MUD file URL as part of the device onboarding process. Having such a field will greatly simplify the process of conveying the device's MUD file URL to the MUD manager and will obviate the need for a MUD registry.

As shown in Figure 8-1, the flow of messages needed to support installation of MUD-based access control rules for a MUD-capable device in Build 3 is as follows:

- The device must be put into onboarding mode to cause it to display its QR code (which contains its bootstrapping information) and to listen for Wi-Fi Easy Connect protocol messages (step 1).
- The Micronets mobile application is opened and scans the device's QR code. The user also inputs the micronet class to which the device should be assigned, and a device name (step 2).
- The user clicks "onboard," and the application sends the bootstrapping request with the device bootstrapping and other information to the service provider's MSO portal. The Micronets mobile application and the Micronets Manager are each associated with a specific user/subscriber. The onboarding request is sent to the MSO portal so that the portal can ensure that the appropriate Micronets Manager (i.e., the one that is associated with the Micronets mobile application that generated the onboarding request) receives the onboarding request (step 3).
- The MSO portal sends the onboarding request and bootstrapping information to the Micronets Manager (step 4).

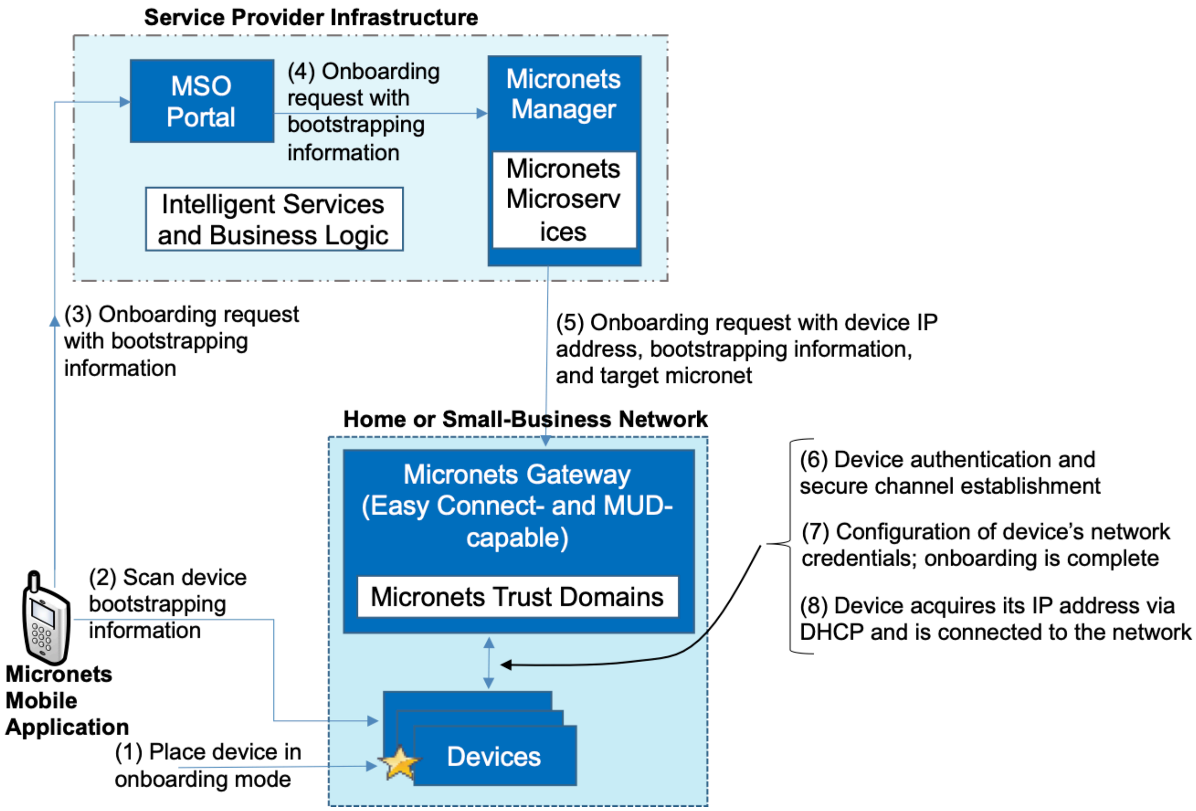
- The Micronets Manager extracts the information element and public key from the bootstrapping information and provides it to the MUD registry (step 5).
- The MUD registry responds with the URL of the device's MUD file (step 6).
- The Micronets Manager provides the MUD file URL to the MUD manager (step 7).
- Once the MUD URL is received, the MUD manager checks its cache to determine if the MUD file is there and, if so, makes sure it has not been there so long that it has exceeded the MUD file caching policy time-out period. If the MUD file is not there or if the file is there but was retrieved too long ago, the MUD manager uses the MUD URL to fetch the MUD file from the MUD file server (step 8a); if successful, the MUD file server at the specified location will serve the MUD file (step 8b).
- Next, the MUD manager requests the signature file associated with the MUD file (step 9a) and upon receipt (step 9b) verifies the MUD file by using its signature file.
- Assuming the MUD file has been verified successfully, the MUD manager parses the MUD file into ACLs that apply to the device and provides these to the Micronets Manager (step 10).
- The Micronets Manager sends these MUD-based ACLs to the on-premises Micronets Gateway, which converts them to traffic flow rules that it installs. These rules ensure that if and when the device connects to the network, it will be subject to the communications policies specified in its MUD file. The device will be permitted only to communicate with local and internet hosts that are explicitly approved in its MUD file, and any other attempts to communicate to or from that device will be blocked (step 11).

  The Micronets Manager also provisions the Micronets Gateway with the device's network configuration and bootstrapping information (e.g., its MAC address, public key, the Wi-Fi channel the device is listening on, the micronet and IP subnet/address to which the device should be assigned, and the device's name) (step 11).
- The Micronets Gateway briefly switches to using the Wi-Fi frequency on which the device is listening (as indicated in the device's bootstrapping information). The Micronets Gateway completes a three-way handshake with the device that constitutes the authentication phase of the Wi-Fi Easy Connect protocol. This protocol exchange serves to both authenticate the device and establish a secure channel with the device (step 12).
- The Micronets Gateway switches back to using its original Wi-Fi frequency. The device switches to using the gateway's frequency and completes a three-way protocol handshake with the device that constitutes the configuration phase of the Wi-Fi Easy Connect protocol. This protocol exchange provisions the device with the credentials that it needs to connect to the network (e.g., the network SSID and the device's unique PSK). At this point, onboarding is complete (step 13).
- The device is now able to connect to the network by presenting its credentials in a standard Wi-Fi handshake (step 14).

8.3.1.2 Wi-Fi Easy Connect Device Onboarding

Figure 8-2 is a simplified version of Figure 8-1. It depicts only the flow of messages needed to support device onboarding in Build 3 (i.e., the message flow needed to support onboarding non-MUD-capable devices).

Figure 8-2 Wi-Fi Easy Connect Onboarding Architecture—Build 3



As shown in Figure 8-2, the flow of messages needed to support onboarding a non-MUD-capable device in Build 3 is as follows:

- The device must be put into onboarding mode to cause it to display its QR code (which contains its bootstrapping information) and to listen for Wi-Fi Easy Connect protocol messages (step 1).
- The Micronets mobile application is opened and scans the device's QR code. The user also inputs the micronet class to which the device should be assigned, and a device name (step 2).
- The user clicks "onboard," and the application sends the bootstrapping request with the device bootstrapping and other information to the service provider's MSO portal (step 3).
- The MSO portal sends the onboarding request and bootstrapping information to the Micronets Manager (step 4).

- The Micronets Manager extracts the public key from the bootstrapping information (because there is no information element, no MUD lookup is performed).
- The Micronets Manager provisions the Micronets Gateway with the device's network configuration and bootstrapping information (e.g., its MAC address, public key, the Wi-Fi channel the device is listening on, the micronet to which the device should be assigned, and the device's name). It also allocates an IP address compatible with the device's target micronet (step 5).
- The Micronets Gateway briefly switches to using the Wi-Fi frequency on which the device is listening (as indicated in the device's bootstrapping information). The Micronets Gateway completes a three-way handshake with the device, which constitutes the authentication phase of the Wi-Fi Easy Connect protocol. This protocol exchange both authenticates the device and establishes a secure channel with the device (step 6).
- The Micronets Gateway switches back to using its original Wi-Fi frequency. The device switches to using the gateway's frequency and completes a three-way protocol handshake with the device, which constitutes the configuration phase of the Wi-Fi Easy Connect protocol. This protocol exchange provisions the device with the credentials that it needs to connect to the network (e.g., the network SSID and the device's unique PSK). At this point, onboarding is complete (step 7).
- The device acquires an IP address via DHCP and is connected to the network (step 8).

8.3.1.3 On-Premises Micronets

The on-premises Micronets consists of the Micronets Gateway, micronets managed by the service provider, and customer micronets, all of which are located on the home/small-business network. The Micronets Gateway is responsible for configuring and enforcing the micronets, which segregate devices. Each micronet represents a distinct trust domain and, at a minimum, represents a distinct IP subnet. IoT devices that are not permitted to exchange traffic with other IoT devices must be placed in separate micronets to isolate them from one another. The Micronets Gateway receives instructions regarding what micronets to set up and assignment of devices to micronets from the Micronets Manager that is in the service provider cloud. The Micronets Gateway is integrated with a Wi-Fi access point, but it supports both wired and wireless connectivity.

8.3.1.3.1 MUD-Driven Policies

The Micronets definition and the placement of devices within a given micronet are governed by the Micronets Manager and are driven by specific policies. Note that the Micronets Manager is associated with the specific user/subscriber who has the on-premises gateway and who is associated with the mobile application. In Build 3, devices are assigned to micronets manually; user input to the Micronets mobile application determines the micronet to which each device will be assigned. Future implementations of Micronets are expected to use MUD-based policies to automatically assign devices to specific micronets.

8.3.1.3.2 Customer Micronets

Customers acquire and connect their own devices. They may even integrate entire service-oriented networks, such as a connected home lighting system. In the future, customer-networked devices may be fingerprinted or authenticated by using an ecosystem certificate (e.g., an Open Connectivity Foundation–certified device) and automatically placed into an appropriate micronet.

8.3.1.4 Micronets Microservices

The Micronets Microservices layer in the service provider cloud hosts several network management-related micro-services that interact with the on-premises Micronets Gateway to manage local devices and network connectivity. One of the core micro-services, the Micronets Manager, coordinates the entire state of the Micronets-enabled on-premises network. It orchestrates the overall delivery of services to the IoT devices and ultimately to the user. The Micronets Manager engages and manages numerous micro-services, including the DHCP/DNS manager, identity server, MUD manager, and MUD registry.

8.3.1.5 Intelligent Services and Business Logic

The Intelligent Services and Business Logic layer resides in the service provider cloud. This architectural component is the interface for the Micronets platform to interact with the rest of the world. It functions as a receiver of the user's intent and business rules from the user's services and is designed to use machine-learning-based services to combine the user's intent and business rules into operational decisions that are handed over to the Micronets micro-services for execution. This layer has not been fully implemented in Build 3. However, it is envisioned that in future versions of Micronets, this layer may receive information from various Micronets micro-services and in turn use that information to dynamically update the access rules for connected IoT devices. For example, to support devices that do not have a MUD file, a "synthetic" MUD file generator and MUD file server could be provided that can host crowdsourced MUD files that are provided to the Micronets micro-services. Other examples include an IoT fingerprinting service that could detect classify devices on the network or an artificial intelligence/machine-learning-based malware detection service that could provide updated MUD files or access policies based on actively detected threats in the network.

8.3.1.6 Micronets API Framework

Each Micronets component (the micro-services as well as the gateway services) exposes a set of APIs that form the Micronets API framework. Some of the APIs can be exposed to allow partners and service providers to interface with the customer's Micronets environment to securely provision and deliver specific services that the customer has requested.

8.3.2 Physical Architecture

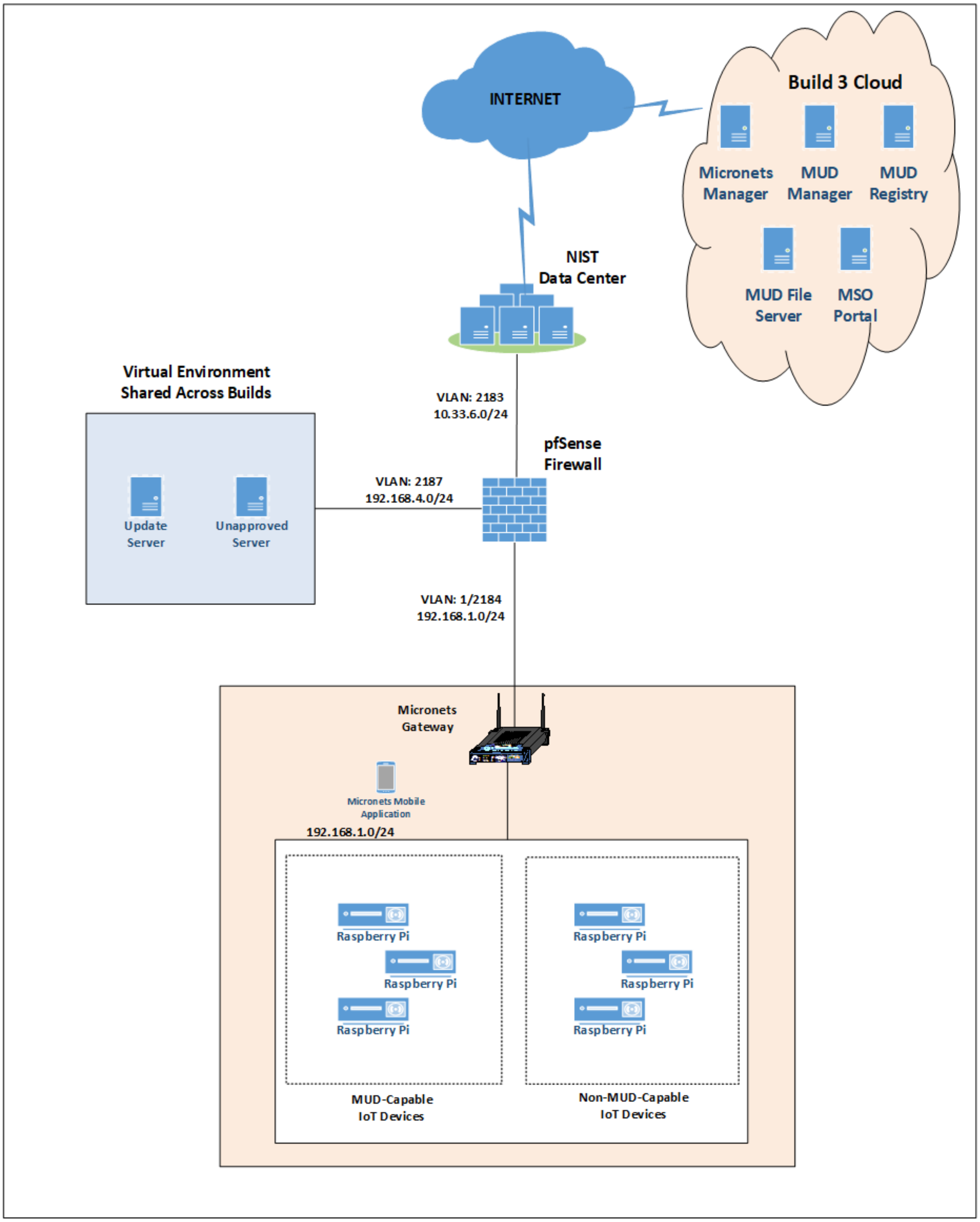
Figure 8-3 depicts the physical architecture of Build 3. The Micronets Gateway that is depicted is an SDN-capable switch. This switch receives instructions from the Micronets Manager in the Build 3 cloud via a RESTful interface. The Micronets Gateway creates and manages various subnetworks (i.e., micronets) on the local network. It allocates an IP address to each MUD-capable and non-MUD-capable IoT device and assigns each device to a specific micronet, which serves as a mechanism to group together devices that are permitted to communicate with one another and to isolate devices that are not. This gateway is also a router configured to enforce the communication constraints of each MUD-capable device as defined in its MUD file. Lastly, the gateway is also Wi-Fi Easy Connect-capable. It uses the Wi-Fi Easy Connect protocol to authenticate devices and provision them with device-specific network credentials. The network infrastructure as configured utilizes the IPv4 protocol for communication both internally and to the internet.

Build 3 also uses a portion of the virtual environment that is shared across builds. Services hosted in this environment include an update server and an unapproved server.

Internet-accessible cloud services are also supported in Build 3. Those depicted include a Micronets Manager, a MUD registry, a MUD manager, and a MUD file server. The Micronets Manager manages a number of different micro-services that are also hosted in the cloud but not depicted, including a configuration micro-service that manages the onboarding process in the service provider cloud.

The Micronets mobile application is also used within the NCCoE laboratory. It runs on a mobile phone and uses that phone’s camera to scan in the QR code of IoT devices. This application serves as the user and device bootstrapping interface for the Wi-Fi Easy Connect onboarding process, requesting user input such as the micronet class and name of each device. This application obtains each device’s bootstrapping information from the device’s QR code and sends it and the user-provided information, along with the onboarding request, to the Micronets Configuration Microservice via the MSO portal. The MSO portal is the fifth cloud service depicted.

2590 **Figure 8-3 Physical Architecture—Build 3**



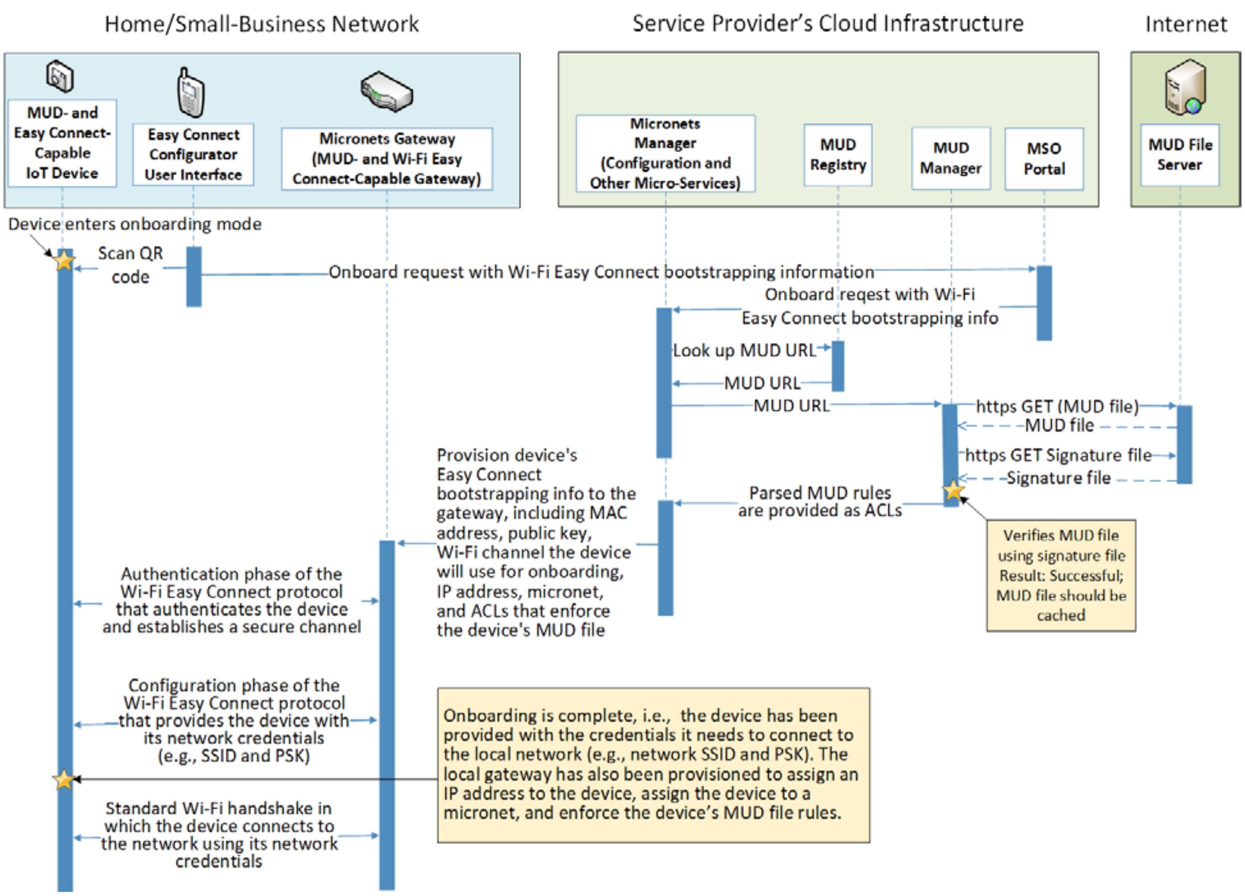
8.3.3 Message Flow

This section presents the message flows used in Build 3 during several different processes of note.

8.3.3.1 Onboarding MUD-Capable Devices

Figure 8-4 depicts the message flows involved in the process of onboarding a MUD-capable device in Build 3, which is accomplished using the Wi-Fi Alliance's Wi-Fi Easy Connect protocol.

Figure 8-4 MUD-Capable IoT Device Onboarding Message Flow—Build 3



The components used to support Build 3 are deployed across the home/small-business network, the service provider cloud, and the internet in general. As shown in Figure 8-4, the onboarding message flow for MUD-capable devices is as follows:

- The IoT device must be placed in onboarding mode. This causes it to display a QR code and to listen for Wi-Fi Easy Connect protocol messages.

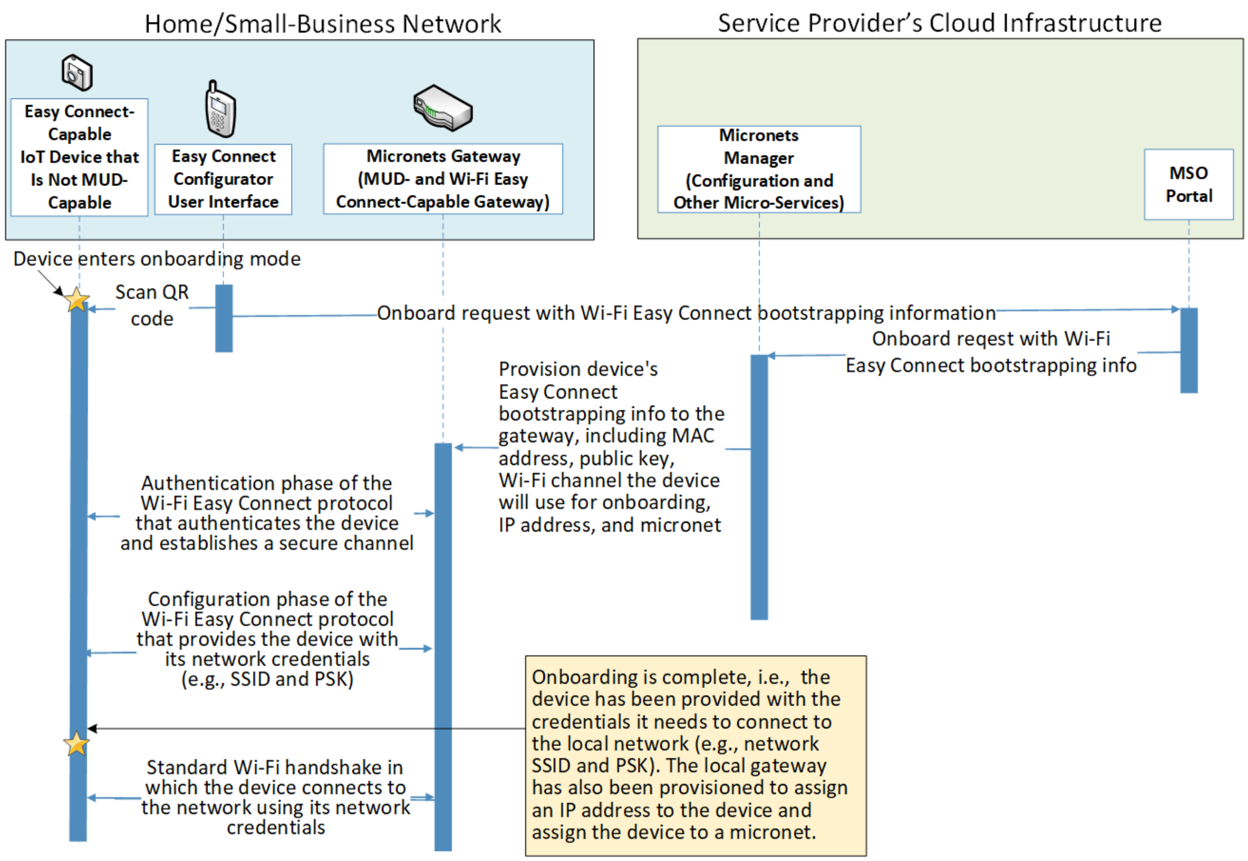
- The Micronets mobile application is opened, “onboard” is selected, and the phone is positioned so that its camera can read the device’s QR code. This provides the device’s bootstrapping information to the configuration element running in the mobile application. The user is also required to input additional device-specific information to the mobile phone application, such as the micronet class of the device and a human-friendly name for the device.
- The mobile phone application sends an onboarding request, along with the device’s bootstrapping and other information, to the service provider’s MSO portal.
- The MSO portal forwards this request and information to the Micronets Manager that is running in the service provider cloud.
- The Micronets Manager sends the information element and the public key field values from the device’s bootstrapping information to the MUD registry.
- The MUD registry responds with the URL for the device’s MUD file.
- The Micronets Manager sends the MUD file URL to the MUD manager.
- The MUD manager fetches the MUD file and the MUD file signature file from the MUD file server.
- After verifying that the MUD file is valid, the MUD manager sends the access control rules that correspond to the MUD file rules to the Micronets Manager.
- The Micronets Manager provisions the device’s bootstrapping information to the Micronets Gateway that is running on the home/small-business network. Specifically, the Micronets Manager provides the gateway with the device’s MAC address, its public key, the Wi-Fi channel on which it will listen for onboarding messages, its micronet, its IP subnet/address, its name, and ACLs needed to enforce the communications profile specified by the device’s MUD file.
- The Micronets Gateway initiates the authentication phase of the Wi-Fi Easy Connect protocol: It sends an authentication request to the IoT device, receives an authentication response from the device, and responds by sending an authentication confirmation to the device. As a result of this exchange, the device has been authenticated, and there is now a secure channel between the Micronets Gateway and the IoT device.
- The IoT device initiates the configuration phase of the Wi-Fi Easy Connect protocol: It sends a configuration request to the Micronets Gateway, receives a configuration response from the Micronets Gateway, and responds by sending a configuration result to the Micronets Gateway. As noted earlier, configuration may happen on a frequency different from the one used for authentication. This completes the onboarding process. As a result of the configuration message it received, the device has learned the SSID and the unique credential that it needs to connect to the home/small-business network. In addition, the Micronets Gateway has been provided with both the micronet to which the device will be assigned upon connection to the network and ACLs that express the device’s communications profile, as specified in its MUD file.

- With onboarding complete, the device initiates a standard Wi-Fi handshake and presents its newly provisioned credentials to connect to the network. It will be assigned its provisioned IP address, it will be located in a micronet that had been specified by the user of the Micronets mobile application at onboarding time, and it will be able to send and receive messages in accordance with both its micronet and the rules specified in its MUD file (i.e., it will not be permitted to communicate with any local devices that are in a different micronet unless such communication is explicitly permitted by its MUD file).

8.3.3.2 Onboarding Non-MUD-Capable Devices

Figure 8-5 depicts the message flows involved in the process of onboarding devices that are Wi-Fi Easy Connect-capable but not MUD-capable in Build 3.

Figure 8-5 Non-MUD-Capable IoT Device Onboarding Message Flow—Build 3



As shown in Figure 8-5, the onboarding message flow for non-MUD-capable devices is as follows:

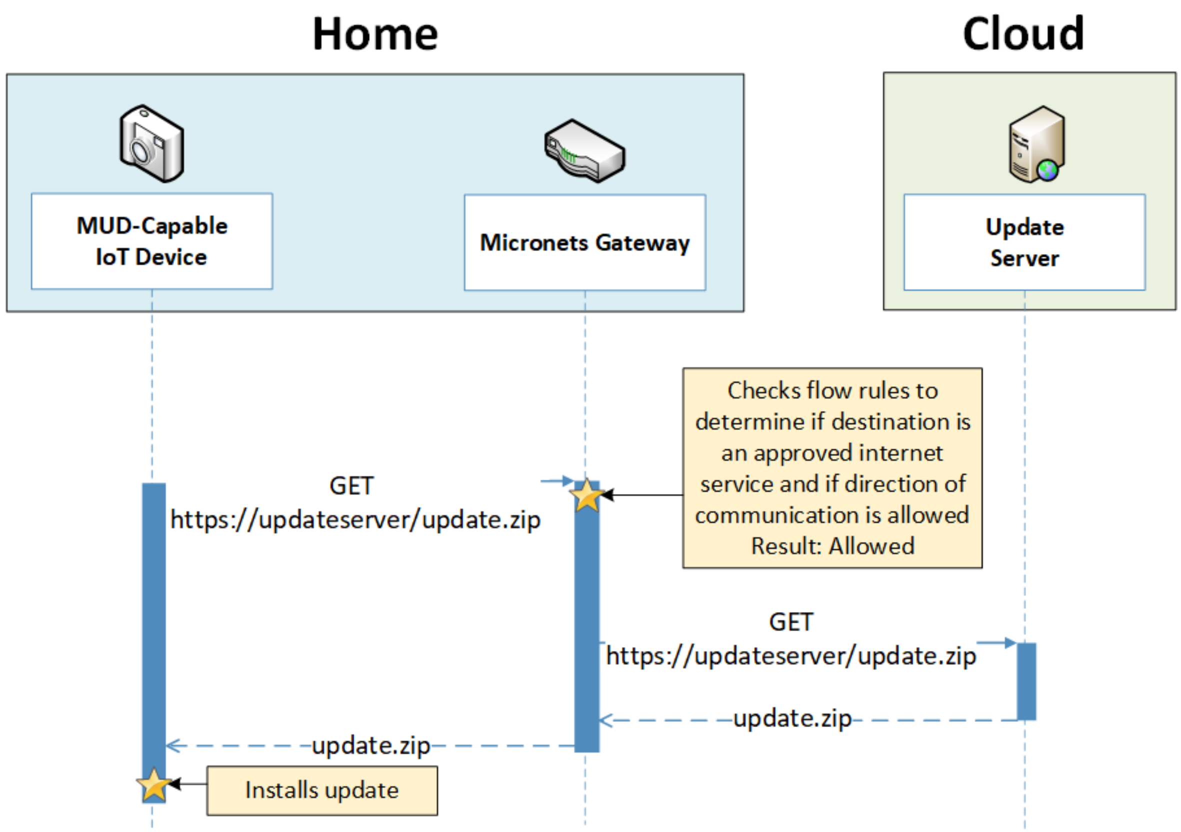
- The IoT device must be placed in onboarding mode. This causes it to display a QR code and to listen for Wi-Fi Easy Connect protocol messages.

- The Micronets mobile application is opened, "onboard" is selected, and the phone is positioned so that its camera can read the device's QR code. This provides the device's bootstrapping information to the configuration element running in the mobile application. The user is also required to input additional device-specific information to the mobile phone application, such as the micronet class of the device and a human-friendly name for the device.
- The mobile phone application sends an onboarding request, along with the device's bootstrapping and other information, to the service provider's MSO portal.
- The MSO portal forwards this request and information to the Micronets Manager that is running in the service provider cloud.
- The Micronets Manager extracts the public key from the bootstrapping information (because there is no information element, no MUD lookup is performed).
- The Micronets Manager provisions the device's bootstrapping information to the Micronets Gateway that is running on the home/small-business network. Specifically, the Micronets Manager provides the gateway with the device's MAC address, its public key, the Wi-Fi channel it will use, its micronet, and its name.
- The Micronets Gateway initiates the authentication phase of the Wi-Fi Easy Connect protocol: It sends an authentication request to the IoT device, receives an authentication response from the device, and responds by sending an authentication confirmation to the device. As a result of this exchange, the device has been authenticated, and there is now a secure channel between the Micronets Gateway and the IoT device.
- The IoT device initiates the configuration phase of the Wi-Fi Easy Connect protocol: It sends a configuration request to the Micronets Gateway, receives a configuration response from the Micronets Gateway, and responds by sending a configuration result to the Micronets Gateway. This completes the onboarding process. As a result of the configuration message it received, the device has learned the SSID and the unique credential that it needs to connect to the home/small-business network. In addition, the Micronets Gateway has been provided with the micronet class to which the device will be assigned upon connection to the network.
- With onboarding complete, the device initiates a standard Wi-Fi handshake and presents its newly provisioned credentials to connect to the network. It will be assigned an IP address, it will be located on the micronet that had been specified by the user of the Micronets mobile application at onboarding time, and it will be able to send and receive messages in accordance with its micronet class (i.e., it will not be permitted to communicate with any local devices that are in a different micronet).

8.3.3.3 Updates

After a device has connected to the home/small-business network, it should periodically check for updates. The message flow for updating the IoT device is shown in Figure 8-6.

Figure 8-6 Update Process Message Flow—Build 3



As shown in Figure 8-6, the message flow is as follows:

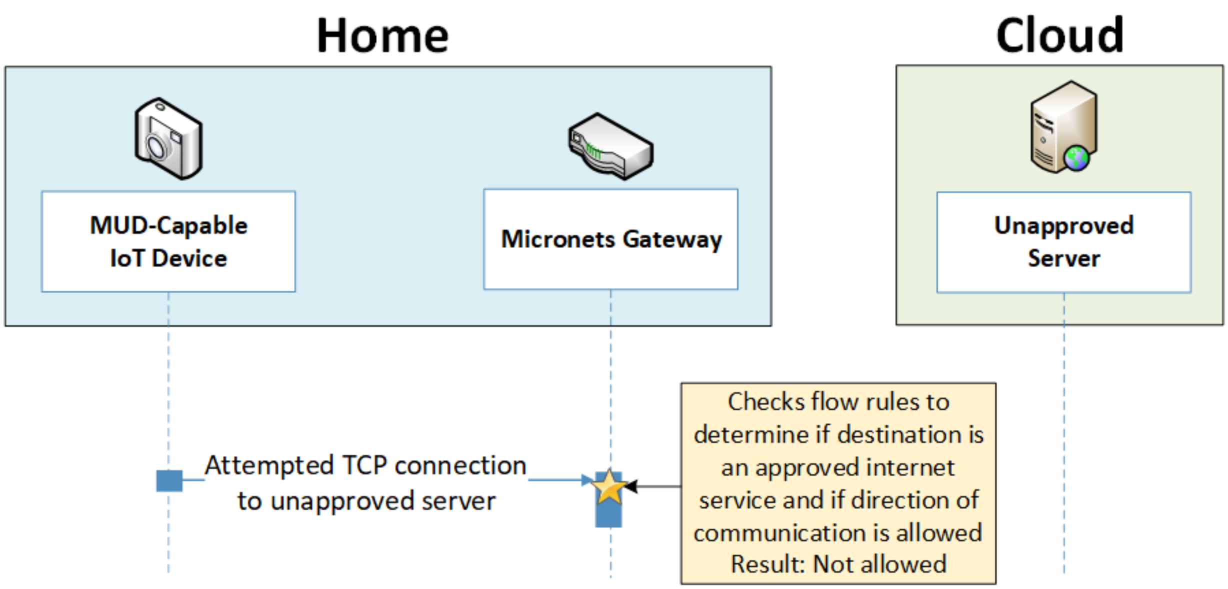
- The device generates an https GET request to its update server.
- The Micronets Gateway will consult the flow rules for this device to verify that it is permitted to send traffic to the update server. Assuming there were explicit rules in the device's MUD file enabling it to send messages to this update server, the Micronets Gateway will forward the request to the update server.
- The update server will respond with a zip file containing the updates.
- The Micronets Gateway will forward this zip file to the device for installation.

8.3.3.4 Prohibited Traffic

Figure 8-7 shows an attempt to send traffic that is prohibited by the MUD file and so is blocked by the Micronets Gateway.

- A connection attempt is made from a local IoT device to an unapproved server. (The unapproved server is located at a domain to which the MUD file does not explicitly permit the IoT device to send traffic.)
- This connection attempt is blocked because there is no flow rule in the Micronets Gateway that permits traffic from the IoT device to the unapproved server.

Figure 8-7 Unapproved Communications Message Flow—Build 3



8.4 Functional Demonstration

A functional evaluation and a demonstration of Build 3 were conducted that involved two types of activities:

- evaluation of conformance to the MUD RFC—Build 3 was tested to determine the extent to which it correctly implements basic functionality defined within the MUD RFC.
- demonstration of additional capabilities—Build 3 supports onboarding via the Wi-Fi Easy Connect protocol and provides the capability to segregate devices onto specific micronets upon connection to the network. Both capabilities were demonstrated.

Table 8-2 summarizes the tests used to evaluate Build 3's MUD-related capabilities, and Table 8-3 summarizes the exercises used to demonstrate Build 3's non-MUD-related capabilities (i.e., its onboarding and Micronets-related functionality). Both tables list each test or exercise identifier, a summary of the test or exercise, the test or exercise's expected and observed outcomes, and the applicable Cybersecurity Framework Subcategories and NIST SP 800-53 controls for which each test or exercise verifies support. The tests and exercises listed in the table are detailed in a separate supplement for functional demonstration results. Boldface text highlights the gist of the information that is being conveyed.

Table 8-2 Summary of Build 3 MUD-Related Functional Tests

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
IoT-1	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate.	**A MUD-capable IoT device that is also Wi-Fi Easy Connect-capable is onboarded** as follows: The device is put into onboarding mode, causing it to display a QR code containing its bootstrapping information and to listen for Wi-Fi Easy Connect messages on the frequency indicated by the QR code. The Micronets mobile onboarding application is opened and scans the QR code. The user provides additional information and clicks "onboard." This causes the device bootstrapping information to be sent to the Micronets Manager via the operator's MSO portal in the service provider cloud. The following operations are then performed automatically: The Micronets Manager looks up the device's MUD file URL in the MUD registry and	The Micronets Gateway will be configured to enforce the policies specified in **the IoT device's MUD file. ACLs will be installed on the gateway** to reflect MUD-filtering rules.	Pass

	NIST SP 800-53 Rev. 4 AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.	provides the URL to the MUD manager. The MUD manager contacts the MUD file server and verifies that it has a valid TLS certificate. The MUD manager requests the MUD file and the MUD signature file and validates the MUD file. The MUD manager parses the MUD rules and translates these to ACLs (route-filtering rules) that it sends to the Micronets Manager. The Micronets Manager provides the device's bootstrapping information and its MUD ACLs to the Micronets Gateway so that the gateway is now configured to enforce the policies specified in the device's MUD file. The gateway briefly switches to the device's frequency and initiates Wi-Fi Easy Connect authentication. The device switches to the gateway's frequency and receives network credentials via Wi-Fi Easy Connect. The device connects to the network.		
IoT-2	**PR.AC-7:** Users, devices, and other assets are authenticated (e.g., sin-	A MUD-capable IoT device initiates the Wi-Fi	The MUD manager will detect that the MUD file server	Pass

	gle-factor, multifactor) commensurate with the risk of the transaction (e.g., individuals' security and privacy risks and other organizational risks). **NIST SP 800-53 Rev. 4** AC-7, AC-8, AC-9, AC-11, AC-12, AC-14, IA-1, IA-2, IA-3, IA-4, IA-5, IA-8, IA-9, IA-10, IA-11	Easy Connect onboarding process, but the **MUD file server that is hosting the device's MUD file does not have a valid TLS certificate.** Therefore, the device's MUD manager drops the connection to the MUD file server. The Micronets Manager provisions the device on the Micronets Gateway as if the device had not been associated with a MUD file. **The device does not have any MUD-related restrictions imposed on its communications.** (Note that it is a local policy decision as to whether an implementation will fail "closed" and restrict all communications or fail "open" and not impose any communications restrictions. Build 3 fails open. In theory, it could also act such as assigning the device to a more restricted micronet.)	does not have a valid TLS certificate and will drop the connection to the MUD file server. According to local policy, **the Micronets Gateway will be configured to permit the device to connect to the network and communicate without any MUD-based restrictions**.	
IoT-3	**PR.DS-6:** Integrity-checking mechanisms verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-capable IoT device initiates the Wi-Fi Easy Connect onboarding process, but the **certificate that was used to sign the MUD file for this device had already expired at**	The MUD manager will detect that the MUD file's signature was created by using a certificate that had already expired at signing. According to local policy, **the**	Pass

		signing. Therefore, the Micronets Manager provisions the device on the Micronets Gateway as if the device had not been associated with a MUD file. **The device does not have any MUD-related restrictions imposed on its communications.** (Note that it is a local policy decision as to whether the implementation will fail "closed" and restrict all communications or fail "open" and not impose any communications restrictions. Build 3 fails open. In theory, it could also act such as assigning the device to a more restricted micronet.)	**Micronets Gateway will be configured to permit the device to connect to the network and communicate without any MUD-based restrictions.**	
IoT-4	**PR.DS-6:** Integrity-checking mechanisms verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-capable IoT device initiates the Wi-Fi Easy Connect onboarding process, but the **signature of the device's MUD file is invalid.** Therefore, the Micronets Manager provisions the device on the Micronets Gateway as if the device had not been associated with a MUD file. **The device does not have any MUD-related restrictions imposed on its communications.**	The MUD manager will detect that the MUD file's signature is invalid. According to local policy, **the Micronets Gateway will be configured to permit the device to connect to the network and communicate without any MUD-based restrictions.**	Pass

		(Note that it is a local policy decision as to whether the implementation will fail "closed" and restrict all communications or fail "open" and not impose any communications restrictions. Build 3 fails open. In theory, it could also act such as assigning the device to a more restricted micronet.)		
IoT-5	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7	Test IoT-1 has run successfully, meaning that the Micronets Gateway has been configured based on a **MUD file that permits traffic to/from some internet locations and implicitly denies traffic to/from all other internet locations.**	When the MUD-capable IoT device is connected to the network, its **Micronets Gateway will be configured to enforce the route filtering that is described in the device's MUD file with respect to** traffic being permitted to/from some **internet locations,** and traffic being implicitly blocked to/from all remaining internet locations.	Pass (for testable procedure –ingress cannot be tested)
IoT-6	**ID.AM-3:** Organizational communication and data flows are mapped.	Test IoT-1 has run successfully, meaning that the Micronets Gateway has been configured	When the MUD-capable IoT device is connected to the	Partial pass. The Micronets Gateway

	NIST SP 800-53 Rev. 4 AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7	based on a **MUD file that permits traffic to/from some lateral hosts and implicitly denies traffic to/from all other lateral hosts.** (The MUD file does not explicitly identify the hosts as lateral hosts; it identifies classes of hosts to/from which traffic should be denied, where one or more hosts of this class happen to be lateral hosts.)	network, its Micronets Gateway **will be configured to enforce the access control information that is described in the device's MUD file with respect to** traffic being permitted to/from some **lateral (local) hosts,** and traffic being implicitly blocked to/from all remaining lateral (local) hosts.	does support protocol-based traffic enforcement for local traffic, but it does not yet support port-level traffic enforcement. Also, as is the case for traffic that originates at internet locations and is inbound toward a MUD-capable IoT device, the gateway does not enforce inbound rules for local communications.
IoT-9	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5	Test IoT-1 has run successfully, meaning the Micronets Gateway has been configured based on the MUD file for a	A domain in the MUD file resolves to two different IP addresses. **The Micronets Manager**	Pass

	ID.AM-2: Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CM-2, SI-4 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM- 5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place.	specific MUD-capable device in question. **The MUD file contains domains that resolve to multiple IP addresses. The Micronets Gateway should be configured to permit communication to or from all IP addresses for the domain.**	**will create flow rules on the Micronets Gateway that permit the MUD-capable device to send traffic to both IP addresses.** The MUD-capable device attempts to send traffic to each of the IP addresses, and the Micronets Gateway permits the traffic to be sent in both cases.	

	NIST SP 800-53 Rev. 4 CM-3, CM-4, SA-10 **PR.DS-2:** Data in transit is protected. **NIST SP 800-53 Rev. 4** SC-8, SC-11, SC-12			
IoT-10	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial	A MUD-capable IoT device is onboarded as described in test IoT-1. As part of this onboarding process, the device's MUD file is retrieved, and the Micronets Gateway is configured to enforce the policies specified in the MUD file for that device. **Within 24 hours (i.e., within the cache-validity period for that MUD file), a second IoT device that is associated with the same MUD file is connected to the network.** After 24 hours have elapsed, a third IoT device that is associated with the same MUD file is connected to the network.	Upon connection of the second IoT device to the network, **the MUD manager does not contact the MUD file server. Instead, it uses the cached MUD file.** It translates this MUD file's contents into appropriate route-filtering rules and provides these to the Micronets Manager for installation onto the Micronets Gateway for the second IoT device. Upon connection of the third IoT device to the network, the MUD manager does fetch a new MUD file.	Pass

	control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.			
IoT-11	**ID.AM-1:** Physical devices and systems within the organization are inventoried.	**A MUD-capable IoT device conveys its MUD file URL via two fields in its bootstrapping information (information element and public key), which are encoded in its QR code. The information element contains a code that identifies the device vendor. It is assumed that each manufacturer has a well-known location for serving MUD files. The public key locates the device's MUD file. A MUD registry is deployed on the service provider cloud that, when provided with the information element and public key**	During the onboarding process, the Micronets Manager extracts the information element and public key field values from the device's bootstrapping information and provides these to the MUD registry. The MUD registry responds with the URL of the device's MUD file**.** The Micronets Manager provides this URL to the MUD manager, and the appropriate MUD file for the device is fetched and used as the basis for the flow rules that are configured on	Pass

		field values from a device's bootstrapping information, responds with the URL of the device's MUD file.	the Micronets Gateway for the device.	

In addition to supporting MUD, Build 3 supports onboarding via the Wi-Fi Easy Connect protocol and provides the capability to place devices onto specific micronets when they are provisioned on the network. Wi-Fi Easy Connect supports easy onboarding of both MUD-capable and non-MUD-capable devices. Micronets are subnetworks that serve to isolate devices. Devices that are on one micronet are not able to exchange traffic with devices on other micronets unless this restriction is overridden by their MUD files. Different micronet classes have been predefined. When a device is onboarded using the Micronets mobile application, the user is asked to input or confirm the class of micronet to which the device should be assigned.

Table 8-3 lists the non-MUD-related (e.g., the Wi-Fi Easy Connect onboarding- and Micronet-related) capabilities that were demonstrated for Build 3.

Table 8-3 Wi-Fi Easy Connect Onboarding- and Micronets-Related Functional Capabilities Demonstrated

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
MnMUD-1	**PR.AC-1:** Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users, and processes. **NIST SP 800-53 Rev. 4** AC-1, AC-2, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11 **NIST SP 800-53 Rev. 4** PE-2, PE-3, PE-4, PE-5, PE-6, PE-8 **PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15	Demonstrates that **non-MUD-capable devices that are Wi-Fi Easy Connect-capable can be onboarded using the Wi-Fi Easy Connect protocol and that, once onboarded, can successfully connect to the network with the credentials they were provided during onboarding; and that they are assigned to the correct micronet.** Specifically, the follow-	**Both devices can successfully connect to the network, and they can send and receive messages to and from each other**.	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	PR.AC-4: Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected (e.g., network segregation, network segmentation). **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7	ing steps are performed for two separate IoT devices: The device is put into onboarding mode, causing it to display a QR code containing its bootstrapping information and to listen for Wi-Fi Easy Connect messages on the frequency indicated by the QR code. The Micronets mobile onboarding application is opened and scans the QR code. The user assigns the device to a particular micronet and clicks “onboard.” This causes the device bootstrapping information to be sent to the Micronets Manager via the operator’s MSO portal in the service provider cloud. The following operations are then performed automatically: The Micronets Manager provides the device’s bootstrapping information and its MUD ACLs to the Micronets Gateway. The gateway briefly switches to the device’s frequency and initiates Wi-Fi Easy		

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
		Connect authentication to authenticate the device and establish a secure channel with it. The device switches to the gateway's frequency and initiates the Wi-Fi Easy Connect configuration phase to receive its network credentials from the gateway. The device connects to the network. Note that both IoT devices are assigned to the same micronet class.		
MnMUD -2	**PR.AC-1:** Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users, and processes. **NIST SP 800-53 Rev. 4** AC-1, AC-2, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11 **NIST SP 800-53 Rev. 4** PE-2, PE-3, PE-4, PE-5, PE-6, PE-8 **PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24	Demonstrates that **devices that are assigned to the same micronet can communicate with each other but not with devices in a different micronet**. Run exercise MnMUD-1, with the result that there are two devices connected to the correct network (Device 1 and Device 2), and they are on the same micronet. Run exercise MnMUD-1 for a third device, but this time assign the device to a different micronet class in step 7a and name it Device 3 in step 7b.	Non-MUD-capable devices can be onboarded with the network credentials necessary to ensure that they connect to the correct network and, once connected, are assigned to the correct micronet. **Devices in the same micronet can communicate with one another, but devices in different micronets cannot.**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	PR.AC-5: Network integrity is protected (e.g., network segregation, network segmentation). **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7	Verify that Device 1 and Device 2 (which are both on micronet CLASS 1) can send and receive messages to and from each other. Verify that neither Device 1 nor Device 2 can send or receive messages to or from Device 3 (which is on micronet CLASS 2).		
MnMUD-3	**PR.AC-1:** Identities and credentials are issued, managed, verified, revoked, and audited for authorized devices, users, and processes. **NIST SP 800-53 Rev. 4** AC-1, AC-2, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11 **NIST SP 800-53 Rev. 4** PE-2, PE-3, PE-4, PE-5, PE-6, PE-8 **PR.AC-3:** Remote access is managed. **NIST SP 800-53 Rev. 4** AC-1, AC-17, AC-19, AC-20, SC-15 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC- 5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected (e.g., network segregation, network segmentation). **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7	Run exercise MnMUD-1, with the result that there are two devices connected to the correct network (Device 1 and Device 2), and they are on the same micronet. Run exercise MnMUD-1 for a third device (Device 3), and assign this device to the same micronet class as the first two devices. Verify that all three devices are connected to the correct network and can exchange messages with one another. Then configure the gateway to **revoke the credentials of Device 2. Verify that Device 2 cannot send messages to or receive messages from Device 1 or Device 3. Verify**	After multiple IoT devices have been onboarded and connected to the network, **the credentials of one of these devices can be revoked at the Micronets Gateway, causing that device to be disconnected. The other devices, which have their own unique credentials, remain connected**.	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
		that Device 1 and Device 3 can send messages to and from each other.		

8.5 Observations

Build 3 was able to successfully onboard IoT devices using the Wi-Fi Easy Connect protocol, assign those devices to the appropriate micronet class based on user input, and, if the devices are MUD-capable, permit and block traffic to and from the devices as specified in the devices' MUD files. Build 3 was also able to constrain communications to and from local devices (both MUD-capable and non-MUD-capable) based on the micronet class to which the devices were assigned.

We observed the following limitations to Build 3 that are informing improvements to its current proof-of-concept implementation:

- MUD manager:
  - Port/protocol-level traffic filtering is not supported in this version of the MUD manager. If a MUD file rule permits some type of communication between two local devices using a specific port or protocol, Build 3 erroneously permits this communication between those two local devices using all ports. It does not matter whether the MUD rule is specified using port numbers (e.g., 80/443) or protocols (UDP/TCP); neither level of traffic filtering is supported.
- micronets assignment:
  - Within a micronet, all devices can communicate with one another. To enforce the lateral communications rules specified in a device's MUD file, only devices whose MUD files explicitly permit them to communicate with one another should be assigned to the same micronet. Build 3 currently requires assignment of devices to micronets to be performed manually by the user who operates the Micronets mobile application during onboarding. It may not be realistic to expect this user to be familiar with the contents of the device's MUD file and know how to assign devices to micronets accordingly. Ideally, the assignment of devices to micronets should be performed automatically, with the Micronets Manager examining the MUD file rules for the device and, based on those rules, automatically assigning the device to micronets that will enforce the device's local communications profile. Such automatic assignment of devices to micronets, however, is not yet supported. Currently, the only way to ensure that only local communications that are explicitly permitted by the MUD file will be permitted is for the user who is performing the

onboarding to manually assign each device to its own separate micronet. Future implementations of the Micronets Manager may be capable of automatically adding devices with similar local-network restrictions into discrete micronets.

- conveyance of the device's MUD file URL:
    - Build 3 implements Wi-Fi Easy Connect protocol Release 1, which was the current version at the time. Wi-Fi Easy Connect Release 1 does not have a mechanism for conveying the device's MUD file URL in the device bootstrapping information. As a result, Build 3 relies on a workaround to indicate the URL of the MUD file associated with a device. As described previously, this workaround uses the information element field and the public key field in the device bootstrapping information. It also relies on a MUD registry lookup service and an assumption that every manufacturer has a well-known location for serving MUD files. On the other hand, the most recent version of Wi-Fi Easy Connect, Release 2, as specified in the Wi-Fi Alliance's DRAFT Device Provisioning Protocol Specification Version 1.2, does define a mechanism for optionally including the device's MUD file URL in the device bootstrapping information that is conveyed. Future versions of Micronets, subsequent to Build 3, are expected to simply implement the latest Wi-Fi Easy Connect release (Release 2 or later) and will thereby greatly simplify the process of conveying the device's MUD file URL to the MUD manager. Anyone desiring to duplicate the Build 3 implementation in their own environment must either provide their own MUD registry or use the MUD registry created by CableLabs, which CableLabs has offered to make available for this purpose.
- authenticating the association between a device and its MUD file URL

    It is worth noting that the MUD registry that is implemented in Build 3 serves not just as a mechanism for locating each device's MUD file. Assuming that the registry is trusted, it also serves to authenticate the association between the device and its MUD file. When using Build 3, the assumption is that the central registry is a trusted and reliable entity with which each vendor has registered the location of its MUD file server (or the location of a secondary registry that can be used to locate that vendor's MUD file servers). Therefore, this central registry can be trusted to provide a valid association between each device and its MUD file or between each device and the vendor-specific registry that will point to the particular MUD file. The MUD registry architecture that is in place to support the central registry and vendor-specific subregistries in Build 3 is nontrivial; there are no shortcuts when it comes to providing an authenticated association between a device and its MUD file.

    Once Easy Connect Release 2 is implemented, the MUD registry will no longer be necessary. The association between the device and its MUD file will be provided by inclusion of the MUD URL in the device bootstrapping information. Trust in this association will rely on the manufacturer's root of trust, i.e., on the trustworthiness of the certificate authority that signed the certificate for the manufacturer that signed the MUD file. Hence, to be able to trust that a MUD file is in fact correctly associated with a particular device, either:

- The certificate authority that signed the device manufacturer's certificate must be trusted, (as will be the case when Easy Connect Release 2 is implemented) or
- The association between the device and its MUD file must be provided by a central registry that everyone trusts (as is the case in Build 3).

We observed the following benefit of Build 3:

- MUD configuration during onboarding avoids periods during which connected MUD-capable devices are permitted to communicate unrestrained.
  - In implementations other than Build 3 that configure the MUD-related traffic flow rules during device connection, there may be small windows of time during which a device is permitted unrestricted communications while its MUD file is being requested and processed, before the MUD rules related to the device are applied. Because Build 3 configures the MUD-related traffic flow rules on the Micronets Gateway during onboarding, before the device is provisioned with its network credentials, it is not possible for there to be a time period during which the device is connected to the network before its MUD traffic flow rules are provisioned on the gateway.
- Use of Wi-Fi Easy Connect in Build 3 enables each device to be provisioned with its own unique network credentials.
  - Per-device credentialing ensures that even if the credentials of one device are known, these credentials cannot be presented by other devices (e.g., devices that are not authorized to connect to the network) to gain access to the network.
  - Per-device credentialing enables the credentials of some devices to be revoked or changed without interfering with the ability of other devices to connect to the network.
- Network credentials are provisioned to each device via an automated protocol, thereby minimizing the opportunity for human error.
- Network credentials are provisioned to each device over a secure channel, minimizing the possibility of their disclosure. No human being has an opportunity to be privy to the credentials of any device.

9 Build 4

The Build 4 implementation uses software developed at the NIST Advanced Networking Technologies laboratory that is called NIST-MUD. The purpose of this implementation is to serve as a working prototype of the MUD RFC to demonstrate feasibility and scalability. NIST-MUD is intended to provide a platform for research and development by industry and academia. It is released as a simple, minimal, open-source reference implementation of an SDN controller/MUD manager on GitHub.

The NIST MUD manager is implemented as a feature that is running on an OpenDaylight SDN controller. The SDN controller/MUD manager uses the OpenFlow (1.3) protocol to configure the MUD rules on an SDN-capable switch that is deployed on the home or small-business network. Build 4 also uses certificates from DigiCert.

9.1 Collaborators

Collaborators that participated in this build are described briefly in the subsections below.

9.1.1 NIST Advanced Networking Technologies Laboratory

The NIST Advanced Networking Technologies lab mission is networking research and advanced prototyping of emerging standards.

9.1.2 DigiCert

See Section 6.1.2 for a description of DigiCert.

9.2 Technologies

Table 9-1 lists all of the products and technologies used in Build 4 and provides a mapping among the generic component term, the specific product used to implement that component, and the security control(s) that the product provides. When applicable, both the Function Subcategories that a component provides directly and those that it supports but does not provide directly are listed and labeled as such. For rows in which the provides/supports distinction is not noted, all listed Categories are directly provided by the component. Some functional Subcategories are described as being directly provided by a component. Others are supported but not directly provided by a component. Refer to Table 5-1 for an explanation of the NIST Cybersecurity Framework Subcategory codes.

Table 9-1 Products and Technologies

Component	Product	Function	Cybersecurity Framework Subcategories
SDN controller	OpenDaylight SDN Controller	Used to manage the SDN switch on the home/small-business network. Provides a protocol stack on top of which the MUD manager is built; includes an OpenFlow plug-in that is used	Provides ID.AM-3 PR.PT-3

Component	Product	Function	Cybersecurity Framework Subcategories
		to send flow rules to the SDN switch.	
MUD manager	NIST-MUD SDN controller/MUD manager (implemented as a feature on an OpenDaylight open-source SDN controller)	Fetches, verifies, and processes MUD files from the MUD file server maintained by the manufacturer; can also receive MUD files through a Representational State Transfer (REST) API if a manufacturer does not provide a MUD file server. Parses MUD files and converts them to flow rules. Eavesdrops on IoT device DNS requests to obtain the IP address values to insert into flow rules when instantiating MUD file access control entries (ACEs).	Provides PR.PT-3 Supports ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 DE.AE-1
MUD file server	NCCoE-hosted Python (requests)-based https server	Hosts MUD files and signature files; serves MUD files to the MUD manager by using https	ID.AM-1 ID.AM-2 ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
MUD file maker	MUD file maker (https://www.mud-maker.org/)	GUI used to create example MUD files	ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
MUD file	A YANG model instance that has been serialized in JSON (RFC 7951). The manufacturer of a MUD-capable device creates that device's MUD file. MUD file maker (see previous row) can be used to create MUD files. Each MUD file is also associated with a separate MUD signature file.	Specifies the communications that are permitted to and from a given device	Provides PR.PT-3 Supports ID.AM-1 ID.AM-2 ID.AM-3
DHCP server	dnsmasq DHCP server	Functions as a generic DHCP server; does not provide any MUD-specific functions	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
Router or switch	Northbound Networks wireless SDN switch	Routes traffic on the home/small-business network. Gets configured with OpenFlow 1.3 flow rules that enforce MUD file ACEs.	ID.AM-3 PR.AC-4 PR.AC-5 PR.DS-5 PR.PT-3 DE.AE-1
Certificates	DigiCert Premium Certificate	Used to sign MUD files and generate corresponding signature file	PR.AC-1 PR.AC-3 PR.AC-5 PR.AC-7
MUD-capable IoT device 1 (has MUD file profile1)	Raspberry Pi Model 3	Emits a MUD URL as part of its DHCP REQUEST	ID.AM-1
Second MUD-capable IoT device (has MUD file profile1)	Raspberry Pi model 3	Emits a MUD URL as part of the DHCP REQUEST. Acts as the second device made by the same manufacturer as device 1.	ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
Third MUD-capable IoT device (has MUD file profile2)	Raspberry Pi Model 3	Emits a MUD URL as part of the DHCP REQUEST. Acts as a device made by another manufacturer (so we can test interactions between the first type of device and the second type of device).	ID.AM-1
Non-MUD-capable IoT device	Raspberry Pi without a MUD profile	Acts as a typical IoT device on the home/small-business network; does not emit a MUD URL and does not have an associated MUD file. Its traffic is unrestricted.	ID.AM-1
Controller	Raspberry Pi without a MUD profile	Acts as a device controller for the first MUD-enabled device	
Update server	NCCoE-hosted Raspberry Pi Python (request)-based servers (two are used)	Acts as a device manufacturer's update server that would communicate with IoT devices to provide patches and other software updates	PR.IP-1 PR.IP-3
Unapproved server	Raspberry Pi running a web server	Acts as an internet host that has not been explicitly approved in a MUD file	DE.DP-3 DE.AM-1

9.2.1 SDN Controller

The switch on the home/small-business network is an SDN switch that is managed by an OpenDaylight SDN controller. OpenDaylight provides protocol stacks on top of which the MUD manager is built. In Build 4, the protocol stack used is a southbound protocol plug-in for the OpenFlow 1.3 protocol that is used by OpenDaylight applications (e.g., the MUD manager) to send flow rules to the OpenFlow-enabled SDN switch on the home/small-business network. OpenDaylight also allows applications to export “northbound” RESTCONF/YANG model APIs that are primarily used for configuration purposes.

9.2.2 MUD Manager

The MUD manager is an OpenDaylight application written in Java. OpenDaylight uses the Apache Karaf Open Service Gateway Initiative container. The MUD manager is a Karaf feature that uses OpenDaylight libraries and bundles. The IETF-published YANG model for MUD is imported into OpenDaylight directly for the MUD manager implementation.

The MUD manager receives the MUD URL for an IoT device, fetches that MUD file and its corresponding signature file, and uses the signature file to verify the validity of the MUD file. If signature verification succeeds, the MUD manager generates SDN flow rules corresponding to the ACEs that are in the MUD file and pushes them to the SDN switch on the home/small-business network by using the OpenFlow protocol. The instantiation of some flow rules (i.e., those relating to DNS names that have not yet been resolved) may have to be deferred because the IP addresses to be inserted into the flow rules corresponding to these ACEs depend on domain name resolution as seen by the IOT device, which may not yet have been performed. If domain name resolution is performed by a device on the home/small-business network for any domain name that is referenced by a flow rule, the flow rule will be instantiated and sent to the SDN switch.

If signature verification fails or if the MUD file is not retrievable (for example, if the manufacturer website is down or does not have a valid TLS certificate), the MUD manager sends packet classification flow rules to the SDN switch that cause the device to be blocked. In a blocked state, the device may only access DHCP, DNS, and NTP services on the network. This effectively quarantines the device until the MUD file may be verified.

The MUD manager can manage multiple switches. The system achieves memory scalability by a multiple flow table design that uses O(N) flow rules for N distinct MAC addresses seen at the switch.

9.2.3 MUD File Server

In the absence of a commercial MUD file server for use in this project, the NCCoE implemented its own MUD file server by using a Python (requests)-based web server. This file server serves the MUD files along with their corresponding signature files for the IoT devices used in the project. Upon receiving a GET request for the MUD files and signatures, it serves the request to the MUD manager by using https.

9.2.4 MUD File

We test interactions between two manufacturers and between two devices made by the same manufacturer. To accomplish this, two MUD files are defined (referred to as "profile1" and "profile2" in the table above).

9.2.5 Signature File

According to the IETF MUD specification, "a MUD file MUST be signed using CMS as an opaque binary object." The MUD files were signed with the OpenSSL tool by using the command described in the specification (as detailed in Volume C of this guide). A Premium Certificate, requested from DigiCert, was leveraged to generate the signature files. Once created, the signature files are stored on the MUD file server along with the MUD files. The certificate is added to the trust store of the Java Virtual Machine running the MUD manager to enable signature verification.

9.2.6 DHCP Server

NIST-MUD is a Layer-2 implementation. Devices are identified by MAC addresses. NIST-MUD is designed to work with devices that join the network by issuing a DHCP request.

DHCP requests for MUD-enabled devices may contain a MUD URL. The DHCP request (with embedded MUD URL) is sent to the SDN switch, which forwards it simultaneously to the SDN controller/MUD manager and the DHCP server. This is accomplished via an SDN flow rule that is inserted by the MUD manager into the switch flow table when the switch connects to the MUD manager. After extracting the MUD URL from the DHCP packet, the MUD manager proceeds to retrieve the MUD file that is pointed to by the MUD URL.

Because the SDN switch forwards the DHCP request to the MUD manager rather than the DHCP server forwarding the DHCP request to the MUD manager, no modifications to the DHCP server are needed. The MUD manager instead of the DHCP server is responsible for stripping the MUD URL out of the DHCP request. Therefore, Build 4 can use a generic DHCP server that is not required to support any MUD-specific capabilities.

9.2.7 Router/Switch

The switch used on the home/small-business network is a wireless SDN switch that comes bundled with the Northbound Networks Wireless Access Point. The access point bundles a NAT router, DNS server, and DHCP server. The SDN controller/MUD manager is connected to the public-facing side of the switch's NAT component. The switch is OpenFlow-enabled and interacts with its SDN controller/MUD manager via the OpenFlow 1.3 protocol. The SDN switch serves as the enforcement point for MUD policy. Packets sent between devices, between devices and controllers referenced in MUD files, and between devices and the internet must pass through the switch, which is where enforcement occurs.

9.2.8 Certificates

DigiCert provisioned a Premium Certificate for signing the MUD files. The Premium Certificate supports the key extensions required to sign and verify CMS structures as required in the MUD specification. Further information about DigiCert's CertCentral web-based platform, which allows for provisioning and managing publicly trusted X.509 certificates, can be found in Section 6.2.8.

9.2.9 IoT Devices

This section describes the IoT devices used in the laboratory implementation. There are two distinct categories of devices: devices that can emit a MUD URL in compliance with the MUD specification, i.e., MUD-capable IoT devices; and devices that are not capable of emitting a MUD URL in compliance with the MUD specification, i.e., non-MUD-capable IoT devices.

9.2.9.1 MUD-Capable IoT Devices

Three Raspberry Pi devkits used on the home/small-business network are designated as MUD-capable. Two emit the same MUD URL (corresponding to profile1) and the third emits a different MUD URL (corresponding to profile2).

9.2.9.2 Non-MUD-Capable IoT Devices

A fourth Raspberry Pi on the home/small-business network functions as a non-MUD-capable IoT device. Because it does not have an associated MUD file, its communications are not restricted.

9.2.10 Controller and My-Controller

A fifth Raspberry Pi device on the home/small-business network is designated as controller and my-controller. Note that a host cannot simultaneously be designated as a controller and be part of the local network. Hence, the Raspberry Pi that performs this function is not part of the local network category.

9.2.11 Update Server

The update server is designed to represent a device manufacturer or trusted third-party server that provides patches and other software updates to the IoT devices. This project used an NCCoE-hosted update server that provides faux software update files.

9.2.11.1 NCCoE Update Server

The NCCoE implemented its own update server by using an Apache web server. This file server hosts faux software update files to be served as software updates to the IoT device devkits. When the server receives an http request, it sends the corresponding faux update file.

In Build 4, there are two update servers, both of which are Raspberry Pi hosts on the public side of the switch. The DNS server on the switch is configured to return two addresses corresponding to the DNS name of the update server (e.g., www.nist.local maps to two IP addresses). This enables us to test access control when multiple addresses are returned from a DNS lookup.

9.2.12 Unapproved Server

A Raspberry Pi running a web server acts as an unapproved internet host and is used to test the communication between a MUD-capable IoT device and an internet host that is not included in the device's MUD file, so the IoT device should not be permitted to send traffic to it. To verify that the traffic filters were applied as expected, communication to and from the unapproved server and the first MUD-capable IoT device (with profile1) was tested. This unapproved server (www.antd.local) maps to a single IP address and is set up on the public side of the switch.

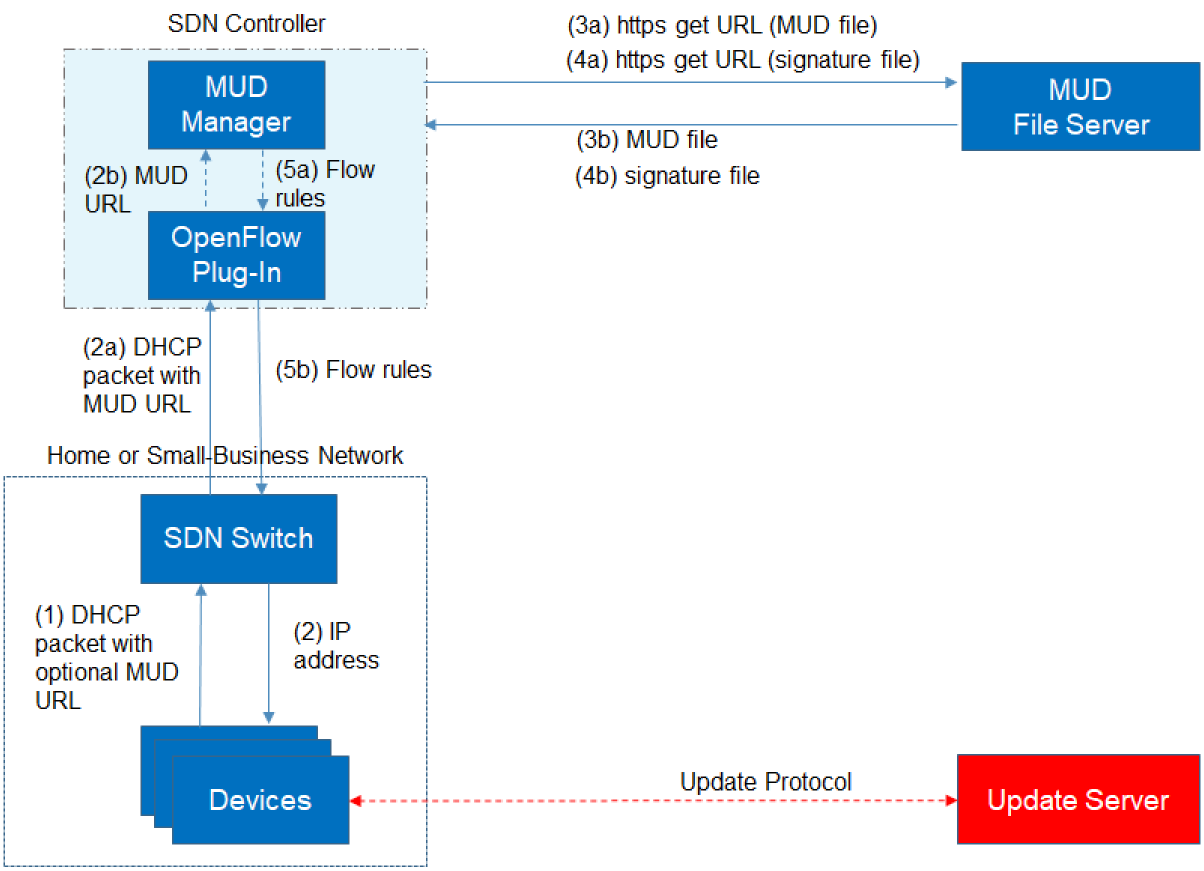
9.3 Build Architecture

In this section we present the logical architecture of Build 4 relative to how it instantiates the reference architecture depicted in Figure 4-1. We also describe Build 4's physical architecture and present message flow diagrams for some of its processes.

9.3.1 Logical Architecture

Figure 9-1 depicts the logical architecture of Build 4. It includes a single device that serves as the SDN controller/MUD manager, which is assumed to be cloud-resident. This SDN controller/MUD manager controls and manages an OpenFlow-enabled SDN switch on the home/small-business network. The SDN switch serves as the MUD policy enforcement point for MUD-capable IoT devices that connect to the home/small-business network. The only automatic MUD URL discovery capability that Build 4 supports is emission of the MUD URL via DHCP. Build 4 does not support LLDP-based or certificate-based MUD URL discovery. However, it is also possible to associate a MUD file with a device that is not capable of emitting a MUD URL by manually associating that device's MAC address with a MUD file URL when using Build 4.

Figure 9-1 Logical Architecture—Build 4



As shown in Figure 9-1, the steps that occur when a MUD-capable IoT device connects to the home/small-business network using Build 4 are as follows:

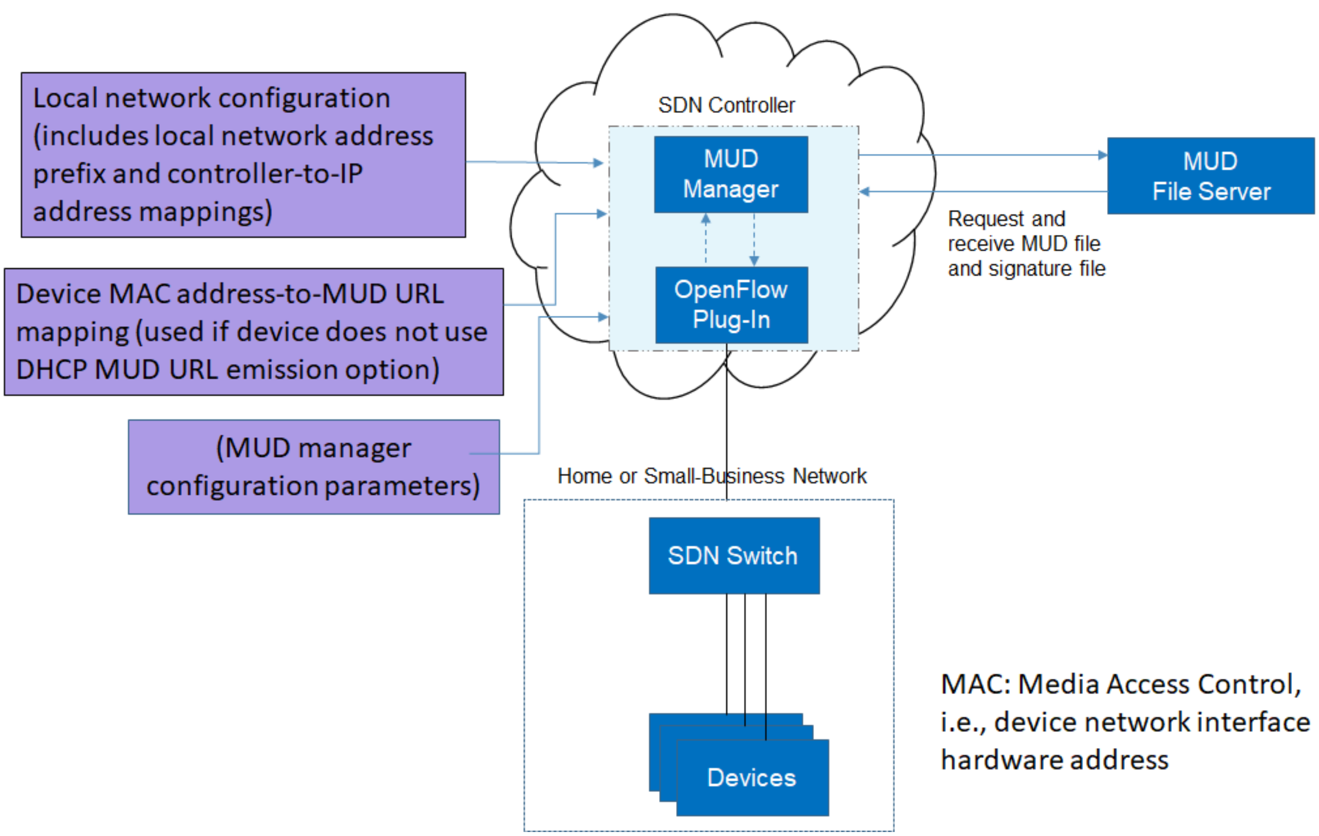
- Upon connecting a MUD-capable device, the MUD URL is emitted via DHCP (step 1).
- The SDN switch sends the DHCP packet containing the MUD URL to the SDN controller/MUD manager via the OpenFlow protocol (step 2a); this is passed from the OpenFlow plug-in to the MUD manager (step 2b).
- Simultaneously, the device is assigned an IP address (step 2).
- Once the DHCP packet is received at the MUD manager, the MUD manager extracts the MUD URL from the DHCP packet and requests the MUD file from the MUD file server by using the MUD URL (step 3a); if successful, the MUD file server at the specified location will serve the MUD file (step 3b).

- Next, the MUD manager requests the signature file associated with the MUD file (step 4a) and upon receipt (step 4b) verifies the MUD file by using its signature file.
- After the MUD file has been verified successfully, the MUD manager creates flow rules corresponding to the MUD file ACEs and provides these to the OpenFlow plug-in (step 5a), which in turn sends the flow rules to the SDN switch, where they are applied (step 5b).

Once the device's flow rules are installed at the SDN switch, the MUD-capable IoT device will be able to communicate with approved local hosts and internet hosts as defined in the MUD file, and any unapproved communication attempts will be blocked. Devices that are not MUD-capable will not have their communications restricted in any way by the MUD manager, assuming they have not been manually associated with a MUD file.

Figure 9-2 depicts some configuration information that can be provided to the Build 4 SDN controller/MUD manager via its REST API.

Figure 9-2 Example Configuration Information for Build 4



As shown in Figure 9-2, the MUD manager exports a YANG-based REST API to allow administrators to configure the SDN controller/MUD manager. This API is not exposed to the network users. It provides the following capabilities:

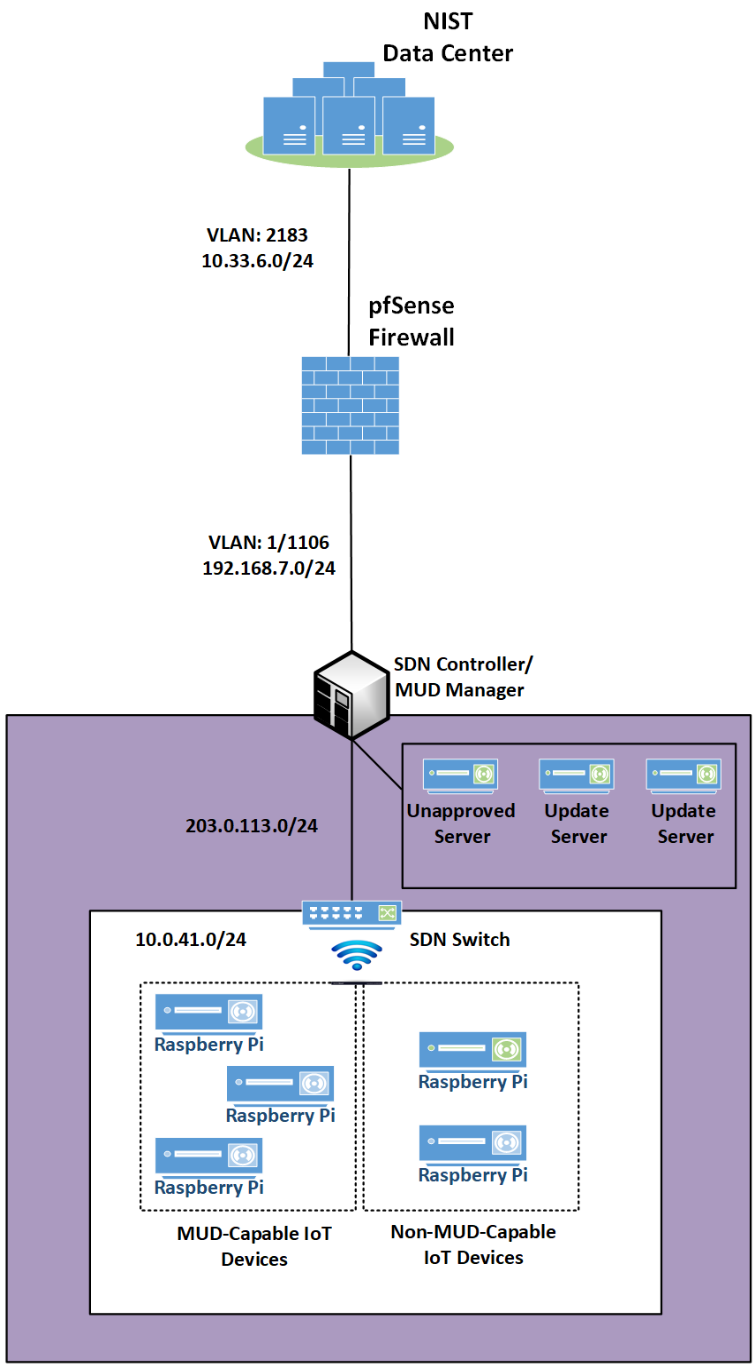
- application configuration—This allows the network administrator to define parameters for the application. The SDN controller/MUD manager must be provided with configuration information for the home and small-business networks that it manages. In addition, configuration parameters for the MUD manager must be supplied.
- controller-class mapping API—This allows the network administrator to define "well-known" network services such as DNS, NTP, and DHCP on the local network and the address prefix used for "local networks."
- device-association—In Build 4, the MUD file URL can be provided to the MUD manager by using the normal DHCP-based MUD URL emission mechanism that is depicted in Figure 9-1. Alternatively, to support devices that are not able to emit a MUD URL, the network administrator can use the REST API to optionally define an association between a device MAC address and a MUD URL.
- MUD file supplied directly—A network administrator can optionally provide a MUD file to the MUD manager by copying it directly into the controller cache in case the manufacturer does not provide a MUD file server.

9.3.2 Physical Architecture

Figure 9-3 depicts the physical architecture of Build 4. A single DHCP server instance is configured for the local network to dynamically assign IPv4 addresses to each IoT device that connects to the SDN switch. This single subnet hosts both MUD-capable and non-MUD-capable IoT devices. The network infrastructure as configured utilizes the IPv4 protocol for communication both internally and to the internet.

The SDN switch is connected across a Wide Area Network (WAN) to the SDN controller/MUD manager. This connection allows the SDN switch to be managed by the SDN controller/MUD manager and enables network flow rules to be updated appropriately. The update servers and unapproved server for Build 4 are also located in this WAN.

3032 **Figure 9-3 Physical Architecture—Build 4**



9.3.3 Message Flow

This section presents the message flows used in Build 4 during several different processes of note.

NIST MUD works by using six flow tables containing flow rules that are applied to each packet in the following order:

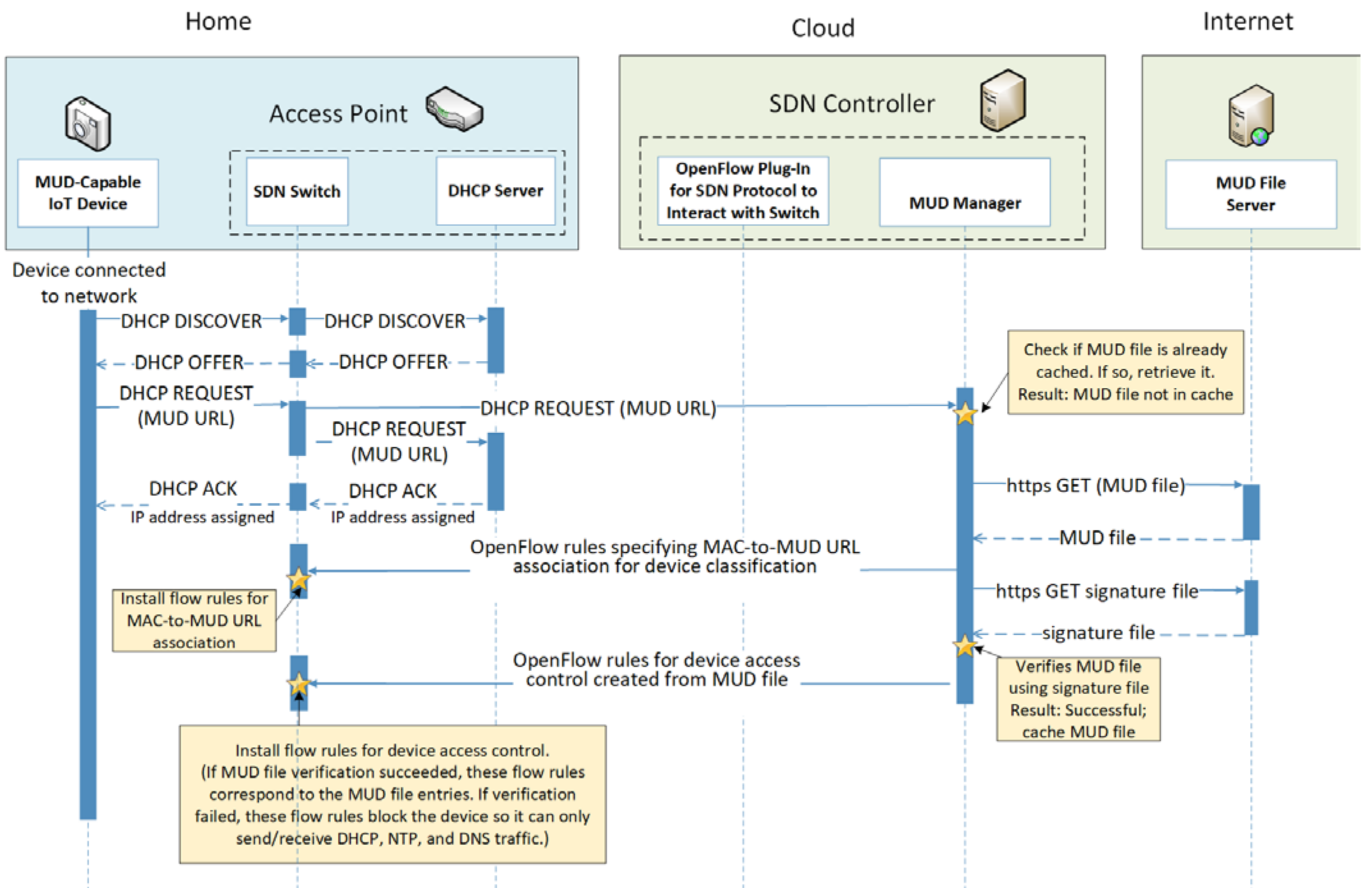
- Table 0, Source MAC address classification table, classifies a packet based on its source IP/MAC address.
- Table 1, Destination MAC address classification table, classifies a packet based on its destination IP/MAC address.
- Table 2, From-Device flow rules table, associates ACEs with the packet based on the packet's source classification if such ACEs exist. ACEs in this table correspond to the From-Device policy in the MUD file. The MUD-specific ACEs that are applied in this table are matched to the packet based on metadata assigned in the first two tables.
- Table 3, To-Device flow rules table, associates ACEs with the packet based on the packet's destination classification if such ACEs exist. ACEs in this table correspond to the To-Device policies in the MUD file. The MUD-specific ACEs that are applied in this table are matched to the packet based on metadata assigned in the first two tables.
- Table 4, Pass-Through table—If a packet has an ACE associated with it (i.e., if it has had a MUD-specific ACE applied to it by table 2 or by table 3 that indicates that it should be permitted), it will be sent to this table and the SDN switch will forward it. (For device-to-device communication based on the manufacturer, model, or local network constructs, there must be both a From-Device rule (in table 2) and a To-Device rule (in table 3) for the communication to be allowed. Otherwise the packet is dropped.)
- Table 5, Drop table—All packets from MUD-enabled devices are by default sent to the Drop table unless there is a MUD rule (and therefore a MUD-specific ACE) that applies to the packet indicating that the packet should be permitted (in which case the packet would have been sent to the Pass-Through table). Unprotected devices are metadata-associated with the reserved MUD URL "UNCLASSIFIED," which allows all packets to and from these devices to be permitted (i.e., there are rules in tables 2 and 3 that permit all traffic to these unprotected devices).

Note that a packet may have just one classification based on source and destination MAC/IP address. Packets originating from devices with assigned MUD URLs are not considered to be part of the local network. Hosts with controller classifications (including those with "well-known" controller classifications such as DHCP, DNS, and NTP servers) are not considered to be part of the local network.

9.3.3.1 Installing MUD-Based Access Control Rules for MUD-Capable Devices

Figure 9-4 shows the message flow that occurs when a MUD-capable device connects to the home/small-business network in Build 4.

3068 **Figure 9-4 MUD-Based Flow Rules Installation Message Flow—Build 4**



3069 As shown in Figure 9-4, the message flow is as follows:

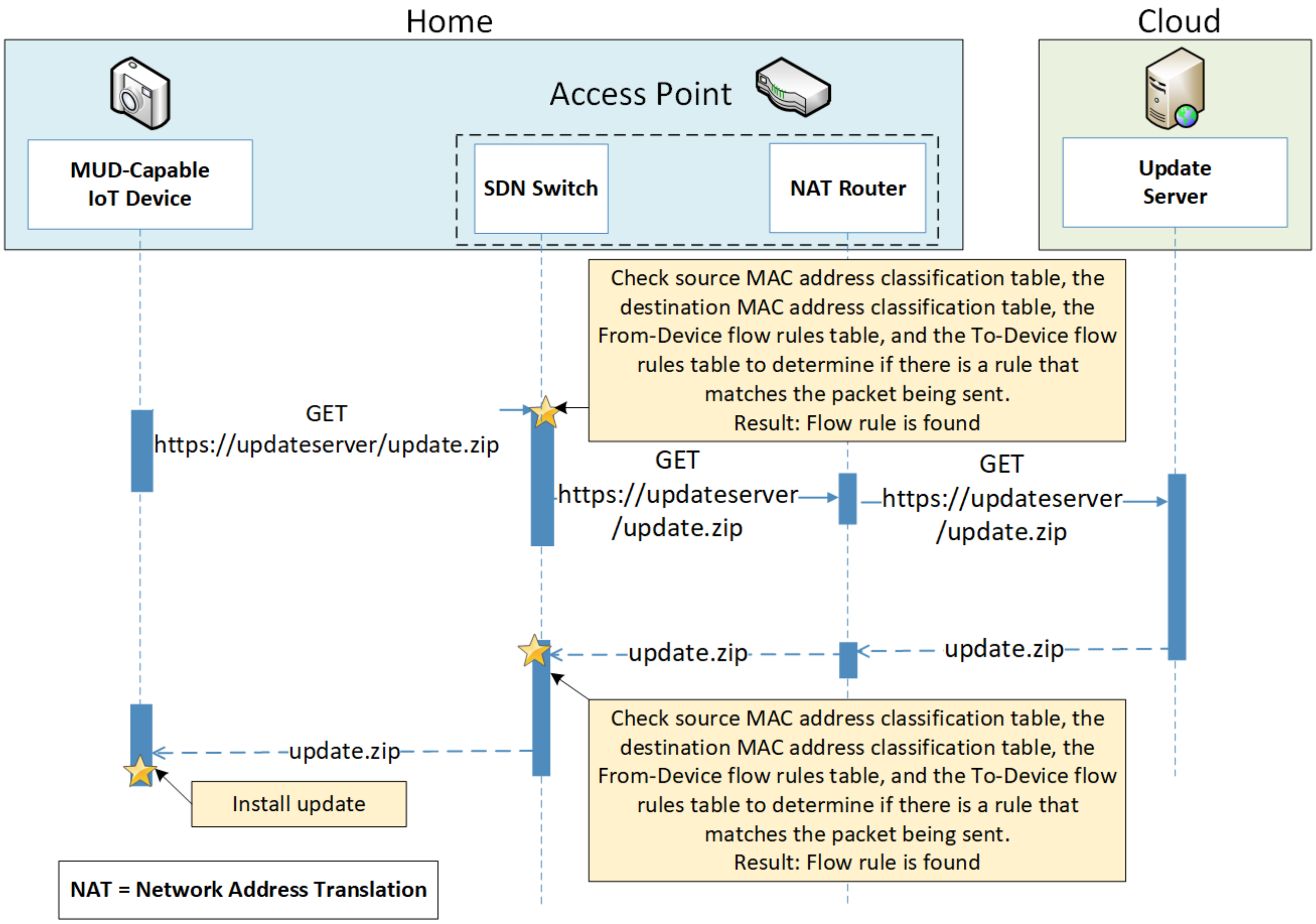
3070 - The IoT device sends out a DHCP DISCOVER message to the SDN switch.
3071 - The AP resident DHCP server sends back a DHCP offer that gets sent back to the device via the
3072 SDN switch.
3073 - The device then sends out a DHCP request containing the MUD URL, which gets sent
3074 simultaneously to the AP resident DHCP server by the SDN switch and to the MUD manager.
3075 - The AP resident DHCP server sends an IP address to the device in a DHCP ACK message via the
3076 switch.
3077 - Based on the MUD URL presented in the DHCP request, the MUD manager checks to see if the
3078 corresponding MUD file is already cached. In the example depicted, the MUD file is not in the
3079 cache.
3080 - The MUD manager retrieves the MUD file from the manufacturer server.

- The MUD manager installs packet classification flow rules into flow tables 0 and 1 (see Section 9.3.3.4) on the SDN switch. These classification rules associate the MAC address of the device interface with the MUD URL. Other classification information such as whether the packet belongs to the local network is also assigned in the first two tables. Table 0 is for source classification and table 1 is for destination classification. If the device had previously sent out packets, i.e., before it was associated with a MUD file, they would have been classified as UNCLASSIFIED in tables 0 and 1. Hence, the entries in tables 0 and 1 that correspond to the device must be cleared at this point and repopulated so subsequent packets are associated with the MUD URL.
- The MUD manager installs the MUD file ACEs as a set of flow rules in tables 2 and 3 (see Section 9.3.3.4).

9.3.3.2 Updates

After a device has been permitted to connect to the home/small-business network, it should periodically check for updates. The message flow for updating the IoT device is shown in Figure 9-5.

Figure 9-5 Update Process Message Flow—Build 4



As shown in Figure 9-5, the message flow is as follows:

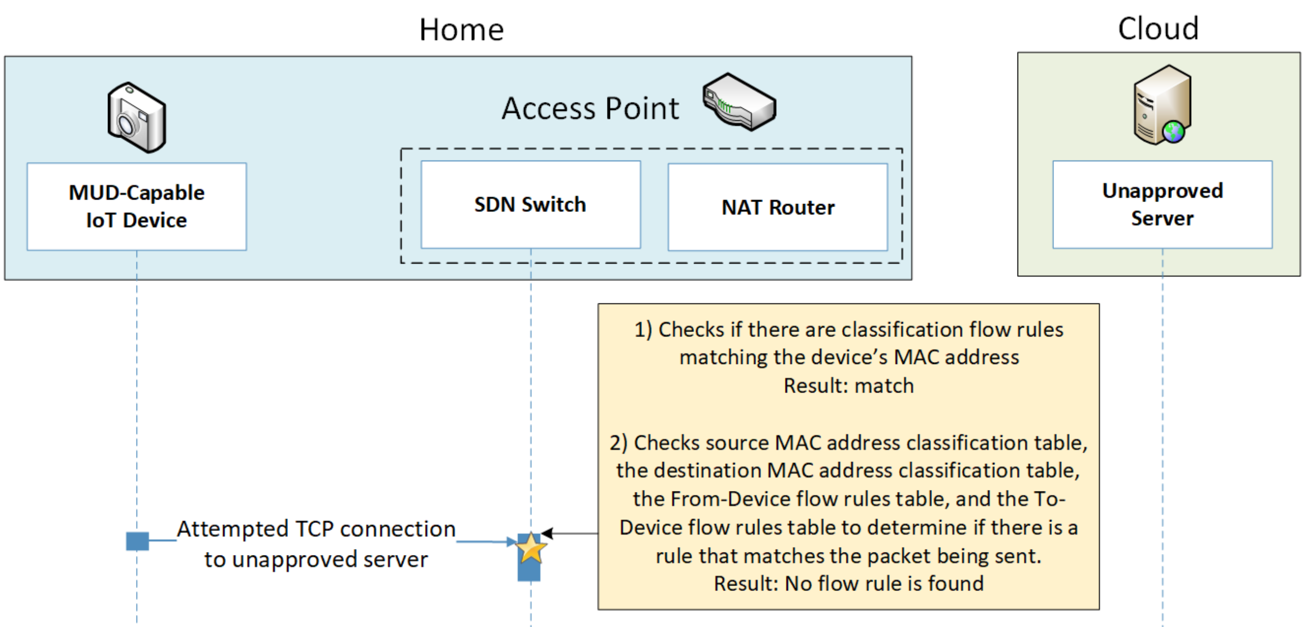
- The device generates an https GET request to its update server.
- The SDN switch will consult its flow rules for this device to verify that it is permitted to send traffic to the update server. Assuming there were explicit rules in the device's MUD file enabling it to send messages to this update server, the SDN switch will forward the request to the NAT router, which will then forward it to the update server.
- The update server will respond with a zip file containing the updates.
- The return traffic will be sent via the NAT router to the switch.
- The destination MAC address of the packet identifies the device, and appropriate metadata is assigned in table 1.
- The source MAC and IP are UNCLASSIFIED, and appropriate metadata is assigned in table 0.

- The packet is forwarded through table 2 and finds a matching flow rule in table 3 from where it is forwarded to the Pass-Through table (4). Two-way communication is thus established.
- The SDN switch will forward this zip file to the device for installation.

9.3.3.3 Prohibited Traffic

Figure 9-6 shows the message flow that occurs when an IoT device attempts to send traffic that is not permitted by its MUD file.

Figure 9-6 Unapproved Communications Message Flow—Build 4



As shown in Figure 9-6, the message flow is as follows:

- A TCP packet is originated from the IoT device with a source MAC address of the device's switch-facing interface and a destination MAC address that is set to the AP-resident router's switch-facing interface. The source IP address is set to the device IP address and destination IP address is set to the unapproved server IP address.
- The packet arrives at the SDN switch, at which point it:
    - enters flow tables 0 and 1, where it is classified and receives the following metadata assignment as a result:
        - <<source-manufacturer, source-model, is-local> <dest-manufacturer, dest-model, is-local>> is assigned in tables 0 and 1

The <source-manufacturer, source-model> are obtained from the MUD URL assigned to the packet. The is-local flag will be set to False because devices with MUD URLs assigned are not considered to be part of the local network.

The destination manufacturer and model assignments will be UNCLASSIFIED, UNCLASSIFIED and is-local is false because the router MAC address is UNCLASSIFIED, and the destination IP address is not part of the local network. Thus, the metadata assignment after table 0 and 1 are traversed will be

<<source-manufacturer,source-model,False><UNCLASSIFIED,UNCLASSIFIED,False>>

  - enters flow table 2, where source metadata-based flow rules have been previously inserted
    - If there is a flow rule that allows the communication, the packet is sent to table 4 (the Pass-Through table), which allows the communication. In the example scenario that is depicted in Figure 9-6, there is no flow rule in table 3 that allows the communications.
    - However, there is a flow rule in table 2 that matches the <source-manufacturer, source-model> that sends the packet to the Drop table (table 5).

- In the example scenario depicted, there is no flow rule found that matches the packet that the IoT device is attempting to send. Therefore, the SDN switch sends the packet to table 5 where there is a single rule that drops the packet.

9.3.3.4 Installation of Timed-Out Flow Rules and Eventual Consistency

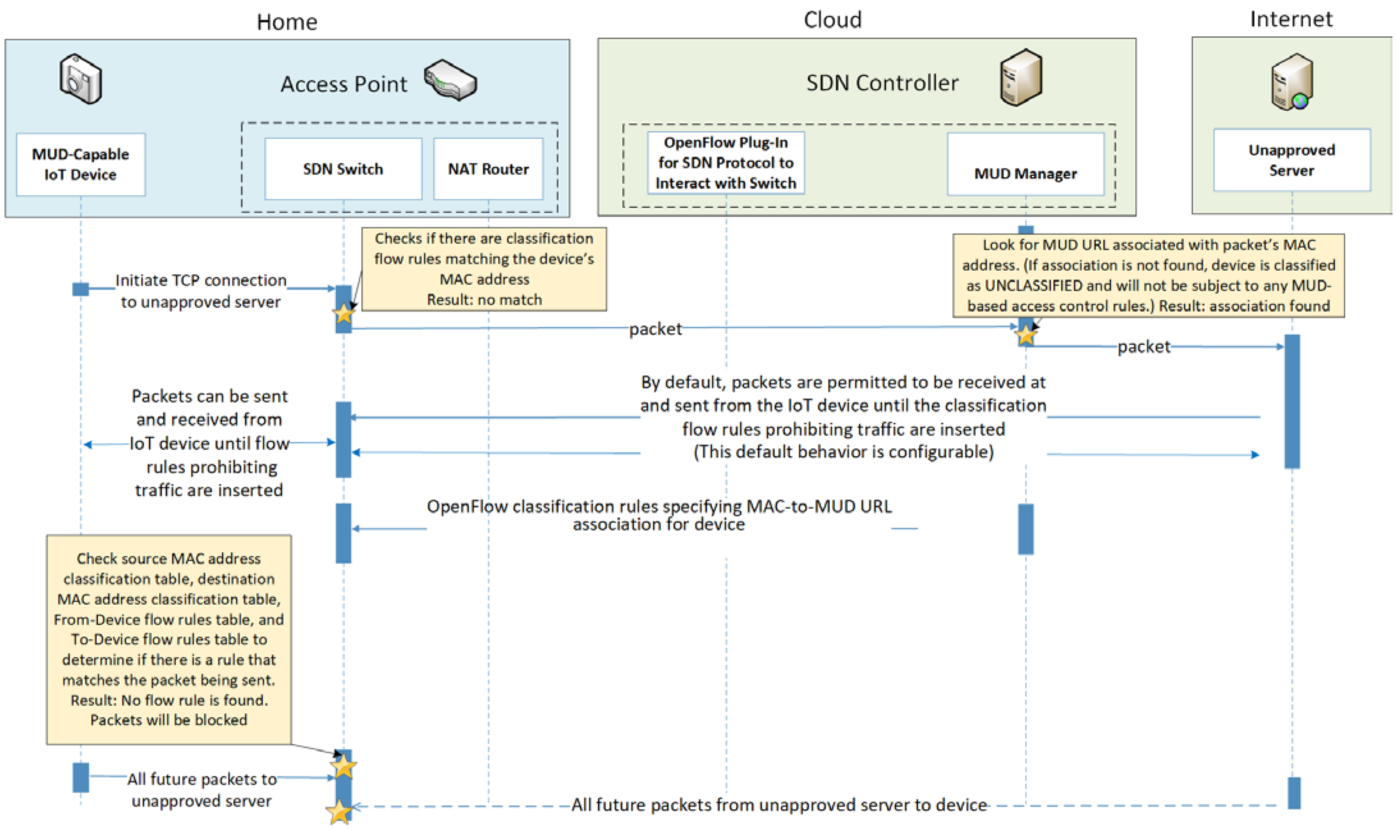
Insertion of flow rules onto the SDN switch on the home/small-business network is dynamic. Rules are computed at the SDN controller/MUD manager and installed on the SDN switch. Flow rules are configured to time out on inactivity to avoid having the SDN switch's flow table fill up. (If an IoT device disconnects from the home/small-business network, there is no need to continue to maintain flow rules for that device on the switch. However, if a device's IP address lease times out, the DHCP server, which has not been modified at all, will not alert the SDN controller/MUD manager of this event. Thus, having the rules time out is an alternative to ensure that rules for disconnected devices will eventually be removed from the switch.)

If an IoT device tries to send a packet, if a packet intended for that device is received at the switch and the source or destination MAC address of the packet does not yet have classification flow rules on the switch, or if the classification flow rules for one or both of those MAC addresses have timed out, the flow rules will need to be sent from the SDN controller/MUD manager to the switch. In this situation, the default OpenFlow rule at the switch (which is inserted in tables 0 and 1 when the switch connects) sends the packet to the MUD manager, and consequently a packet-in event encapsulating the packet is generated at the MUD manager. The packet classification flow rules are then computed and pushed to the switch by the MUD manager during processing of the packet-in event. During this period, additional packets may arrive at the switch.

A design decision had to be made regarding whether to permit the IoT device to send and receive traffic during the window of time while its flow rules are being computed and pushed to the switch. The decision was made to allow an “eventually consistent” model. That is, packets sent by or intended for the IoT device are permitted to proceed through the switch while the SDN flow rules for packet classification are being computed at the SDN controller/MUD manager and sent to the switch. This may result in a few packets that are prohibited by the MUD file ACEs getting through before such violating flows are eventually blocked. This can happen the first time a device sends a packet and every time the flow rules time out due to inactivity. Thus, a misbehaving device or an attacker can have small windows of time during which packets that the MUD file intends to prohibit will be permitted to be exchanged with the device. The alternative is to block the packets while flow rules are computed and inserted. While this alternative behavior can be configured in NIST-MUD, it is not a recommended configuration because it blocks the processing pipeline (resulting in packet drops) while the flow rules are being computed and pushed.

Figure 9-7 shows the message flow that occurs when a device whose flow rules have timed out attempts to initiate communications with an unapproved external server, i.e., a server that is not explicitly listed as a permissible destination in the device’s MUD file.

Figure 9-7 Installation of Timed-Out Flow Rules and Eventual Consistency Message Flow—Build 4



As shown in Figure 9-7, the message flow is as follows:

- The MUD-capable IoT device sends a packet attempting to initiate a TCP connection to an unapproved server.
- The SDN switch checks to see if it has packet classification flow rules for this device (which it determines by looking for rules that match the device's MAC address in tables 0 and 1). In this case, no flow rules are found for this device.
- The SDN switch sends the packet to the SDN controller/MUD manager as a result of the default rule. This is delivered in a packet-in event at the MUD manager.
- The MUD manager receives the packet-in event and looks to see if there is a MUD URL associated with the device's MAC address. (If the device does not have an associated MUD file, it will not be subject to any MUD-based access control rules and will be assigned a reserved MUD URL of UNCLASSIFIED.) In the example scenario depicted in Figure 9-7, the device was found to be associated with a MUD file.

- Even though the flow rules corresponding to the sending device's MUD file are not currently installed on the switch, the SDN controller/MUD manager forwards the packet to the unapproved server.
- The unapproved server responds with an acknowledgment packet.
- The IoT device and the unapproved server are permitted to exchange packets for the time being.
- Meanwhile, the MUD manager computes the SDN flow rules that correspond to the device's MUD file and installs them on the SDN switch.
- After the flow rules have been installed on the switch, when the IoT device attempts to send a packet to the unapproved server, the switch will check each of its flow tables in order (i.e., it will check the Source MAC address classification table [table 0], Destination MAC address classification table [table 1], From-Device flow rules table [table 2], and To-Device flow rules table [table 3]) to determine if there is an ACE that matches the packet being sent. In the example scenario depicted, the switch will find packet classification flow rules for the device in tables 0 and 1, but it will not find any matching flow rules in table 2, indicating that the IoT device's MUD file did not contain an ACE that permits the packet to be sent. As a result, the switch will drop the packet.
- In addition, any subsequent packets that may be sent by the unapproved server and received at the SDN switch will be similarly blocked as a result of the switch consulting its flow rules and determining that there are no ACEs that permit the unapproved server to send packets to the IoT device.

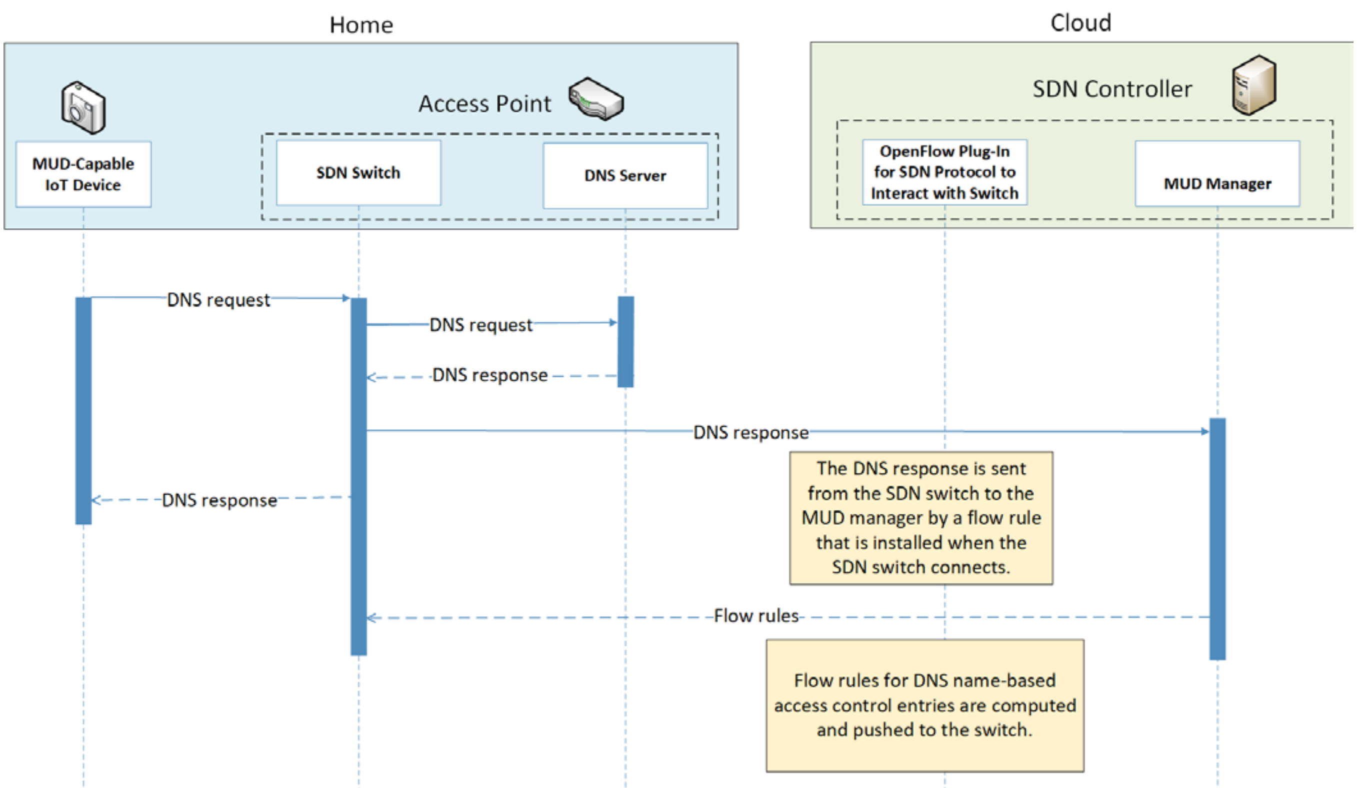
9.3.3.5 DNS Events

MUD allows traffic flow rules to be based on domain names. However, the corresponding SDN flow rules configured in the SDN switch must be based on IP addresses rather than domain names. The MUD manager needs to resolve each host name that is in a MUD file ACE rule to the same value to which it would be resolved by the MUD-enabled IoT device. NIST-MUD is built on the assumption that the SDN controller/MUD manager, which is assumed to be in the cloud, does not necessarily have access to the same DNS resolver as the home/small-business network. Therefore, the SDN controller/MUD manager cannot simply issue DNS queries to resolve domain names that are in MUD files and populate the SDN switch's flow table with the IP addresses that it receives back because the IP addresses that the SDN controller/MUD manager would receive back may not be the same as those that the IoT device would receive back. Instead, as DNS packets are sent from the IoT devices through the SDN-enabled switch, they are also sent to the SDN controller/MUD manager, enabling the SDN controller/MUD manager to snoop on DNS queries and responses that occur on the home/small-business network. The SDN controller/MUD manager extracts the IP address resolution information from each DNS response and uses that information to populate the flow table with the appropriate IP address for rules in the MUD file.

Each time a domain name is resolved for a device on the home/small-business network, the MUD manager must check to determine if there are any flow rules that use that domain name that had previously been deferred (i.e., that have not yet been instantiated and sent to the switch) because the IP address corresponding to that domain name had not yet been known. If so, the MUD manager must instantiate those flow rules by inserting the IP address that corresponds to that domain name in place of that domain name and sending the flow rules to the SDN switch.

Figure 9-8 shows the message flow that occurs when the MUD-capable device does a DNS name lookup and the SDN controller/MUD manager uses the IP address returned in the DNS response to instantiate deferred flow rules for installation on the SDN switch.

Figure 9-8 DNS Event Message Flow—Build 4



As shown in Figure 9-8, the message flow is as follows:

- The IoT device (or any device on the network managed by the switch) does a name lookup by sending a DNS request to the SDN switch, which has a default rule that allows access to DNS.

- The SDN switch forwards the DNS request to a DNS server. In our experiment, this DNS server is resident on the access point.
- The DNS server sends a DNS response back to the SDN switch. The response contains a domain name resolution. Note that if the access point were configured to use an upstream DNS server, the response would be returned from that server and routed back to the device via the switch. For simplicity and control of our experimental setup, we use the AP-resident DNS server so there is no routing of DNS request and response.
- The SDN switch sends the DNS response to the MUD manager, which caches the name resolution information for the switch and updates any DNS-name-based ACEs for MUD files that it manages.
- Concurrently with the previous step, the SDN switch also sends the DNS response to the device that originally generated the DNS request.
- The MUD manager instantiates flow rules corresponding to these DNS-name-based ACEs by substituting each domain's IP address for its domain name and installing the flow rules into flow tables 2 and 3 on the SDN switch.

9.4 Functional Demonstration

A functional evaluation and a demonstration of Build 4 were conducted that involved evaluation of conformance to the MUD RFC. Build 4 was tested to determine the extent to which it correctly implements basic functionality defined within the MUD RFC.

Table 9-2 summarizes the tests that were performed to evaluate Build 4's MUD-related capabilities. It lists each test identifier, the test's expected and observed outcomes, and the applicable Cybersecurity Framework Subcategories and NIST SP 800-53 controls for which each test is designed to verify support. The tests that are listed in the table are detailed in a separate supplement for functional demonstration results. Boldface text is used to highlight the gist of the information that is being conveyed.

Table 9-2 Summary of Build 4 MUD-Related Functional Tests

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
IoT-1	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried.	A **MUD-enabled IoT device is configured to emit a MUD URL.** The MUD manager requests the MUD file and signature from the MUD file server, and the MUD file server	Upon connection to the network, the MUD-enabled IoT device has its MUD **PEP router/switch automatically configured according to the MUD file's**	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	NIST SP 800-53 Rev. 4 CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place.	serves the MUD file to the MUD manager. The MUD file explicitly permits traffic to/from some internet services and hosts, and implicitly denies traffic to/from all other internet services. **The MUD manager translates the MUD file information into local network configurations that it installs on the router or switch that is serving as the MUD PEP for the IoT device.**	**route-filtering policies.**	

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	NIST SP 800-53 Rev. 4 CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.			
IoT-2	**PR.AC-7:** Users, devices, and other assets are authenticated (e.g., single-factor, multifactor) commensurate with the risk of the transaction (e.g., individuals' security and privacy risks and other organizational risks). **NIST SP 800-53 Rev. 4** AC-7, AC-8, AC-9, AC-11, AC-12, AC-14, IA-1, IA-2, IA-3, IA-4, IA-5, IA-8, IA-9, IA-10, IA-11	A MUD-enabled IoT device is configured to emit a URL for a MUD file, but the **MUD file server that is hosting that file does not have a valid TLS certificate. Local policy has been configured to ensure that if the MUD file for an IoT device is located on a server with an invalid certificate, the router/switch will be configured to deny all communication to/from the device.**	When the MUD-enabled IoT device is connected to the network, the MUD manager sends locally defined policy to the router/switch that handles whether to allow or block traffic to the MUD-enabled IoT device. Therefore, the **MUD PEP router/switch will be configured to block all traffic to and from the IoT device.**	Pass
IoT-3	**PR.DS-6:** Integrity-checking mechanisms are used to verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-enabled IoT device is configured to emit a URL for a MUD file, but the **certificate that was used to sign the MUD file had already expired at signing. Local policy has been configured to ensure that if the MUD file for a device has a signature that was**	When the MUD-enabled IoT device is connected to the network and the MUD file and signature are fetched, the MUD manager will detect that the MUD file's signature was created by using a certificate that had already expired	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		signed by a certificate that had already expired at the time of signature, the device's MUD PEP router/switch will be configured to deny all communication to/from the device.	at signing. According to local policy, the **MUD PEP will be configured to block all traffic to/from the device.**	
IoT-4	**PR.DS-6:** Integrity-checking mechanisms are used to verify software, firmware, and information integrity. **NIST SP 800-53 Rev. 4** SI-7	A MUD-enabled IoT device is configured to emit a URL for a MUD file, but the **signature of the MUD file is invalid. Local policy has been configured to ensure that if the MUD file for a device is invalid, the router/switch will be configured to deny all communication to/from the IoT device.**	When the MUD-enabled IoT device is connected to the network, the MUD manager sends locally defined policy to the router/switch that handles whether to allow or block traffic to the MUD-enabled IoT device. Therefore, the **MUD PEP router/switch will be configured to block all traffic to and from the IoT device.**	Pass
IoT-5	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.IP-1:** A baseline configuration of information technology/industrial	Test IoT-1 has run successfully, meaning that the MUD PEP router/switch has been configured based on a **MUD file that permits traffic to/from some internet locations and implicitly denies traffic to/from all other internet locations.**	When the MUD-enabled IoT device is connected to the network, its MUD PEP **router/switch will be configured to enforce the route filtering that is described in the device's MUD file** with	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7		respect to traffic being permitted to/from some internet locations, and traffic being implicitly blocked to/from all remaining internet locations.	
IoT-6	**ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place.	Test IoT-1 has run successfully, meaning that the MUD PEP router/switch has been configured based on a **MUD file that permits traffic to/from some lateral hosts and implicitly denies traffic to/from all other lateral hosts.** (The MUD file does not explicitly identify the hosts as lateral hosts; it identifies classes of hosts to/from which traffic should be denied, where one or more hosts of this class happen to be lateral hosts.)	When the MUD-enabled IoT device is connected to the network, its MUD PEP **router/switch will be configured to enforce the access control information that is described in the device's MUD file** with respect to traffic being permitted to/from some lateral hosts, and traffic being implicitly blocked to/from all remaining lateral hosts.	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.PT-3: The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-3:** Assets are formally managed throughout removal, transfers, and disposition.			
IoT-9	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4 **DE.AE-1:** A baseline of network operations and expected data flows for users and systems is established and managed. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CM-2, SI-4 **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24	Test IoT-1 has run successfully, meaning the MUD PEP **router/switch has been configured based on the MUD file** for a specific MUD-capable device in question. The MUD file contains domains that resolve to multiple IP addresses. The MUD PEP router/switch should be configured to permit communication to or from all IP addresses for the domain.	A domain in the MUD file resolves to two different IP addresses. The MUD manager will create firewall rules that permit the MUD-capable device to send traffic to both IP addresses. The MUD-capable device attempts to send traffic to each of the IP addresses, and the MUD PEP router/switch permits the traffic to be sent in both cases.	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	PR.AC-5: Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM- 5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.DS-2:** Data in transit is protected. **NIST SP 800-53 Rev. 4** SC-8, SC-11, SC-12			
IoT-10	**ID.AM-1:** Physical devices and systems within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-2:** Software platforms and applications within the organization are inventoried. **NIST SP 800-53 Rev. 4** CM-8, PM-5 **ID.AM-3:** Organizational communication and data flows are mapped. **NIST SP 800-53 Rev. 4** AC-4, CA-3, CA-9, PL-8 **PR.DS-5:** Protections against data leaks are implemented. **NIST SP 800-53 Rev. 4** AC-4, AC-5, AC-6, PE-19, PS-3, PS-6, SC-7, SC-8, SC-13, SC-31, SI-4	A MUD-capable IoT device is configured to emit a MUD URL. Upon being connected to the network, its MUD file is retrieved, and the PEP is configured to enforce the policies specified in that MUD URL for that device. **Within 24 hours (i.e., within the cache-validity period for that MUD file), the IoT device is reconnected to the network.** After 24 hours have	Upon reconnection of the IoT device to the network, **the MUD manager does not contact the MUD file server. Instead, it uses the cached MUD file.** It translates this MUD file's contents into appropriate route-filtering rules and installs these rules onto the PEP for the IoT device. Upon reconnection of the IoT device to the network, after 24	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	DE.AE-1: A baseline of network operations and expected data flows for users and systems is established and managed. **PR.AC-4:** Access permissions and authorizations are managed, incorporating the principles of least privilege and separation of duties. **NIST SP 800-53 Rev. 4** AC-1, AC-2, AC-3, AC-5, AC-6, AC-14, AC-16, AC-24 **PR.AC-5:** Network integrity is protected, incorporating network segregation where appropriate. **NIST SP 800-53 Rev. 4** AC-4, AC-10, SC-7 **PR.IP-1:** A baseline configuration of information technology/industrial control systems is created and maintained, incorporating security principles (e.g., concept of least functionality). **NIST SP 800-53 Rev. 4** CM-2, CM-3, CM-4, CM-5, CM-6, CM-7, CM-9, SA-10 **PR.IP-3:** Configuration change control processes are in place. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10 **PR.PT-3:** The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. **NIST SP 800-53 Rev. 4** AC-3, CM-7 **PR.DS-2:** Data in transit is protected.	elapsed, the same device is reconnected to the network.	hours have elapsed, the MUD manager does fetch a new MUD file.	

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
IoT-11	**ID.AM-1:** Physical devices and systems within the organization are inventoried.	A **MUD-enabled IoT device can emit a MUD URL.** The device should leverage one of the specified manners for emitting a MUD URL.	Upon initialization, the MUD-enabled IoT device broadcasts a DHCP message on the network, including at most one **MUD URL, in https scheme, within the DHCP transaction OR as an LLDP extension.**	Pass

9.5 Observations

NIST-MUD was able to successfully permit and block traffic to and from MUD-capable IoT devices as specified in the MUD files for the devices.

NIST-MUD does not implement LLDP extensions or certificate-based device authentication. (An authentication server can, however, inform the MUD manager of the MAC to MUD URL association using the API provided by NIST-MUD.) The current implementation supports devices that emit their MUD URL using the MUD DHCP extension or that are associated with their MUD URL by the provided API (i.e., the administrator or network authentication server configures the association).

NIST-MUD does not implement secure conveyance of the device's MUD URL. A device may "lie" about its identity by issuing a spurious DHCP request with a MUD URL embedded. There are no certificate-based checks to verify that the MUD URL that the device emits is in fact that device's MUD URL.

As was discussed in Section 9.3.3.4, a misbehaving device or an attacker can have small windows of time where illegal packets can be exchanged with a device the first time the device sends or receives packets after its flow rules have timed out. This is because the design decision was made to permit packets sent by or intended for the IoT device to proceed through the switch while the SDN flow rules for packet classification are being computed at the SDN controller/MUD manager and pushed to the switch. The alternative is to block the packets while classification rules are inserted. While this can be configured, it is not a recommended configuration because it disrupts correct behavior.

10 General Findings, Security Considerations, and Recommendations

This section introduces findings based on the build implementations and demonstrations, security considerations, and recommendations.

10.1 Findings

Based on our experiences with the various builds considered and demonstrated in this project, we offer the following findings:

- It is possible to achieve significantly better security than is typically achieved in today's (non-MUD-capable) home and small-business networks by deploying and using MUD on those networks to constrain the communications of IoT devices.
- MUD is designed to protect limited-purpose devices whose communication needs can be clearly defined. These communication needs are defined in terms of not only the ports and protocols with which the IoT devices are permitted to communicate but also the destinations with which the IoT devices can communicate. If a device is not a limited-purpose device but instead has very general communication requirements that cannot be clearly defined (e.g., a laptop or a phone), then the device does not lend itself to protection by MUD.
- The demonstrated approach, as implemented in each of the builds, shows that by using MUD-capable IoT devices on networks where support for MUD has been deployed, it is possible to manage access to MUD-capable IoT devices in a manner that maintains device functionality while
    - preventing access to the MUD-capable IoT device from other devices on the internal network that are not from manufacturers or device classes explicitly permitted by the MUD-capable device's MUD file
    - preventing the MUD-capable IoT device from being used to access unauthorized external domains
    - preventing the MUD-capable IoT device from accessing other devices on the internal network that are not from manufacturers or device classes explicitly permitted by the MUD-capable device's MUD file
- MUD can help prevent MUD-capable IoT devices from being used to launch DDoS and other network-based attacks that are typically made possible by commandeering IoT devices found on today's home and small-business networks. For MUD to provide this protection, it must be deployed correctly, networks must use MUD-capable IoT devices, and MUD files must be written and available for these devices so that the files authorize only the outgoing communications that each MUD-capable IoT device needs to maintain its intended functionality.

- There are commercially available network visibility/monitoring technologies that can detect connected devices and identify certain device attributes (e.g., type, IP address, OS) throughout the duration of a device's connection to the network. These "fingerprinting" technologies are also able to detect when the devices leave the network or are powered off and to note their change of status accordingly.
- Setup and configuration of the components needed to deploy MUD on a network (MUD-capable router/switch and MUD manager) should ideally be able to be performed easily, right out of the box, to enable typical home or small-business users to deploy MUD successfully. While Build 2 and Build 3 are plug-and-play solutions designed to be easily deployable, setup and configuration of the other builds are not currently sufficiently user-friendly to enable the typical, nontechnical user to deploy these implementations easily and seamlessly. For MUD to be widely deployed on home/small-business networks, emphasis on ease of use will be crucial.
- MUD has the potential to help with the security of even those IoT devices that have been deprecated and are no longer receiving regular updates. Eventually, most IoT devices will reach a point at which they will no longer be updated by their manufacturer. This is a dangerous point in any device's life cycle because it means that any of its security vulnerabilities that become known after this point will not be protected against, leaving the device open to attack. For MUD-capable devices that reach this end-of-life stage, however, the use of MUD provides additional protection that is not available to non-MUD-capable devices. Even if a MUD-capable device can no longer be updated, its MUD file will still limit the other devices with which that MUD-capable device is able to communicate, thereby limiting what other devices could be used to attack it and what other devices it could be used to attack. In the future, there are expected to be many IoT devices that are no longer being updated by their manufacturers but will continue to be used. The ability to leverage MUD to limit the communication profiles of such unsupported devices will be important for protecting these highly vulnerable devices from attack by unauthorized endpoints and for protecting the internet from attack by these vulnerable devices.
- Even when using components that are fully conformant to the MUD specification, there are still some behaviors that will be determined by local policy. If the default policy that is provided by a specific product out of the box is not sufficient, user action will be required to configure the device according to a different and desired policy. User-friendly interfaces will be needed to enable the typical, nontechnical user of a home or small-business network to interact with the MUD components to modify their default settings when needed. For example, the MUD specification does not dictate what action to take (e.g., block or permit traffic to the IoT device) if the MUD manager is not able to validate the device's MUD file server's TLS certificate or if the MUD manager is not able to validate the device's MUD file signature. In either of these cases, if the default behavior that the device is configured to perform is not acceptable to the user, the user would need to reconfigure the device to perform the desired behavior. Ideally the device would provide a user-friendly interface through which to do so.
- In the absence of mechanisms that enable users to configure the specific local policy that is enforced when encountering certain error situations, MUD manager implementers may want to

give additional thought to the local policies that the MUD manager enforces by default. There is a trade-off to be made between security and availability. Enforcing default local policies that are nuanced may enable an implementation to achieve a more desirable balance between security and availability in some situations. For example, the MUD RFC does not specify what behavior an implementation should exhibit when errors are experienced during retrieval or validation of a device's MUD file. A MUD file server could be found to have an invalid TLS certificate, which is highly suspicious and therefore concerning; or it could be found to have an otherwise valid TLS certificate that has simply expired, which may be less concerning. Similarly, the MUD file itself could be found to have an invalid signature (concerning) or a signature that is otherwise valid but whose associated certificate had expired at the time it was used to sign the MUD file (perhaps less concerning). Given the absence of guidance in the RFC regarding how an implementation should behave in such situations, the implementation is expected to behave according to local policy.

The implementation can fail closed, as do Builds 1 and 4, meaning that the device will not be permitted to send or receive any traffic. While such a policy is extremely secure, it also renders the devices unreachable and effectively useless. Alternatively, the implementation can fail open, as it does in the case of Builds 2 and 3, meaning that the device is permitted to communicate freely, as if it does not have an associated MUD file. Builds 2 and 3 enable MUD-capable devices that have invalid MUD files or that have MUD file servers with invalid TLS certificates to connect to the network and communicate without being subject to any MUD-related traffic constraints. While this behavior is not erroneous, some users may be surprised to learn that a device that purports to be MUD-capable may not actually be subject to any of the rules in its MUD file in these situations.

There is merit in the argument that devices should be able to communicate unconstrained (rather than not being able to communicate at all) when their MUD file or MUD file server certificates are otherwise valid but have expired. However, it is more difficult to make the case that these devices should be able to be communicate unconstrained if their MUD file signature or MUD file server certificate has not expired but is invalid. It may be desirable, therefore, to consider implementing a default local policy that determines whether to fail open or fail closed depending on the reason that the MUD file signature or MUD file server certificate cannot be validated. Alternatively, an implementer may want to take advantage of unique product features in its response to error situations such as these and consider classifying devices as being in a specific category (in the case of Build 2) or placing devices in a specific micronet (in the case of Build 3) that results in the devices being subjected to appropriate communication constraints. An implementer utilizing Easy Connect onboarding could even prevent a wireless device from being provisioned with network credentials if the MUD manager were not able to validate the device's MUD file.

- The MUD specification (RFC 8520) states that the mud-signature element in the MUD file is optional, but it does not specify what the behavior of the MUD manager should be in the event that the mud-signature element is not present in a MUD file. MUD manager implementers should give careful thought to the behavior that their MUD manager implementations enforce by default. They should make this behavior clear so that users who are deploying MUD on their networks understand whether their MUD manager will automatically process a MUD file that does not have a mud-signature element or whether it will cease processing such a MUD file and wait for administrator input. MUD manager implementers should also make it possible for users to configure this MUD manager behavior as needed by local policy. A MUD manager that automatically processes MUD files that do not include a mud-signature element is vulnerable to accepting and processing as valid MUD files that have been modified by attackers if those attackers have deleted the mud-signature element from the MUD file.

- There is still a dearth of MUD-capable IoT devices. Users wanting to deploy MUD do not yet have the option to do so because of a lack of availability of MUD-capable IoT devices. More vendor buy-in is required to encourage IoT device manufacturers to implement support for MUD in their devices.
- To encourage further adoption of MUD, early adopters should tell their organizational story of change: who in the organization is responsible for understanding what goes into the MUD file, building the MUD file, making the MUD file available on a server, modifying the device to emit a URL, testing MUD-related features, and determining if a MUD file needs to be updated, among other functions.
- Communications between the MUD manager and the router/switch, between the threat-signaling server and the MUD manager/router, and between the IoT devices and their corresponding update servers are not standardized. This lack of standardization has the potential to inhibit interoperability of components that are obtained from different manufacturers, thereby limiting the choice that consumers have to mix architectural components from different vendors in their MUD deployments.
- RFC 8520 states clearly that if the cache-validity timer has not expired, the MUD manager must not check for a new MUD file and should use the cached file instead. It also clearly states that expiration of the cache-validity timer does not require the MUD manager to discard the MUD file. It does not, however, state that if the cache-validity timer has expired, the MUD manager should check for a new MUD file, even though this is the behavior that the RFC authors had intended to specify. It is our understanding that this will be submitted as an erratum for clarification. In the meantime, implementations wishing to conform to the desired behavior should be designed such that if the cache-validity timer has expired, the MUD manager checks for a new MUD file.
- MUD rules are defined in terms of domain names, but when MUD rules are instantiated on routers, IP addresses, rather than domain names, are used. However, the IP address to which any given domain resolves may change. So, if a domain is listed in a MUD file rule and device traffic filters that instantiate this MUD file rule have been installed on the router, when the domain begins resolving to a different address, the device will initially not behave as intended. If the device attempts to communicate with this new IP address, it will not be permitted to do so because there will not yet be device traffic filters in its router that permit it to access this new IP address. The device traffic filters in the router will still be permitting access to the old IP

address. In other words, the device will not be permitted to communicate with the desired domain, despite this communication being permitted by the device's MUD file. This undesirable situation will persist until the device traffic filters in the router are updated to use the new IP address to which the domain now resolves.

To minimize the effect of such a situation, the MUD implementation (e.g., the MUD manager) should periodically generate DNS resolution requests for each of the domains listed in the MUD file and, if any of these domains now resolve to different IP addresses than previously, the device traffic filters using the old IP address should be deleted from the router or switch, and the device traffic filters using the new IP address should be installed. Regarding how often a MUD implementation might want to perform this periodic checking of domain name resolution values, one suggestion is to do so at intervals of TTL+V, where TTL is the time to live value in the A record of the domain's DNS entry, and V might be as long as 86,400 seconds (i.e., 24 hours). (The TTL value specifies how long a resolver is supposed to cache the DNS query before the query expires and the domain should be resolved again. If a DNS record for a domain changes, a new lookup will not be done until the cache expires.) Users should be cautioned that if the IP address to which a domain name resolves changes, the IoT device may be prohibited from communicating with that domain for some period (i.e., V) after the TTL for the domain's DNS entry has expired.

- When a MUD-capable IoT device performs a domain name lookup, it is important that the IP addresses to which the domain name gets resolved match the IP addresses that that domain name got resolved to when the MUD rule containing that domain was installed at the router or switch. If they do not match, then the device could be prohibited from communicating with the desired domain despite the existence of a MUD rule explicitly permitting the device to do so.

  If the router or switch itself does a domain name lookup when the MUD rule is installed on it, and if the device and the router or switch are co-located, then the device and the router or switch will be in the same region and would be expected to have their domain name lookups resolved to the same IP addresses. Therefore, if the router or switch itself performs the domain name lookup when translating a MUD rule to device traffic filters, the IP address(es) that are returned to the IoT device when it performs a domain name lookup should be the same as the IP address(es) that were configured in the device traffic filters.

  However, if some other component, such as a MUD manager or controller that is in the cloud, performs a domain name lookup and sends the resulting device traffic filters to the router or switch for installation, then it is possible that the controller/MUD manager and the router or switch could be in a different region, which could mean that their domain name lookups for a given domain do not resolve to the same IP addresses. For MUD rules to be enforced as expected, measures need to be taken to ensure that the IP addresses that are used in the device traffic filters match the IP addresses that the IoT device would in fact use. Some possible ways of ensuring address alignment include:

  - requiring that the IoT device and the entity that is instantiating the MUD rules as device traffic filters use the same DNS server

    - having the entity that is instantiating the MUD rules as device traffic filters eavesdrop on the DNS queries made by the IoT device so it can learn what IP addresses the IoT device receives back in the DNS responses
    - having the router or switch occasionally send DNS queries for the list of domains it used in MUD files and updating the device traffic filters based on those queries

- In working with project collaborators, the NCCoE determined that MUD is only one of several foundational elements that are important to IoT security. First and foremost, it is imperative that IoT device manufacturers follow best practices for security when designing, building, and supporting their devices. Manufacturers should, for example, understand and manage the security and privacy risks posed by their devices as discussed in NISTIR 8228, *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks,* as well as the more general guidelines for identifying, assessing, and managing security risks that are discussed in the *Framework for Improving Critical Infrastructure Cybersecurity* (Cybersecurity Framework). In addition, they should continue to support their devices throughout their full life cycle, from initial availability through eventual decommissioning, with regular patches and updates. Cisco has proposed the following four elements as necessary for IoT security:
    - device security by design: certifiable device capabilities
    - device intent: MUD
    - device network onboarding: secure, scalable, automated—bootstrapping remote secure key infrastructure/autonomic networking integrated model approach
    - life-cycle management: behavior, software patches/updates

  All four builds in this project support the second security element listed above (device intent: MUD). Build 3 also supports the third security element (secure, scalable, and automated onboarding of devices to the network) through use of the Wi-Fi Easy Connect protocol.

- When devices are onboarded using the Wi-Fi Easy Connect R1 protocol (as in Build 3), network security is enhanced because:
    - Each IoT device is assigned unique network credentials, which ensures that
        - even if the credentials of one device are known, these credentials cannot be presented by other devices (e.g., devices that are not authorized to connect to the network) to gain access to the network.
        - Credentials of some devices may be revoked or changed without interfering with the ability of other devices to connect to the network.
    - Network credentials are provisioned to each device via an automated protocol, thereby minimizing the opportunity for human error and exposure.
    - Network credentials are provisioned to each device over a secure channel, minimizing the possibility of their disclosure because

  - The credentials are never displayed to the user, so presentation of the device's network credentials to the network does not pose any risk that the credentials will be viewed and thereby disclosed.
  - No human being has an opportunity to be privy to the credentials of any device.

- While the Wi-Fi Easy Connect protocol onboarding performed in Build 3 (R1) is largely automated, it does require an individual to perform the manual operations of putting the IoT device into onboarding mode (assuming the device does not come out of the box ready to onboard) and scanning the device's QR code. Use of the Wi-Fi Easy Connect protocol relies on trust that the individual who scans the QR code will select the correct network to which to onboard the device. An individual who onboards a device to a network other than the device's intended network effectively executes a takeover attack on that IoT device, thereby preventing the device's intended network from taking control of the device. Such a takeover attack could be executed, in theory, by a rogue individual by:
  - positioning an alternative network within Wi-Fi range of the device
  - obtaining access to the device's QR code
  - putting the device into onboarding mode (or waiting until someone else puts the device into onboarding mode) and onboarding the device to the alternative network before the device is onboarded to its intended network

  By onboarding a device to a network other than its intended network, the owner of the alternative network can take control of the device, thereby denying the owner of the device the ability to use it on its intended network. Even more maliciously, such an attack could allow the owner of the alternative network to access and tamper with the device before eventually allowing it to be onboarded to the intended network, thus enabling a compromised device to be onboarded to the intended network.

- There are numerous ways in which support for MUD can be provided within a home/small-business network. Build 3 demonstrates support for MUD in residential gateway equipment and service provider infrastructure. However, this does not imply any requirement that service providers bear the responsibility for implementing MUD. Builds 1, 2, and 4 simply require that customers acquire and use third-party routers and other related components that are MUD-capable. Integrating MUD capability into residential gateway equipment supplied by service providers, along with strong advocacy and education of customers to explain the benefits of using MUD, represents one approach to encouraging widespread adoption of MUD in home and small-business environments. Factors affecting determination of how and where MUD should be supported include infrastructure and support requirements, cost, and privacy. These are some issues that should be considered:
  - Upgrading all existing internet gateways to be MUD-capable would be a large undertaking, so service providers might perform cost-benefit analyses to determine whether it makes economic sense for them to provide and support MUD-capable internet gateways in homes and small businesses.

- Providing and supporting MUD-capable internet gateways could potentially cast service providers into a situation in which they might be perceived as responsible for troubleshooting problems with the IoT devices themselves. This is a function that is generally outside the service provider's control.
- In addition to upgrading internet gateways to be MUD capable, service providers might choose to make changes to other aspects of the service provider network to support MUD. A service provider's analysis regarding whether it should integrate support for MUD into the residential gateway or simply encourage its customers to use MUD-capable third-party routers should consider any additional network changes that may be needed.
- The MUD manager, by its very nature, is aware of all MUD-capable IoT devices that are attached to the network and of what domains and other types of local devices they are permitted to communicate with. Such information could have privacy ramifications. Whatever organizational controls the MUD manager will have access to this information. If this organization is a service provider, as in the Build 3 implementation, the service provider will be privy to this personal information.

10.2 Security Considerations

Use of MUD, when implemented correctly, allows manufacturers to constrain communications to and from IoT devices to only those sources and destinations intended by the device's manufacturer. By restricting an IoT device's communications to only those that it needs to fulfill its intended function, MUD reduces both the communication vectors that can be used to attack a vulnerable IoT device and the communication vectors that a compromised IoT device can use to attack other devices. MUD does not, however, provide any inherent security protections to IoT devices themselves. If a device's MUD file permits an IoT device to receive communications from a malicious domain, traffic from that domain can be used to attack the IoT device. Similarly, if the MUD file permits an IoT device to send communications to other domains, and if the IoT device is compromised, it can be used to attack those other domains. Users deploying MUD are advised to keep the following security considerations in mind.

- It is important to ensure that the MUD implementation itself is secure and not vulnerable to attack. If the MUD implementation itself were to be compromised, the compromised MUD infrastructure would serve as a venue for attack. As stated in the Security Considerations section of the MUD specification (RFC 8520), "The basic purpose of MUD is to configure access, so by its very nature, it can be disruptive if used by unauthorized parties." Protecting the MUD infrastructure includes ensuring the integrity and security of the MUD file location (e.g., the IoT device MUD URL emission), the MUD manager, the DHCP server (when used for MUD URL emission), the MUD file server, the router, and the private key used to sign the MUD file. If the MUD implementation itself is compromised—e.g., if an IoT device emits an incorrect MUD file URL; if a different MUD file URL is sent to the MUD manager than that provided by the IoT device; if a well-formed, signed MUD file is malicious; if a malicious actor creates a compromised MUD manager; or if a router is compromised so that it does not enforce its device traffic filters—then MUD can be used to enable rather than prevent potentially damaging communications between affected IoT devices and other domains.

- If a malicious actor can create a well-formed, signed, malicious MUD file, the undesirable communications that will be permitted by that MUD file will be readily visible by reading the MUD file. Therefore, for added protection, users implementing MUD should review the MUD files for their IoT devices to ensure that they specify communications that are appropriate for each device. Unfortunately, on home and small-business networks, where users are not likely to have the technical expertise to understand how to read MUD files, users will be required to trust that the MUD files specify communications appropriate for the device or to rely on a third party to perform this review for them.
- MUD implementation depends on the existence and secure operation of a MUD file server from which a device's MUD file can be retrieved. If the manufacturer goes out of business or does not conform to best common practices for patching, the MUD file server domain would be vulnerable to having malware deployed on it and thereby being transformed into an attack vector. To safeguard against such a scenario, a mechanism needs to be defined to enable the domain of the manufacturer to be invalidated so that the MUD manager can be protected from connecting to the compromised MUD file server, despite the fact that IoT devices may continue to emit the URL of the compromised domain. Use of threat-signaling information is one example of such a mechanism.
- To protect all IoT devices on a network, both MUD-capable and non-MUD-capable, users may want to consider investigating mechanisms for supplying MUD files for legacy (non-MUD-capable) devices.
- By emitting or otherwise conveying a MUD URL, a device reveals information about itself, thereby potentially providing an attacker with guidance on what vulnerabilities it might have and how it might be attacked.
- An attacker could spy on the MUD manager to determine what devices are connected to the network and then use this information to plan an attack.
- If an attacker can gain access to the local network, they may be able to use the MUD manager in a reflected denial of service attack by emitting a large amount of MUD URLs (e.g., from spoofed MAC addresses) and forcing the MUD manager to make connection attempts to retrieve files from those MUD URLs. Safeguards to counter this, such as throttling connection attempts of the MUD manager, should be considered.
- MUD users should understand that the main benefit of MUD is its ability to limit an IoT device's communication profile; it does not necessarily permit owners to find, identify, and correct already-compromised IoT devices.
    - If a system is compromised but it is still emitting the correct MUD URL, MUD can detect and stop any unauthorized communications that the device attempts. Such attempts may also indicate potential compromises.

- On the other hand, a system could be compromised so that it emits a new URL referencing a MUD file that a malicious actor has created to allow the compromised device to engage in communications that should be prohibited. In this case, whether the compromised system will be detected depends on how the MUD manager is configured to react to such a change in MUD URL. According to the MUD specification, if a MUD manager determines that an IoT device is sending a different MUD URL, the MUD manager should not use this new URL without some additional validation, such as a review by a network administrator.
  - If the MUD manager requires an administrator to accept the new URL but the administrator does not accept it, MUD would help owners detect the compromised system and limit the ability of the compromised system to be used in an attack.
  - However, if the MUD manager does not require an administrator to accept the new URL or if it requires an administrator to accept the new URL and the administrator does accept the new URL, MUD would not help owners detect the compromised system, nor would it limit the ability of the compromised system to be used in an attack.
  - As a third possibility, a compromised system could be subjected to a more sophisticated attack that enables it to dynamically change its identity (e.g., its MAC address) along with emitting a new URL. In this case, the compromised system would not be detected unless the MUD manager were configured to require the administrator to explicitly add each new identity to the network.

- The following security considerations are specific to the MUD deployment and configuration process:
  - When an IoT device emits its MUD URL by using DHCP or LLDP rather than using an X.509 certificate that can provide strong authentication of the device or by using some other mechanism that provides a trusted association between the MUD URL and the device, the device may be able to lie about its identity and thereby gain network access it should not have. If a network includes IoT devices that emit their MUD URL by using one of these insecure mechanisms, as do some of the builds implemented in this project, network administrators should take additional precautions to try to improve security. For example, the MUD implementation should be configured to:
    - prevent devices that have not been authenticated from being in the same class as devices that have been strongly authenticated to prevent the non-authenticated devices from getting possibly elevated permissions that are granted to the authenticated devices
    - prevent devices that have not been authenticated from being able to use the same MUD URL as devices that have been strongly authenticated
    - whenever possible, bind communications to the authentication that has been used, e.g., IEEE 802.1X, 802.1AE (MACsec), 802.11i (WPA2), WFA Easy Connect, or future authentication types

  - o remove state if an unauthenticated method of MUD URL emission is being used and any form of break in that session is detected
  - o not include unauthenticated devices into the manufacturer grouping of any specific manufacturer without additional validation
  - o use additional discovery and classification components that may be on the network to try to fingerprint devices that have not been authenticated to try to verify that they are of the type they are asserting to be by their MUD URLs
  - o raise an alert and require administrator approval if the MUD manager detects that the signer of a MUD file has changed, to protect against rogue Certificate Authorities
  - o raise an alert and require administrator approval if the MUD manager detects that a device's MUD file has changed, to protect compromised IoT devices that seek to be associated with malevolent MUD files

- To protect against domain name ownership changes that would permit a malicious actor to provide MUD files for a device, MUD managers should be configured to cache certificates used by the MUD file server. If a new certificate is retrieved, the MUD manager should check to see if ownership of the domain has changed and, if so, it should raise an alert and require administrator approval.

The points above provide only a summary of the security considerations discussed in the MUD specification (RFC 8520). Users deploying a MUD implementation are encouraged to consult that document directly for more detailed discussion.

Additionally, please refer to NISTIR 8228, *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks,* for more details related to IoT cybersecurity and privacy considerations.

10.3 Recommendations

The following are recommendations for using MUD:

- Home and small-business network owners should make clear to vendors that both IoT devices and network components need to be MUD-capable. They should use MUD-capable IoT devices on their networks and enable MUD on their networks by deploying all of the MUD-capable network components needed to compose a MUD-capable infrastructure.
- Service providers should consider either providing and supporting or encouraging their customers to use MUD-capable routers on their home and small-business networks. (Note: MUD requires the use of a MUD-capable router; this router could be either standalone equipment provided by a third-party network equipment vendor or integrated with the service provider's residential gateway equipment. While service providers are not required to do so, some may choose to make their residential gateway equipment MUD-capable.)

- IoT device manufacturers should configure their devices to emit or otherwise convey a MUD URL.
- IoT device manufacturers should write MUD files for their devices. By doing so, they will be able to provide network administrators the confidence to know what sort of access their device needs (and what sort of access it does not need), and they will do so in a way that someone trained to operate and install the device does not need to understand network administration.
- IoT device manufacturers should ensure that the MUD files for their devices remain continuously available by hosting these MUD files at their specified MUD URLs throughout the devices' life cycles.
- IoT device manufacturers should update each of their MUD files over the course of their devices' life cycles, as needed, if the communication profiles for their devices evolve.
- Even after an IoT device manufacturer deprecates an IoT device so that it will no longer be supported, the manufacturer should continue to make the device's MUD file available so the device's communication profile can continue to be enforced. This will be especially important for deprecated IoT devices that have unpatched vulnerabilities.
- IoT device manufacturers should provide regular updates to patch security vulnerabilities and other bugs that are discovered throughout the life cycle of their devices, and they should make these updates available at a designated URL that is explicitly named in the device's MUD file as being a permissible endpoint with which the device may communicate.
- Manufacturers of MUD managers, MUD-capable DHCP servers, MUD-capable routers, device onboarding equipment, components for supporting threat signaling, components for supporting device discovery, and other networking equipment that is targeted for use on home and small-business networks should strive to make deployment and configuration of these devices as easy to understand and as user-friendly as possible to increase the probability that they will be deployed and configured correctly and securely, even when the person performing the deployment has limited understanding of network administration.
- Home and small-business network owners should use the information presented in the Security Considerations section of the MUD specification (RFC 8520) to enhance protection of MUD deployments.
- Standards development organizations should standardize communications between the MUD manager and the router, between the threat-signaling server and the MUD manager/router, and between the IoT devices and their corresponding update servers.

The following are recommendations for improving the security of home and small-business networks and IoT devices in general:

- Home and small-business network owners should deploy and use equipment and services that apply policies based on threat, thereby benefitting them with available information on known threats.

- Home and small-business network owners should perform periodic updates to all IoT devices to ensure that the devices will be protected with up-to-date software patches.
- IoT device manufacturers should provide ongoing support for the devices that they sell by making regular software updates and patches available on an ongoing basis.
- Home and small-business network owners should have visibility into every device on their network. Any device is a potential attack or reconnaissance point that must be discovered and secured. Non-MUD-capable devices are inviting targets.
- Home and small-business network owners should segment their networks where possible. Where there are IoT devices with known security risks, e.g., non-MUD-capable devices, these devices should be kept on a separate network segment from the everyday computing devices that are afforded a higher level of cybersecurity protection via regular updates and security software. This is an important step to contain any threats that may emerge from the IoT devices.
- Home and small-business network owners should deploy network components that are needed to support a secure, automated, and easy-to-use onboarding protocol, and they should use IoT devices that are capable of being onboarded via this protocol.
- Manufacturers of network equipment that is targeted for use on home and small-business networks should offer components that support secure, automated, and user-friendly IoT device onboarding, threat signaling, and device discovery.
- Service providers should either provide residential gateway equipment that supports secure, automated, and easy-to-use IoT device onboarding; threat signaling; and device discovery, or they should encourage their customers to use third-party equipment with these capabilities on their home and small-business networks.
- IoT device manufacturers should design their devices to be capable of being onboarded via a secure, automated, and easy-to-use process.
- Home and small-business network owners should consider their deployment of MUD to be only one pillar in the overall security of their network and IoT devices. Deployment of MUD is not a substitute for performing best practices to ensure overall, comprehensive security for their network.
- Manufacturers of MUD-capable network components and MUD-capable IoT devices should consider MUD to be only one pillar in helping users secure their networks and IoT devices. Manufacturers should, for example, understand the security and privacy risks posed by their devices as discussed in NISTIR 8228, *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks,* as well as the guidelines for identifying, assessing, and managing security risks that are discussed in the *Framework for Improving Critical Infrastructure Cybersecurity* (Cybersecurity Framework). They should use this information as they make decisions regarding both how they design their MUD-capable components and the default configurations with which they provide these components, being mindful of the fact that home

and small-business network users of their components may have only a limited understanding of network administration and security.

The following recommendations are for the MUD RFC editors:

- Consider revising the MUD specification (RFC 8520) to be explicit about the fact that it is deliberately not specifying what the behavior of the MUD manager should be in the event that the mud-signature element is not present in a MUD file. As currently written, it is reasonable to interpret the RFC in several different ways. It could be interpreted as implying that if the mud-signature element is not present, then:
    - The MUD file has not been signed, so the MUD manager may process the MUD file without attempting to validate its signature. This interpretation is vulnerable to hackers modifying the MUD file and deleting the MUD file's mud-signature element to prevent modification of the MUD file from being detected. Unless all MUD files are required to be signed and to have their signatures validated before processing, it will not be possible for a MUD manager to distinguish between a MUD file that has not been signed and a MUD file that was originally signed but has been modified by an attacker so that its mud-signature element has been deleted.
    - The MUD manager should cease processing the MUD file and wait for administrator input.
    - The MUD manager should attempt to locate and validate the MUD file's signature via some alternative means. However, no such alternative means is mentioned in the RFC. RFC editors may want to consider including suggestions for potential alternative mechanisms for locating MUD file signatures if the mud-signature element (which has been defined as optional) is not present in the MUD file.
- Consider revising Section 16 (Security Considerations) of the MUD specification (RFC 8520) to make readers aware of the security vulnerability that results from using a MUD manager that is configured to automatically process a MUD file that does not have a mud-signature element.
- Consider revising the MUD specification (RFC 8520) to be explicit about the fact that it is deliberately not dictating what action to take (e.g., block or permit traffic to/from the IoT device) if the MUD manager is not able to validate the device's MUD file server's TLS certificate or if the MUD manager is not able to validate the device's MUD file signature. The RFC indicates that the MUD manager should cease processing the MUD file and await administrator approval, but it may be helpful to readers if the RFC were explicit about the fact that it is remaining silent and leaving up to local policy whether the device should be prevented from sending and receiving all traffic (thereby rendering the devices unreachable and effectively useless), whether the device should be permitted to communicate freely (thereby enabling the device to operate as if it did not have an associated MUD file), or whether the device should be subject to some other local policy.
- Consider revising Section 3.5 (Cache-Validity) of the MUD specification (RFC 8520) to explicitly state that if the cache-validity timer has expired, the MUD manager should check for a new

MUD file. We understand that this is the desired behavior; however, it is not currently made clear in the specification.

The following recommendations are suggestions for continuing activity with the collaboration team:

- Continue work with collaborators to enhance MUD capabilities in their commercial products (see Section 10.1).
- Perform additional work that builds on the broader set of security controls identified in Section 5.2.
- Work with collaborators to demonstrate MUD deployments that are configured to address the security considerations that are raised in the MUD specification, such as
  - configuring IoT devices to emit their MUD URLs in a secure fashion by providing the IoT devices with credentials and binding the device’s MUD URLs with their identities
  - restricting the access control permissions of IoT devices that do not emit their MUD URLs in a secure fashion, so they are not elevated beyond those of devices that do not present a MUD policy
  - configuring the MUD manager to raise an exception and seek administrator approval if the signer of a MUD file or the MUD file itself changes
  - for IoT devices that do not emit their MUD URLs in a secure fashion, if their MUD files include rules based on the “manufacturer” construct, performing additional validation measures before admitting the devices to that manufacturer class. For example, look up each device’s MAC address and verify that the manufacturer associated with that MAC address is the same as the manufacturer specified in the “manufacturer” construct in that device’s MUD file
  - incorporating MUD URL discovery and policy into the secure device onboarding process
- Explore the possibility of using crowdsourcing and analytics to perform traffic flow analysis and thereby adapt and evolve traffic profiles of MUD-capable devices over the course of their use. Instead of simply dropping traffic that is received at the router if that traffic is not within the IoT device’s profile, this traffic could be quarantined, recorded, and analyzed for further study. An analytics application that receives such traffic from many sources would be able to analyze the traffic and determine whether there may be valid reasons to expand the device’s communication profile.
- Work with collaborators to define a blueprint to guide IoT device manufacturers as they build MUD support into their devices, from initial device availability to eventual decommissioning. Provide guidance on required and recommended manufacturer activities and considerations.
- Execute performance studies to inform manufacturers of consumer routers how MUD impacts performance. Such studies may address concerns that some manufacturers may have regarding the potential performance impacts of MUD.

11 Future Build Considerations

The number of network components that support the MUD protocol continues to grow rapidly. As more MUD-capable IoT devices become available, these too should be demonstrated. In addition, IPv6, for which no MUD-capable products were available for the initial demonstration sequences, adds a new dimension to using MUD to help mitigate IoT-based DDoS and other network-based attacks. As discussed in Section 11.2, inclusion of IPv6-capability should be considered for future builds.

In addition, operationalization, IoT device onboarding, and IoT device life-cycle issues in general are promising areas for further work. With respect to onboarding, mechanisms for devices to securely provide their MUD URL (in addition to using the Wi-Fi Easy Connect protocol) can be investigated and developed as proof-of-concept implementations.

The following features, which are enhancements that are being implemented in Build 4, are potential candidates for inclusion in future IETF MUD drafts:

- The MUD manager implements device quarantine. A device may enter a "quarantine" state when a packet originating from the device triggers an access violation (i.e., does not match any MUD rules). When the device is in a quarantine state, its access is limited to only those ACEs that are allowable under quarantine.
- The MUD manager implements a MUD reporting capability for manufacturers to be able to get feedback on how their MUD-capable devices are doing in the field. To protect privacy, no identifying information about the device or network is included.

11.1 Extension to Demonstrate the Growing Set of Available Components

Arm, CableLabs, Cisco, CTIA, DigiCert, Forescout, Global Cyber Alliance, MasterPeace Solutions, Molex, Patton Electronics, and Symantec have signed CRADAs and are collaborating in the project. There is also strong interest from additional industry collaborators to participate in future builds, particularly if we expand the project scope to include onboarding. Some collaborators have also expressed interest in our demonstrating the enterprise use case. Several of these new potential collaborators may submit letters of interest leading to CRADAs for participation in tackling the challenge of integrating MUD and other security features into enterprise or industrial IoT use cases.

11.2 Recommended Demonstration of IPv6 Implementation

Due to product limitations, the initial phases of this project involved support for only IPv4 and did not include investigation of IPv6 issues. Additionally, due to the absence of NAT in IPv6, all IPv6 devices are directly addressable. Hence, the potential for DDoS and other attacks against IPv6 networks could potentially be worse than it is against IPv4 networks. Consequently, we recommend that demonstration of MUD in an IPv6 environment be performed as part of follow-on work.

Appendix A List of Acronyms

AAA	Authentication, Authorization, and Accounting
ACE	Access Control Entry
ACK	Acknowledgement
ACL	Access Control List
AP	Access Point
API	Application Programming Interface
CIS	Center for Internet Security
CMS	Cryptographic Message Syntax
COBIT	Control Objectives for Information and Related Technology
CRADA	Cooperative Research and Development Agreement
DACL	Dynamic Access Control List
DB	Database
DDoS	Distributed Denial of Service
Devkit	Development Kit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DVR	Digital Video Recorder
FIPS	Federal Information Processing Standard
GCA	Global Cyber Alliance
GUI	Graphical User Interface
http	Hypertext Transfer Protocol
https	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IOS	Cisco's Internetwork Operating System
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISA	International Society of Automation
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
ISP	Internet Service Provider
IT	Information Technology
ITL	National Institute of Standards and Technology's Information Technology Laboratory
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
LLDP	Link Layer Discovery Protocol (Institute of Electrical and Electronics Engineers 802.1AB)

MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
MSO	Multiple System Operators
MUD	Manufacturer Usage Description
NAT	Network Address Translation
NCCoE	National Cybersecurity Center of Excellence
NIST	National Institute of Standards and Technology
NISTIR	NIST Interagency or Internal Report
NTP	Network Time Protocol
OS	Operating System
PEP	Policy Enforcement Point
PoE	Power over Ethernet
PSK	pre-shared key
QR	Quick Response
RADIUS	Remote Authentication Dial-In User Service
REST	Representational State Transfer
RFC	Request for Comments
RMF	Risk Management Framework
SDN	Software Defined Networking
SP	Special Publication
SSID	service set identifier
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TLV	Type Length Value
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
WAN	Wide Area Network
YANG	Yet Another Next Generation

3889 # Appendix B Glossary

Audit	Independent review and examination of records and activities to assess the adequacy of system controls, to ensure compliance with established policies and operational procedures (National Institute of Standards and Technology (NIST) Special Publication (SP) 800-12 Rev. 1).
Best Practice	A procedure that has been shown by research and experience to produce optimal results and that is established or proposed as a standard suitable for widespread adoption (Merriam-Webster).
Botnet	The word "botnet" is formed from the words "robot" and "network." Cyber criminals use special Trojan viruses to breach the security of several users' computers, take control of each computer, and organize all the infected machines into a network of "bots" that the criminal can remotely manage. (https://usa.kaspersky.com/resource-center/threats/botnet-attacks)
Control	A measure that is modifying risk (Note: Controls include any process, policy, device, practice, or other actions that modify risk.) (NIST Interagency or Internal Report [NISTIR] 8053).
Denial of Service	The prevention of authorized access to a system resource or the delaying of system operations and functions (NIST SP 800-82 Rev. 2)
Distributed Denial of Service (DDoS)	A denial of service technique that uses numerous hosts to perform the attack (NISTIR 7711).
Managed Devices	Personal computers, laptops, mobile devices, virtual machines, and infrastructure components require management agents, allowing information technology staff to discover, maintain, and control them. Those with broken or missing agents cannot be seen or managed by agent-based security products.
Mapping	Depiction of how data from one information source maps to data from another information source.
Mitigate	To make less severe or painful or to cause to become less harsh or hostile (Merriam-Webster).

Manufacturer Usage Description (MUD)	A component-based architecture specified in Request for Comments (RFC) 8250 that is designed to provide a means for end devices to signal to the network what sort of access and network functionality they require to properly function.
MUD-Capable	An Internet of Things (IoT) device that can emit a MUD uniform resource locator in compliance with the MUD specification.
Network Address Translation (NAT)	A function by which internet protocol addresses within a packet are replaced with different IP addresses. This function is most commonly performed by either **routers** or firewalls. It enables private IP networks that **use** unregistered IP addresses to connect to the internet. **NAT** operates on a router, usually connecting two networks together, and translates the private (not globally unique) addresses in the internal network into legal addresses before packets are forwarded to another network.
Non-MUD-Capable	An IoT device that is not capable of emitting a MUD URL in compliance with the MUD specification (RFC 8250).
Onboarding	The process by which a device obtains the credentials (e.g., network SSID and password) that it needs in order to gain access to a wired or wireless network.
Operationalization	Putting MUD implementations into operational service in a manner that is both practical and effective.
Policy	Statements, rules, or assertions that specify the correct or expected behavior of an entity. For example, an authorization policy might specify the correct access control rules for a software component. (NIST SP 800-95 and NISTIR 7621 Rev. 1)
Policy Enforcement Point	A network device on which policy decisions are carried out or enforced.
Risk	The net negative impact of the exercise of a vulnerability, considering both the probability and the impact of occurrence. Risk management is the process of identifying risk, assessing risk, and taking steps to reduce risk to an acceptable level. (NIST SP 800-30)
Router	A computer that is a gateway between two networks at open system interconnection layer 3 and that relays and directs data packets through that

	internetwork. The most common form of router operates on IP packets (NIST SP 800-82 Rev. 2).
Server	A computer or device on a network that manages network resources. Examples include file servers (to store files), print servers (to manage one or more printers), network servers (to manage network traffic), and database servers (to process database queries). (NIST SP 800-47)
Security Control	A safeguard or countermeasure prescribed for an information system or an organization designed to protect the confidentiality, integrity, and availability of its information and to meet a set of defined security requirements (NIST SP 800-53 Rev. 4).
Shall	A requirement that must be met unless a justification of why it cannot be met is given and accepted (NISTIR 5153).
Should	This term is used to indicate an important recommendation. Ignoring the recommendation could result in undesirable results. (NIST SP 800-108)
Threat	Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. Also, the potential for a threat-source to successfully exploit a particular information system vulnerability (Federal Information Processing Standards 200).
Threat Signaling	Real-time signaling of DDoS-related telemetry and threat-handling requests and data between elements concerned with DDoS attack detection, classification, trace back, and mitigation (https://joinup.ec.europa.eu/collection/rolling-plan-ict-standardisation/cybersecurity-network-and-information-security).
Traffic Filter	An entry in an access control list that is installed on the router or switch to enforce access controls on the network.
Uniform Resource Locator (URL)	A reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A typical URL could have the form http://www.example.com/index.html, which indicates a protocol (http), a host name (www.example.com), and a file name (index.html). Also sometimes referred to as a web address.

Update	New, improved, or fixed software, which replaces older versions of the same software. For example, updating an operating system brings it up-to-date with the latest drivers, system utilities, and security software. The software publisher often provides updates free of charge. (https://www.computerhope.com/jargon/u/update.htm)
Update Server	A server that provides patches and other software updates to IoT devices.
VLAN	A broadcast domain that is partitioned and isolated within a network at the data link layer. A single physical local area network (LAN) can be logically partitioned into multiple, independent VLANs; a group of devices on one or more physical LANs can be configured to communicate within the same VLAN, as if they were attached to the same physical LAN.
Vulnerability	Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source (NIST SP 800-37 Rev. 2).

Appendix C Bibliography

FIDO Alliance. Specifications Overview [Website]. Available: https://fidoalliance.org/specifications/overview/.

Internet-Draft draft-srich-opsawg-mud-manu-lifecycle-01. (2017, Mar.) "MUD Lifecyle: A Manufacturer's Perspective" [Online]. Available: https://tools.ietf.org/html/draft-srich-opsawg-mud-manu-lifecycle-01.

Internet-Draft draft-srich-opsawg-mud-net-lifecycle-01. (2017, Sept.) "MUD Lifecyle: A Network Operator's Perspective" [Online]. Available: https://tools.ietf.org/html/draft-srich-opsawg-mud-net-lifecycle-01.

Internet Policy Task Force, National Telecommunications Information Administration. Multi-stakeholder Working Group for Secure Update of IoT Devices [Website]. Available: https://www.ntia.doc.gov/category/internet-things.

National Institute of Standards and Technology (NIST). (2018, Apr.) Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf.

National Institute of Standards and Technology (NIST) Draft Interagency or Internal Report 7823. (2012, Jul.) Advanced Metering Infrastructure Smart Meter Upgradeability Test Framework [Online]. Available: http://csrc.nist.gov/publications/drafts/nistir-7823/draft_nistir-_7823.pdf.

National Institute of Standards and Technology (NIST) Interagency or Internal Report 8228. (2018, Sept.) Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks [Online]. Available: https://doi.org/10.6028/NIST.IR.8228.

National Institute of Standards and Technology (NIST). NIST Computer Security Resource Center Risk Management Framework guidance [Website]. Available: https://csrc.nist.gov/projects/risk-management/risk-management-framework-quick-start-guides.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-30. (2002, Jul.) Risk Management Guide for Information Technology Systems [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-30 Revision 1. (2012, Sept.) Guide for Conducting Risk Assessments [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-30r1.pdf.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-37 Revision 2. (2018, Dec.) Risk Management Framework for Information Systems and Organizations [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-37r2.pdf.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-40 Rev. 3. (2013, Jul.) Guide to Enterprise Patch Management Technologies [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-40/rev-3/final.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-52 Revision 2. (2019, Aug.) Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations [Online]. Available: https://doi.org/10.6028/NIST.SP.800-52r2.

National Institute of Standards and Technology (NIST) Draft Special Publication (SP) 800-53 Rev. 5. (2017, Aug.) Security and Privacy Controls for Information Systems and Organizations (Draft) [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/draft.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-57 Part 1 Revision 4. (2016, Jan.) Recommendation for Key Management [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-63-3. (2017, Jun.) Digital Identity Guidelines [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-63/3/final.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-63-B. (2017, Jun.) Digital Identity Guidelines: Authentication and Lifecycle Management [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-63b/final.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-147. (2011, Apr.) BIOS Protection Guidelines [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-147/final.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-147B. (2014, Aug.) BIOS Protection Guidelines for Servers [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-147B.pdf.

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-193. (2018, May.) Platform Firmware Resiliency Guidelines [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-193.pdf.

Office of Management and Budget (OMB) Circular A-130 Revised. (2016, Jul.) Managing Information as a Strategic Resource [Online]. Available: https://obamawhitehouse.archives.gov/omb/circulars_a130_a130trans4/.

Request for Comments (RFC) 2131. (1997, Mar.) "Dynamic Host Configuration Protocol" [Online]. Available: https://tools.ietf.org/html/rfc2131.

Request for Comments (RFC) 2818. (2000, May.) "HTTP Over TLS" [Online]. Available: https://tools.ietf.org/html/rfc2818.

Request for Comments (RFC) 5280. (2008, May.) "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" [Online]. Available: https://tools.ietf.org/html/rfc5280.

Request for Comments (RFC) 5652. (2009, Sept.) "Cryptographic Message Syntax (CMS)" [Online]. Available: https://tools.ietf.org/html/rfc5652.

Request for Comments (RFC) 6020. (2010, Oct.) "YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)" [Online]. Available: https://tools.ietf.org/html/rfc6020.

Request for Comments (RFC) 8520. (2019, Mar.). "Manufacturer Usage Description Specification" [Online]. Available: https://tools.ietf.org/html/rfc8520.

SANS Institute. CWE/SANS Top 25 Most Dangerous Software Errors [Website]. Available: https://www.sans.org/top25-software-errors/.

Wi-Fi Alliance. DRAFT Device Provisioning Protocol Specification Version 1.2, 2020. Available: https://www.wi-fi.org/file/device-provisioning-protocol-draft-specification.

DRAFT

NIST SPECIAL PUBLICATION 1800-15C

Securing Small-Business and Home Internet of Things (IoT) Devices:

Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)

Volume C:
How-To Guides

Mudumbai Ranganathan
NIST

Steve Johnson
Ashwini Kadam
Craig Pratt
Darshak Thakore
CableLabs

Eliot Lear
Cisco

William C. Barker
Dakota Consulting

Adnan Baykal
Global Cyber Alliance

Drew Cohen
Kevin Yeich
MasterPeace Solutions

Yemi Fashina
Parisa Grayeli
Joshua Harrington
Joshua Klosterman
Blaine Mulugeta
Susan Symington
The MITRE Corporation

September 2020

DRAFT

This publication is available free of charge from:
https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos

DISCLAIMER

National Institute of Standards and Technology Special Publication 1800-15C, Natl. Inst. Stand. Technol. Spec. Publ. 1800-15C, 273 pages, (September 2020), CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback. As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: mitigating-iot-ddos-nccoe@nist.gov.

Public comment period: September 16, 2020 through October 16, 2020

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, easily adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align more easily with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

ABSTRACT

The goal of the Internet Engineering Task Force's Manufacturer Usage Description (MUD) architecture is for Internet of Things (IoT) devices to behave as intended by the manufacturers of the devices. This is done by providing a standard way for manufacturers to indicate the network communications that a device requires to perform its intended function. When MUD is used, the network will automatically permit the IoT device to send and receive only the traffic it requires to perform as intended, and the network will prohibit all other communication with the device, thereby increasing the device's resilience to network-based attacks. In this project, the NCCoE has demonstrated the ability to ensure that when an IoT device connects to a home or small-business network, MUD can be used to automatically permit

the device to send and receive only the traffic it requires to perform its intended function. This NIST Cybersecurity Practice Guide explains how MUD protocols and tools can reduce the vulnerability of IoT devices to botnets and other network-based threats as well as reduce the potential for harm from exploited IoT devices. It also shows IoT device developers and manufacturers, network equipment developers and manufacturers, and service providers who employ MUD-capable components how to integrate and use MUD to satisfy IoT users' security requirements.

KEYWORDS

access control; bootstrapping; botnets; firewall rules; flow rules; Internet of Things; IoT; Manufacturer Usage Description; MUD; network segment; onboarding; router; server; threat signaling; update server; Wi-Fi Easy Connect.

DOCUMENT CONVENTIONS

The terms "shall" and "shall not" indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted.

The terms "should" and "should not" indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others or that a certain course of action is preferred but not necessarily required or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited.

The terms "may" and "need not" indicate a course of action permissible within the limits of the publication.

The terms "can" and "cannot" indicate a possibility and capability, whether material, physical, or causal.

Acronyms used in figures can be found in the Acronyms appendix.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory [ITL] draft publication). Such guidance and/or requirements may be directly stated in this ITL publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

1. assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

2. assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

   a. under reasonable terms and conditions that are demonstrably free of any unfair discrimination or

   b. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to mitigating-iot-ddos-nccoe@nist.gov

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Allaukik Abhishek	Arm
Michael Bartling	Arm
Mark Walker	CableLabs
Tao Wan	CableLabs
Russ Gyurek	Cisco
Peter Romness	Cisco
Brian Weis	Cisco

Name	Organization
Rob Cantu	CTIA
Dean Coclin	DigiCert
Avesta Hojjati	DigiCert
Clint Wilson	DigiCert
Katherine Gronberg	Forescout
Tim Jones	Forescout
Rae'-Mar Horne	MasterPeace Solutions, Ltd.
Nate Lesser	MasterPeace Solutions, Ltd.
Tom Martz	MasterPeace Solutions, Ltd.
Daniel Weller	MasterPeace Solutions, Ltd.
Nancy Correll	The MITRE Corporation
Sallie Edwards	The MITRE Corporation
Drew Keller	The MITRE Corporation
Sarah Kinling	The MITRE Corporation
Karri Meldorf	The MITRE Corporation
Mary Raguso	The MITRE Corporation

Name	Organization
Allen Tan	The MITRE Corporation
Mo Alhroub	Molex
Jaideep Singh	Molex
Bill Haag	NIST
Tim Polk	NIST
Murugiah Souppaya	NIST
Paul Watrobski	NIST
Bryan Dubois	Patton Electronics
Stephen Ochs	Patton Electronics
Karen Scarfone	Scarfone Cybersecurity
Matt Boucher	Symantec
Petros Efstathopoulos	Symantec
Bruce McCorkendale	Symantec
Susanta Nanda	Symantec
Yun Shen	Symantec
Pierre-Antoine Vervier	Symantec

Name	Organization
John Bambenek	ThreatSTOP
Russ Housley	Vigil Security

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Partner/Collaborator	Build Involvement
Arm	Subject matter expertise
CableLabs	Micronets Gateway Micronets cloud infrastructure Prototype IoT devices–Raspberry Pi with Wi-Fi Easy Connect support Micronets mobile application
Cisco	Cisco Catalyst 3850S MUD manager
CTIA	Subject matter expertise
DigiCert	Private Transport Layer Security certificate Premium Certificate
Forescout	Forescout appliance–VCT-R Enterprise manager–VCEM-05
Global Cyber Alliance	Quad9 DNS service, Quad9 Threat Application Programming Interface ThreatSTOP threat MUD file server
MasterPeace Solutions	Yikes! router Yikes! cloud Yikes! mobile application

Technology Partner/Collaborator	Build Involvement
Molex	Molex light-emitting diode light bar Molex Power over Ethernet Gateway
Patton Electronics	Subject matter expertise
Symantec	Subject matter expertise

Contents

List of Figures

List of Tables

1 Introduction

This following volumes of this guide show information technology (IT) professionals and security engineers how we implemented this example solution. We cover all of the products employed in this reference design. We do not re-create the product manufacturers' documentation, which is presumed to be widely available. Rather, these volumes show how we incorporated the products together in our environment.

Note: These are not comprehensive tutorials. There are many possible service and security configurations for these products that are out of scope for this reference design.

1.1 How to Use this Guide

This National Institute of Standards and Technology (NIST) Cybersecurity Practice Guide demonstrates a standards-based reference design for mitigating network-based attacks by securing home and small-business Internet of Things (IoT) devices. The reference design is modular, and it can be deployed in whole or in part. This practice guide provides users with the information they need to replicate four example MUD-based implementations of this reference design. These example implementations are referred to as Builds, and this volume describes in detail how to reproduce each one.

This guide contains three volumes and a supplement:

- NIST SP 1800-15A: *Executive Summary – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge*
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics – what we built and why, including the risk analysis performed, and the security control map*
- NIST SP 1800-15C: *How-To Guides – instructions for building the example implementations including all the security relevant details that would allow you to replicate all or parts of this project* ***(you are here)***
- Functional Demonstration Results - supplement to NIST SP 1800-15B: *describes the functional demonstration results for the four implementations of the MUD-based reference solution*

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary,* NIST SP 1800-15A, which describes the following topics:

- challenges that enterprises face in trying to mitigate network-based attacks by securing home and small-business IoT devices
- example solutions built at the National Cybersecurity Center of Excellence (NCCoE)
- benefits of adopting the example solutions

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in NIST SP 1800-15B, which describes what we did and why. The following sections will be of particular interest:

- Section 3.4, Risk Assessment, describes the risk analysis we performed.
- Section 5.2, Security Control Map, maps the security characteristics of these example solutions to cybersecurity standards and best practices.

You might share the *Executive Summary,* NIST SP 1800-15A, with your leadership team members to help them understand the importance of adopting a standards-based solution for mitigating network-based attacks by securing home and small-business IoT devices.

IT professionals who want to implement an approach like this will find this whole practice guide useful. You can use this How-To portion of the guide, NIST SP 1800-15C, to replicate all or parts of one or all four builds created in our lab. This How-To portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solutions. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of products to address this challenge, this guide does not endorse these particular products. Your organization can adopt one of these solutions or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a Manufacturer Usage Description (MUD)-based solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and best practices. NIST SP 1800-15B lists the products that we used in each build and maps them to the cybersecurity controls provided by this reference solution.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but a possible solution. In the case of this guide, it describes four possible solutions. This is a draft guide. We seek feedback on its contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to mitigating-iot-ddos-nccoe@nist.gov.

1.2 Build Overview

This NIST Cybersecurity Practice Guide addresses the challenge of using standards-based protocols and available technologies to mitigate network-based attacks by securing home and small-business IoT devices. It identifies three key forms of protection:

- use of the MUD specification to automatically permit an IoT device to send and receive only the traffic it requires to perform as intended, thereby reducing the potential for the device to be the victim of a network-based attack, as well as the potential for the device, if compromised, to be used in a network-based attack

- use of network-wide access controls based on threat intelligence to protect all devices (both MUD-capable and non-MUD-capable) from connecting to domains that are known current threats
- automated secure software updates to all devices to ensure that operating system (OS) patches are installed promptly

Four builds that serve as example solutions of how to support the MUD specification have been implemented and demonstrated as part of this project. This practice guide provides instructions for reproducing these four builds.

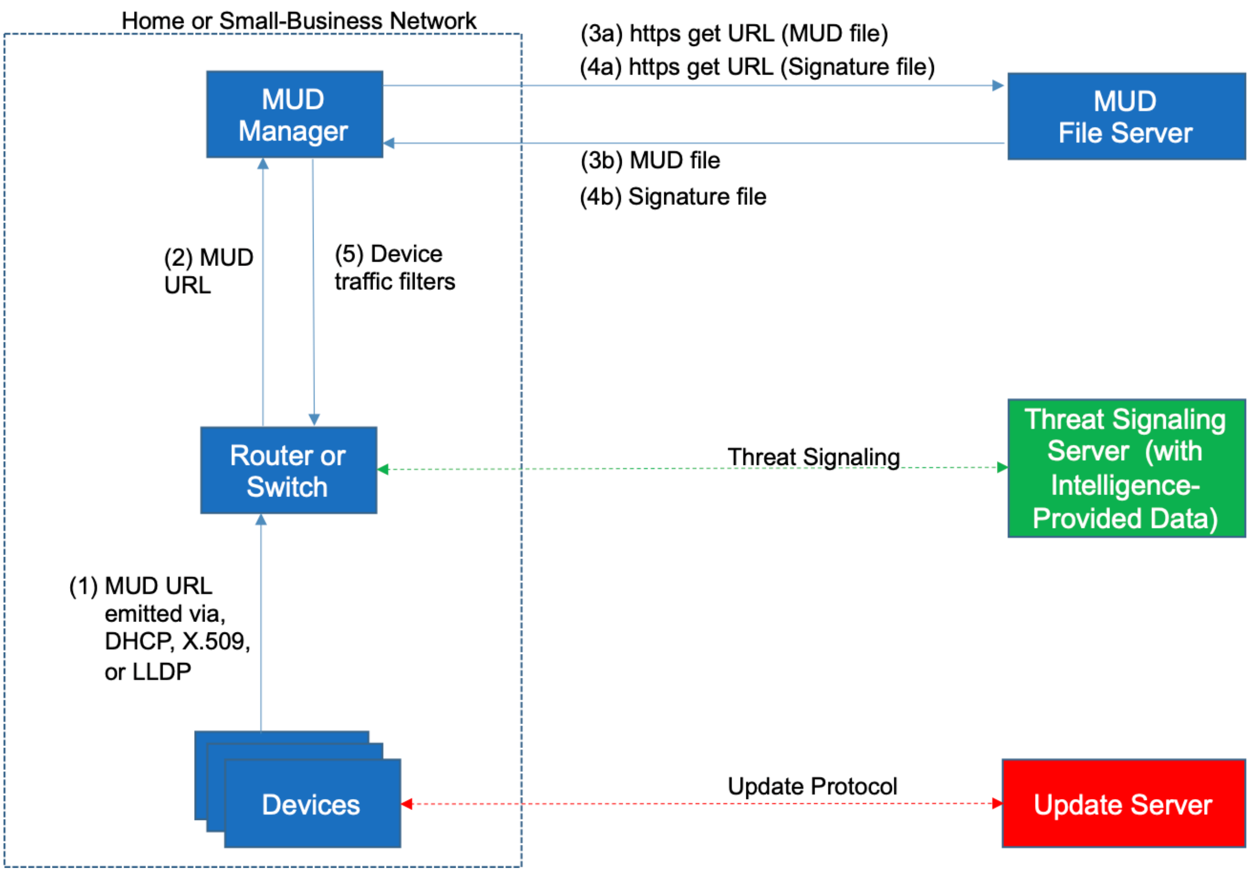
1.2.1 Usage Scenarios

Each of the four builds is designed to fulfill the use case of a MUD-capable IoT device being onboarded and used on home and small-business networks, where plug-and-play deployment is required. All four builds include both MUD-capable and non-MUD-capable IoT devices. MUD-capable IoT devices include the Molex Power over Ethernet (PoE) Gateway and Light Engine as well as four development kits (devkits) that the National Cybersecurity Center of Excellence (NCCoE) configured to perform actions such as power a light-emitting diode (LED) bulb on and off, start network connections, and power a connected lighting device on and off. These MUD-capable IoT devices interact with external systems to access notional, secure updates and various cloud services, in addition to interacting with traditional personal computing devices, as permitted by their MUD files. Non-MUD-capable IoT devices deployed in the builds include three cameras, two mobile phones, two connected lighting devices, a connected assistant, a connected printer, a baby monitor with remote control and video and audio capabilities, a connected wireless access point, and a connected digital video recorder. The cameras, connected lighting devices, baby monitor, and connected digital video recorder are all controlled and managed by a mobile phone. In combination, these devices are capable of generating a wide range of network traffic that could reasonably be expected on a home or small-business network.

1.2.2 Reference Architecture Overview

Figure 1-1 depicts a general reference design for all four builds. It consists of three main components: support for MUD, support for threat signaling, and support for periodic updates.

Figure 1-1 Reference Architecture



1.2.2.1 Support for MUD

A new functional component, the MUD manager, is introduced to augment the existing networking functionality offered by the home/small-business network router or switch. Note that the MUD manager is a logical component. Physically, the functionality it provides can and often will be combined with that of the network router or switch in a single device.

IoT devices must somehow be associated with a MUD file. The MUD specification describes three possible mechanisms through which the IoT device can provide the MUD file URL to the network: inserting the MUD URL into the Dynamic Host Configuration Protocol (DHCP) address requests that they generate when they attach to the network (e.g., when powered on), providing the MUD URL in a Link Layer Discovery Protocol (LLDP) frame, or providing the MUD URL as a field in an X.509 certificate that the device provides to the network via a protocol such as Tunnel Extensible Authentication Protocol. In addition, the MUD specification provides flexibility to enable other mechanisms by which MUD file URLs can be associated with IoT devices. One such alternative mechanism is to associate the device with its MUD file by using the device's bootstrapping information that is conveyed as part of the Wi-Fi Easy Connect (also referred to as Device Provisioning Protocol—DPP) onboarding process. This is the mechanism implemented in Build 3.

Figure 1-1 uses labeled arrows to depict the steps involved in supporting MUD:

- The IoT device emits a MUD URL by using a mechanism such as DHCP, LLDP, or X.509 certificate (step 1).
- The router extracts the MUD URL from the protocol frame of whatever mechanism was used to convey it and forwards this MUD URL to the MUD manager (step 2).
- Once the MUD URL is received, the MUD manager uses https to request the MUD file from the MUD file server by using the MUD URL provided in the previous step (step 3a); if successful, the MUD file server at the specified location will serve the MUD file (step 3b).
- Next, the MUD manager uses https to request the signature file associated with the MUD file (step 4a) and upon receipt (step 4b) verifies the MUD file by using its signature file.
- The MUD file describes the communications requirements for the IoT device. Once the MUD manager has determined the MUD file to be valid, the MUD manager converts the access control rules in the MUD file into access control entries (e.g., access control lists—ACLs, firewall rules, or flow rules) and installs them on the router or switch (step 5).

Once the device's access control rules are applied to the router or switch, the MUD-capable IoT device will be able to communicate with approved local hosts and internet hosts as defined in the MUD file, and any unapproved communication attempts will be blocked.

1.2.2.2 Support for Updates

To provide additional security, the reference architecture also supports periodic updates. All builds include a server that is meant to represent an update server to which MUD will permit devices to connect. Each IoT device on an operational network should be configured to periodically contact its update server to download and apply security patches, ensuring that it is running the most up-to-date and secure code available. To ensure that such updates are possible, the IoT device's MUD file must explicitly permit the IoT device to receive traffic from the update server. Although regular manufacturer updates are crucial to IoT security, the builds described in this practice guide demonstrate only the ability to receive faux updates from a notional update server.

1.2.2.3 Support for Threat Signaling

To provide additional protection for both MUD-capable and non-MUD-capable devices, the reference architecture also incorporates support for threat signaling. The router or switch can receive threat feeds from a threat signaling server to use as a basis for restricting certain types of network traffic. For example, both MUD-capable and non-MUD-capable devices can be prevented from connecting to internet domains that have been identified as potentially malicious.

1.2.2.4 Build-Specific Features

The reference architecture depicted in Figure 1-1 is intentionally general. Each build instantiates this reference architecture in a unique way, depending on the equipment used and the capabilities supported. The logical and physical architectures of each build are depicted and described in NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics*. While all four builds support MUD and the ability to receive faux updates from a notional update server, only Build 2 currently supports threat signaling. Only Build 3 currently supports onboarding MUD-capable devices using the Wi-Fi Alliance Wi-Fi Easy Connect protocol. Build 1 and Build 2 include nonstandard device discovery technology to discover, inventory, profile, and classify attached devices. Such classification can be used to validate that the access being granted to each device is consistent with that device's manufacturer and model. In Build 2, a device's manufacturer and model can be used as a basis for identifying and enforcing that device's traffic profile.

Briefly, the four builds of the reference architecture that have been completed and demonstrated are as follows:

- Build 1 uses products from Cisco Systems, DigiCert, Forescout, and Molex. The Cisco MUD manager supports MUD, and the Forescout virtual appliances and enterprise manager perform non-MUD-related device discovery on the network. Molex PoE Gateway and Light Engine is used as a MUD-capable IoT device. Certificates from DigiCert are also used.
- Build 2 uses products from MasterPeace Solutions Ltd., Global Cyber Alliance (GCA), ThreatSTOP, and DigiCert. The MasterPeace Solutions Yikes! router, cloud service, and mobile application support MUD as well as perform device discovery on the network and apply additional traffic rules to both MUD-capable and non-MUD-capable devices based on device manufacturer and model. The GCA threat agent, Quad9 DNS service, and ThreatSTOP threat MUD file server support threat signaling. Certificates from DigiCert are also used.
- Build 3 uses products from CableLabs and DigiCert. CableLabs Micronets (e.g., Micronets Gateway, Micronets Manager, Micronets mobile phone application, and related service provider cloud-based infrastructure) supports MUD and implements the Wi-Fi Alliance's Wi-Fi Easy Connect protocol to securely onboard devices to the network. It also uses software-defined networking to create separate trust zones (e.g., network segments) called *micronets* to which devices are assigned according to their intended network function. Certificates from DigiCert are also used.
- Build 4 uses software developed at the NIST Advanced Networking Technologies laboratory. This software supports MUD and is intended to serve as a working prototype of the MUD request for comments (RFC) to demonstrate feasibility and scalability. Certificates from DigiCert are also used.

The logical architectures and detailed descriptions of Builds 1, 2, 3, and 4 can be found in NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics*.

1.2.3 Physical Architecture Overview

Figure 1-2 depicts the high-level physical architecture of the NCCoE laboratory environment. This implementation currently supports four builds and has the flexibility to implement additional builds in the future. As depicted, the NCCoE laboratory network is connected to the internet via the NIST data center. Access to and from the NCCoE network is protected by a firewall. The NCCoE network includes a shared virtual environment that houses an update server, a MUD file server, an unapproved server (i.e., a server that is not listed as a permissible communications source or destination in any MUD file), a Message Queuing Telemetry Transport (MQTT) broker server, and a Forescout enterprise manager. These components are hosted at the NCCoE and are used across builds where applicable. The Transport Layer Security (TLS) certificate and Premium Certificate used by the MUD file server are provided by DigiCert.

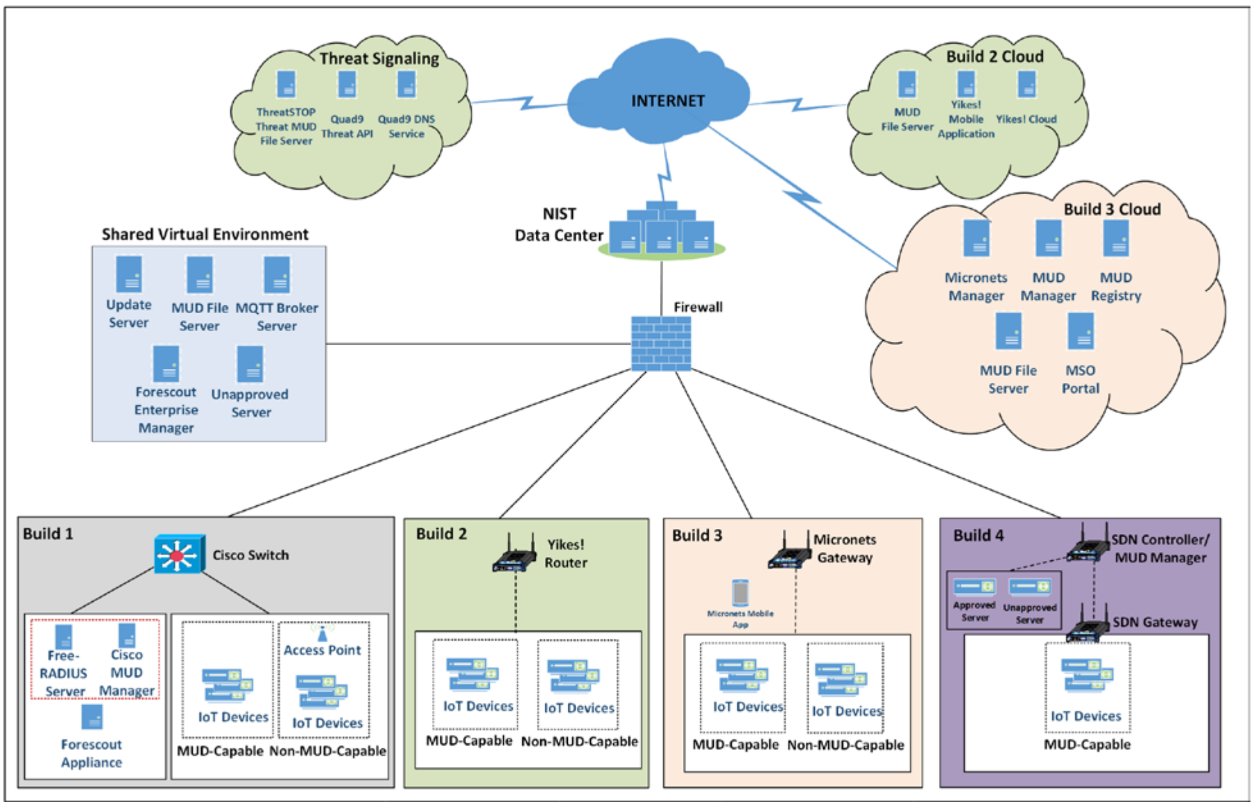
The following four builds, as depicted in the diagram, are supported within the physical architecture:

- Build 1 network components consist of a Cisco Catalyst 3850-S switch, a Cisco MUD manager, a FreeRADIUS server, and a virtualized Forescout appliance on the local network. Build 1 also requires support from all components that are in the shared virtual environment, including the Forescout enterprise manager.
- Build 2 network components consist of a MasterPeace Solutions Ltd. Yikes! router on the local network. Build 2 requires support from the MUD file server, Yikes! cloud, and a Yikes! mobile application that are resident on the Build 2 cloud. The Yikes! router includes threat-signaling capabilities (not depicted) that have been integrated with it. Build 2 also requires support from threat-signaling cloud services that consist of the ThreatSTOP threat MUD file server, Quad9 threat application programming interface (API), and Quad9 DNS service. Build 2 uses only the update server and unapproved server components that are in the shared virtual environment.
- Build 3 network components consist of a CableLabs Micronets Gateway/wireless access point (AP). The Gateway/wireless AP resides on the local network and operates in conjunction with various service provider components and partner/service provider offerings that reside in the Micronets virtual environment in the Build 3 cloud. The Micronets Gateway is controlled by a Micronets Manager that resides in the Build 3 cloud and that coordinates a number of cloud-based Micronets micro-services, some of which are depicted. Build 3 also includes a Micronets mobile application that provides the user and device interfaces for device onboarding.
- Build 4 network components consist of a software-defined networking (SDN)-capable gateway/switch on the local network and an SDN controller/MUD manager and approved and unapproved servers that are located remotely from the local network. Build 4 also uses the MUD file server that is resident in the shared virtual environment.

IoT devices used in all four builds include both MUD-capable and non-MUD-capable IoT devices. The MUD-capable IoT devices used, which vary across builds, include Raspberry Pi, ARTIK, u-blox, Intel UP Squared, BeagleBone Black, NXP i.MX 8M (devkit), and the Molex Light Engine controlled by PoE Gateway. Non-MUD-capable devices used, which also vary across builds, include a wireless access point, cameras, a printer, mobile phones, lighting devices, a connected assistant device, a baby monitor, and a digital video recorder. Each of the completed builds and the roles that their components play in their architectures are explained in more detail in NIST SP 1800-15B.

The remainder of this guide describes how to implement Builds 1, 2, 3, and 4.

Figure 1-2 NCCoE Physical Architecture



1.3 Typographic Conventions

The following table presents typographic conventions used in this volume.

Typeface/Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide*.
Bold	names of menus, options, command buttons, and fields	Choose **File > Edit.**
`Monospace`	command-line input, onscreen computer output, sample code examples, and status codes	`Mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

2 Build 1 Product Installation Guides

This section of the practice guide contains detailed instructions for installing and configuring all the products used to implement Build 1. For additional details on Build 1's logical and physical architectures, please refer to NIST SP 1800-15B.

2.1 Cisco MUD Manager

This section describes how to deploy Cisco's MUD manager version 1.0, which uses a MUD-based authorization system in the network, using Cisco Catalyst switches, FreeRADIUS, and Cisco MUD manager.

2.1.1 Cisco MUD Manager Overview

The Cisco MUD manager is an open-source implementation that works with IoT devices that emit their MUD URLs. In this implementation we tested two MUD URL emission methods: DHCP and LLDP. The MUD manager is supported by a FreeRADIUS server that receives MUD URLs from the switch. The MUD URLs are extracted by the DHCP server and are sent to the MUD manager via Remote Authentication Dial-In User Service (RADIUS) messages. The MUD manager is responsible for retrieving the MUD file and corresponding signature file associated with the MUD URL. The MUD manager verifies the legitimacy of the file and then translates the contents to an internet protocol (IP) ACL-based policy that is installed on the switch.

The version of the Cisco MUD manager used in this project is a proof-of-concept implementation that is intended to introduce advanced users and engineers to the MUD concept. It is not a fully automated MUD manager implementation, and some protocol features are not present. At implementation, the “model” construct was not yet implemented. In addition, if a DNS-based system changes its address, this will not be noticed. Also, IPv6 access has not been fully supported.

2.1.2 Cisco MUD Manager Configurations

The following subsections document the software, hardware, and network configurations for the Cisco MUD manager.

2.1.2.1 Hardware Configuration

Cisco requires installing the MUD manager and FreeRADIUS on a single server with at least 2 gigabytes of random access memory. This server must integrate with at least one switch or router on the network. For this build we used a Catalyst 3850-S switch.

2.1.2.2 Network Configuration

The MUD manager and FreeRADIUS server instances were installed and configured on a dedicated machine leveraged for hosting virtual machines in the Build 1 lab environment. This machine was then connected to virtual local area network (VLAN) 2 on the Catalyst 3850-S and assigned a static IP address.

2.1.2.3 Software Configuration

For this build, the Cisco MUD manager was installed on an Ubuntu 18.04.01 64-bit server. However, there are many approaches for implementation. Alternatively, the MUD manager can be built via docker containers provided by Cisco.

The Cisco MUD manager can operate on Linux operating systems, such as

- Ubuntu 18.04.01
- Amazon Linux

The Cisco MUD manager requires the following installations and components:

- OpenSSL
- cJSON
- MongoDB
- Mongo C driver

- Libcurl
- FreeRADIUS server

At a high level, the following software configurations and integrations are required:

- The Cisco MUD manager requires integration with a switch (such as a Catalyst 3850-S) that connects to an authentication, authorization, and accounting (AAA) server that communicates by using the RADIUS protocol (i.e., a RADIUS server).
- The RADIUS server must be configured to identify a MUD URL received in an accounting request message from a device it has authenticated.
- The MUD manager must be configured to process a MUD URL received from a RADIUS server and return access control policy to the RADIUS server, which is then forwarded to the switch.

2.1.3 Setup

2.1.3.1 Preinstallation

Cisco's DevNet GitHub page provides documentation that we followed to complete this section: https://github.com/CiscoDevNet/MUD-Manager/tree/3.0.1#dependancies

1. Open a terminal window, and enter the following command to log in as root:

   ```
   sudo su
   ```

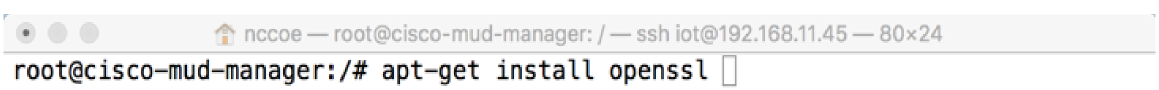
2. Change to the root directory:

   ```
   cd /
   ```

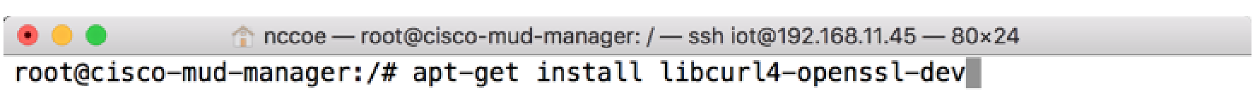
3. To install OpenSSL from the terminal, enter the following command:

   ```
   apt-get install openssl
   ```

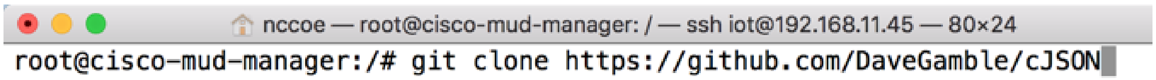
   a. If unable to link to OpenSSL, install the following by entering this command:

      ```
      apt-get install -y libssl-dev
      ```

4. To install cJSON, download it from GitHub by entering the following command:

```
git clone https://github.com/DaveGamble/cJSON
```

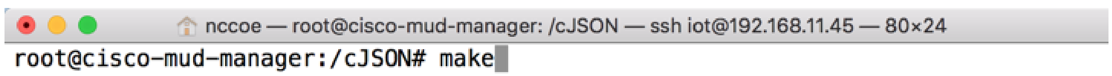


   a. Change directories to the cJSON folder by entering the following command:

```
cd cJSON
```

   b. Build cJSON by entering the following commands:

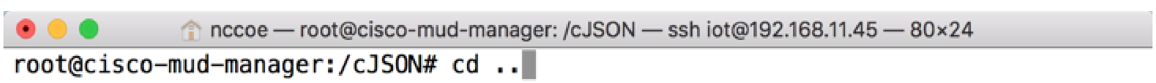
```
make
```




```
make install
```

5. Change directories back a folder by entering the following command:

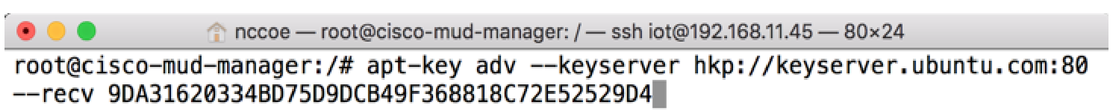
```
cd ..
```



6. To install MongoDB, enter the following commands:

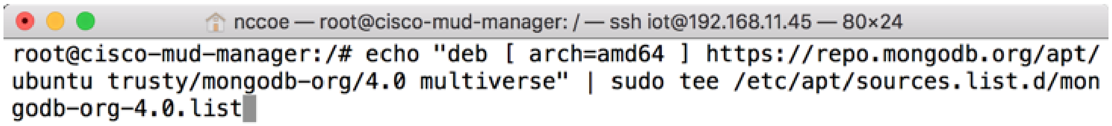
   a. Import the public key:

```
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
9DA31620334BD75D9DCB49F368818C72E52529D4
```



   b. Create a list file for MongoDB:

```
echo "deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu trusty/mongodb-
org/4.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.0.list
```

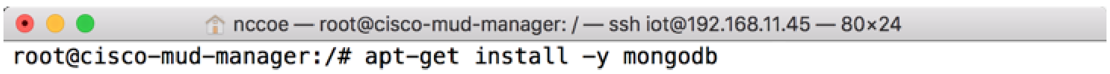


c. Reload the local package database:

```
apt-get update
```

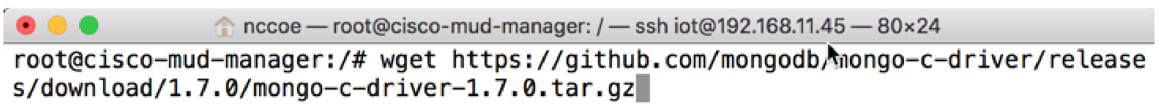
d. Install the MongoDB packages:

```
apt-get install -y mongodb
```



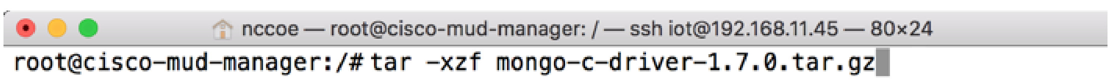
7. To install the Mongo C driver, enter the following command:

```
wget https://github.com/mongodb/mongo-c-driver/releases/download/1.7.0/mongo-c-driver-1.7.0.tar.gz
```



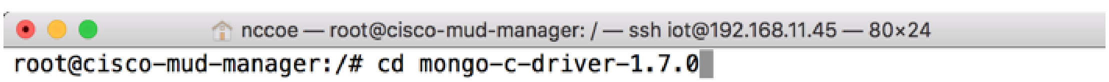
a. Untar the file by entering the following command:

```
tar -xzf mongo-c-driver-1.7.0.tar.gz
```



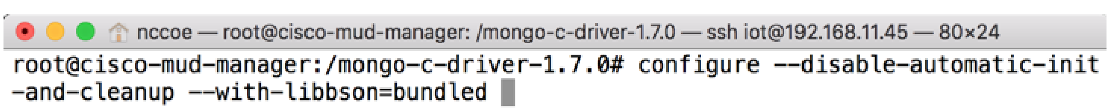
b. Change into the mongo-c-driver-1.7.0 directory by entering the following command:

```
cd mongo-c-driver-1.7.0/
```

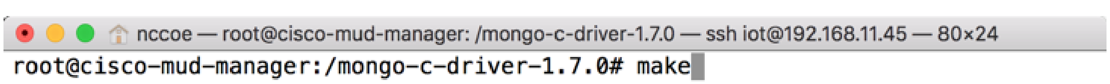


c. Build the Mongo C driver by entering the following commands:

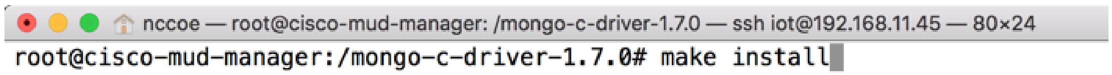
```
./configure --disable-automatic-init-and-cleanup --with-libbson=bundled
```




```
make
```

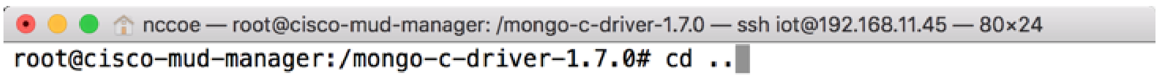


```
make install
```



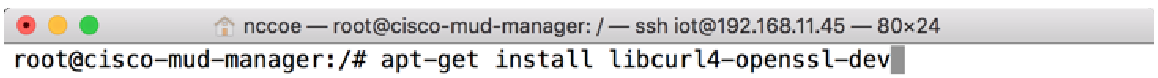
8. Change directories back a folder by entering the following command:

```
cd ..
```



9. To install libcurl, enter the following command:

```
sudo apt-get install libcurl4-openssl-dev
```



2.1.3.2 MUD Manager Installation

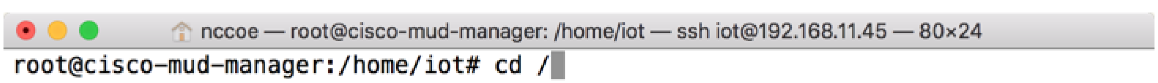
A portion of the steps in this section are documented on Cisco's DevNet GitHub page: https://github.com/CiscoDevNet/MUD-Manager/tree/3.0.1#building-the-mud-manager

1. Open a terminal window, and enter the following command to log in as root:

```
sudo su
```

2. Change to the root directory by entering the following command:

```
cd /
```

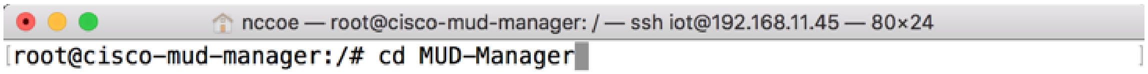


3. To install the MUD manager, download it from Cisco's GitHub by entering the following command:

```
git clone https://github.com/CiscoDevNet/MUD-Manager.git
```

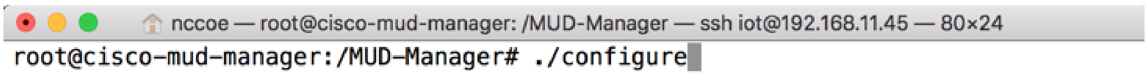
4. Change into the MUD manager directory:

```
cd MUD-Manager
```



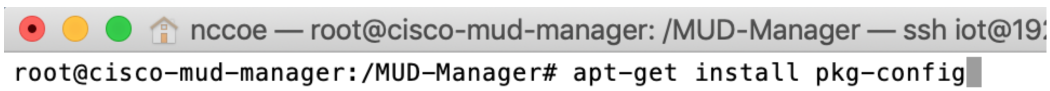
5. Build the MUD manager by entering the following commands:

```
./configure
```

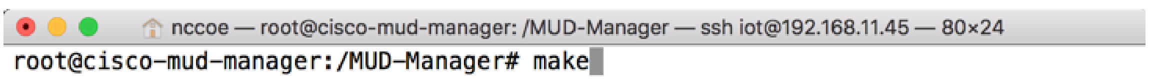


Note: If a “pkg-config error” is thrown, run the command below to install the missing package:

```
apt-get install pkg-config
```

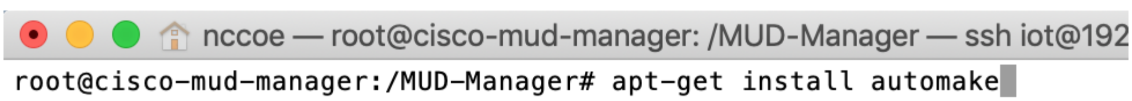



```
make
```

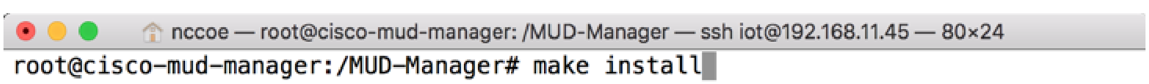


Note: If an “ac.local error” is thrown, run the command below to install the missing package:

```
apt-get install automake
```




```
make install
```



2.1.3.3 MUD Manager Configuration

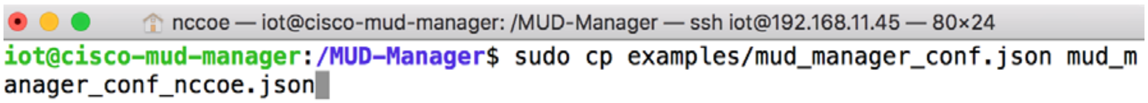
This section describes configuring the MUD manager to communicate with the NCCoE MUD file server and defining the attributes used for translating the fetched MUD files. Details about the configuration file and additional fields that can be set within this file can be accessed here: https://github.com/CiscoDevNet/MUD-Manager#editing-the-configuration-file.

1. In the terminal, change to the MUD manager directory:

```
cd /MUD-Manager
```

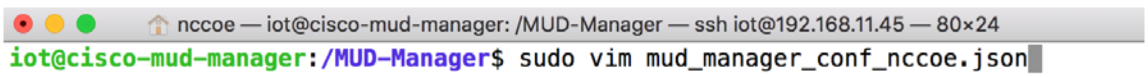
2. Copy the contents of the sample *mud_manager_conf.json* file to a different file:

```
sudo cp examples/mud_manager_conf.json mud_manager_conf_nccoe.json
```



3. Modify the contents of the new MUD manager configuration file:

```
sudo vim mud_manager_conf_nccoe.json
```




```
{
        "MUD_Manager_Version" : 3,
        "MUDManagerAPIProtocol" : "http",
        "ACL_Prefix" : "ACS:",
        "ACL_Type" : "dACL-ingress-only",
        "COA_Password" : "cisco",
        "VLANs" : [
                {       "VLAN_ID" : 3,
                        "v4addrmask" : "192.168.13.0 0.0.0.255"
                },
                {       "VLAN_ID" : 4,
                        "v4addrmask" : "192.168.14.0 0.0.0.255"
                },
                {       "VLAN_ID" : 5,
                        "v4addrmask" : "192.168.15.0 0.0.0.255"
                }
        ],
        "Manufacturers" : [
                { "authority" : "mudfileserver",
                  "cert" : "/home/mudtester/digicertca-chain.crt",
                  "web_cert": "/home/mudtester/digicertchain.pem",
                  "my_controller_v4" : "192.168.10.125",
                  "my_controller_v6" : "2610:20:60CE:630:B000::7",
                  "local_networks_v4" : "192.168.10.0 0.0.0.255",
                  "local_networks_v6" : "2610:20:60CE:630:B000::",
                  "vlan_nw_v4" : "192.168.13.0 0.0.0.255",
                  "vlan" : 3
                },
                {
                "authority" : "www.gmail.com",
                  "cert" : "/home/mudtester/digicertca-chain.crt",
                  "web_cert": "/home/mudtester/digicertchain.pem",
                  "vlan_nw_v4" : "192.168.14.0 0.0.0.255",
                  "vlan" : 4
```

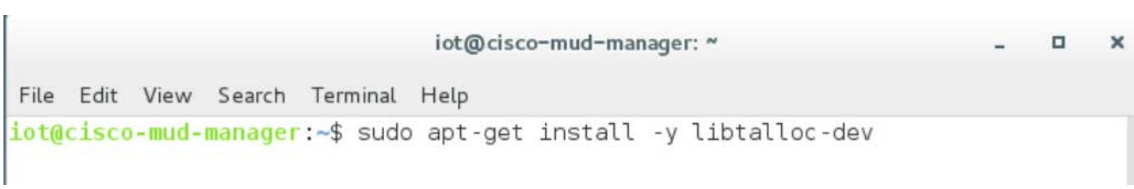
```
                }
        ],
        "DNSMapping" : {
                "www.osmud.org" : "198.71.233.87",
                "www.mqttbroker.com" : "192.168.4.6",
                "us.dlink.com" : "54.187.217.118",
                "www.nossl.net": "40.68.201.127",
                "www.trytechy.com" : "99.84.104.21"
        },

        "DNSMapping_v6" : {
                "www.mqttbroker.com" : "2610:20:60CE:630:B000::6",
                "www.updateserver.com" : "2610:20:60CE:630:B000::7",
                "www.dominiontea.com": "2a03:2880:f10c:83:face:b00c:0:25de"
        },
        "ControllerMapping" : {
                "https://www.google.com" : "192.168.10.104",
                "http://lightcontroller.example2.com": "192.168.4.77",
                "http://lightcontroller.example.com": "192.168.4.78"
        },
        "ControllerMapping_v6" : {
                "https:/www.google.com" : "ffff:2343:4444:::",
                "http://lightcontroller.example2.com": "ffff:2343:4444:::",
                "http://lightcontroller.example.com": "ffff:2343:4444:::"

        },
        "DefaultACL" : ["permit tcp any eq 22 any","permit udp any eq 68 any eq
67","permit udp any any eq 53", "deny ip any any"],
        "DefaultACL_v6" : ["permit udp any any eq 53", "deny ipv6 any any"]

}
```

Details about the contents of the configuration file can be found at the link provided at the start of this section.

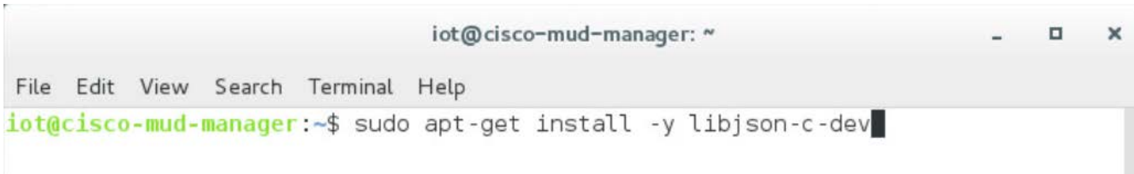
2.1.3.4 FreeRADIUS Installation

1. Install the dependencies for FreeRADIUS:

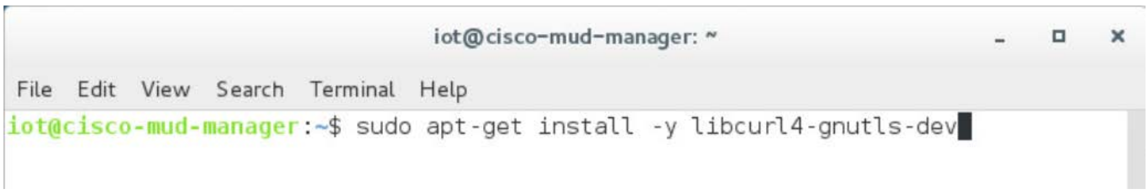
    a. `sudo apt-get install -y libtalloc-dev`



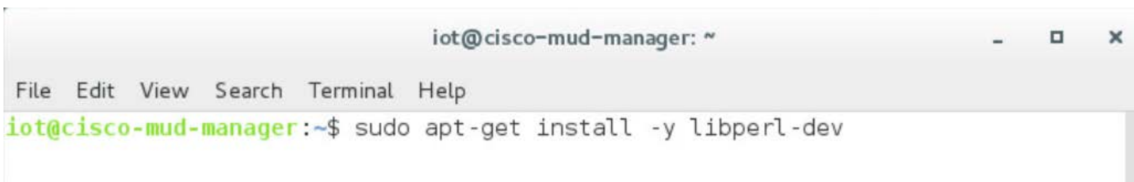
    b. `sudo apt-get install -y libjson-c-dev`



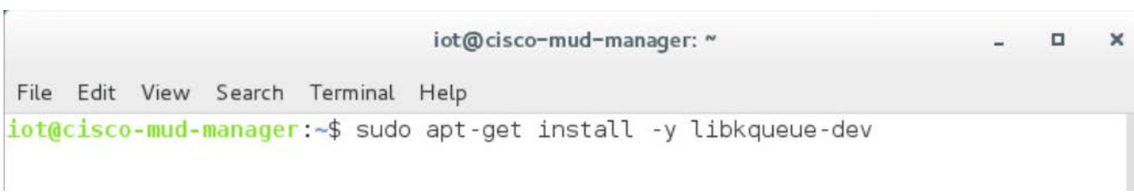
c. `sudo apt-get install -y libcurl4-gnutls-dev`



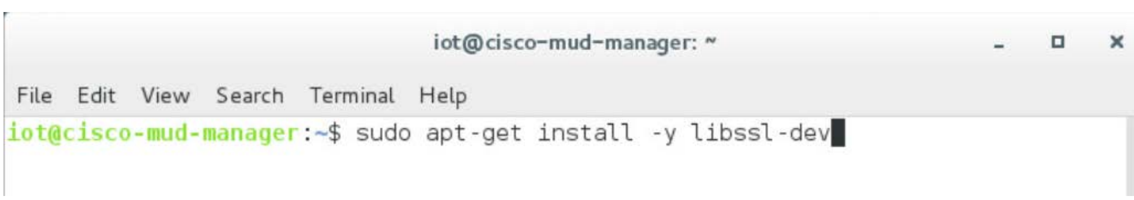
d. `sudo apt-get install -y libperl-dev`



e. `sudo apt-get install -y libkqueue-dev`

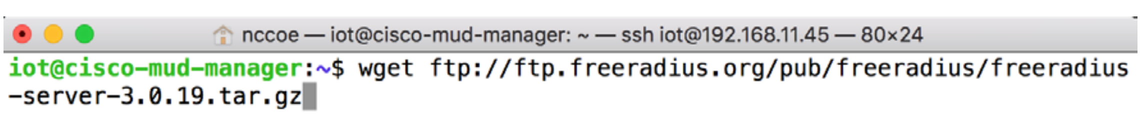


f. `sudo apt-get install -y libssl-dev`



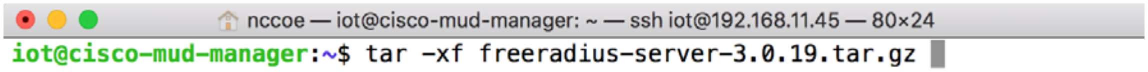
2. Download the source by entering the following command (Note: Version 3.0.19 and later are recommended):

   `wget` ftp://ftp.freeradius.org/pub/freeradius/freeradius-server-3.0.19.tar.gz



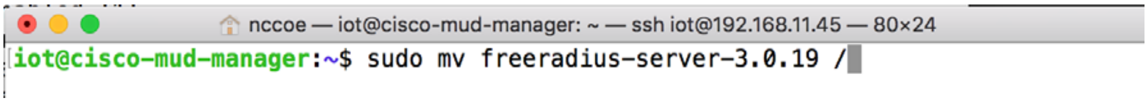
3. Untar the downloaded file by entering the following command:

```
tar -xf freeradius-server-3.0.19.tar.gz
```



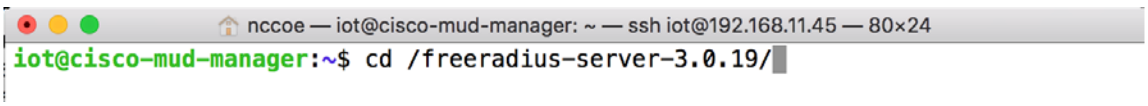
4. Move the FreeRADIUS directory to the root directory:

```
sudo mv freeradius-server-3.0.19/ /
```



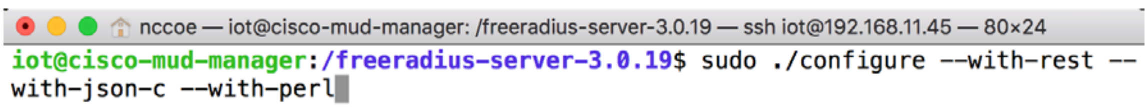
5. Change to the FreeRADIUS directory:

```
cd /freeradius-server-3.0.19/
```

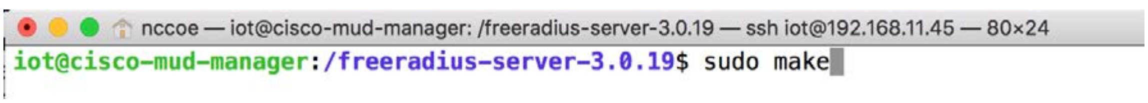


6. Make and install the source by entering the following:

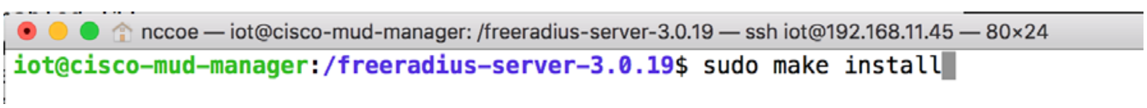
   a. `sudo ./configure --with-rest --with-json-c --with-perl`



   b. `sudo make`



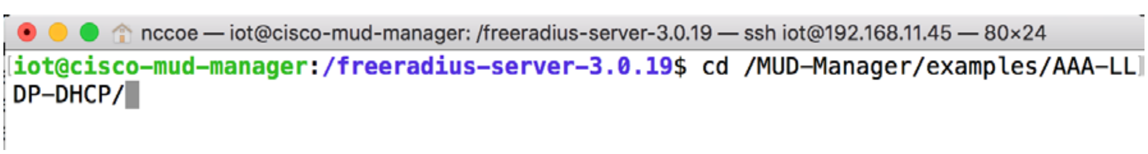
   c. `sudo make install`



2.1.3.5 FreeRADIUS Configuration

1. Change to the FreeRADIUS subdirectory in the MUD manager directory:

```
cd /MUD-Manager/examples/AAA-LLDP-DHCP/
```



2. Run the setup script:

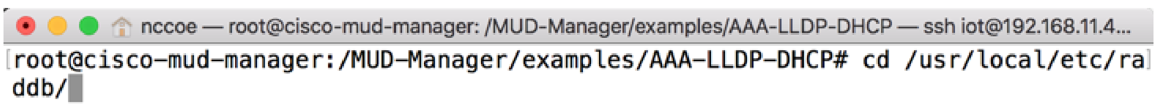
```
sudo ./FR-setup.sh
```

3. Enter the following command to log in as root:

```
sudo su
```

4. Change to the RADIUS directory:

```
cd /usr/local/etc/raddb/
```



5. Open the *clients.conf* file:

```
vim clients.conf
```

6. Add the network access server (NAS) as an authorized client in the configuration file on the server by adding an entry for the NAS in the *client.conf* file that is opened (Note: Replace the IP address below with the IP address of the NAS, and insert the "secret" configured on the NAS to talk to the RADIUS servers):

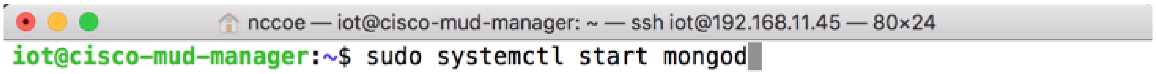
```
client 192.168.10.2 {
      ipaddr = 192.168.10.2
      secret = cisco
   }
```

7. Save and close the file.

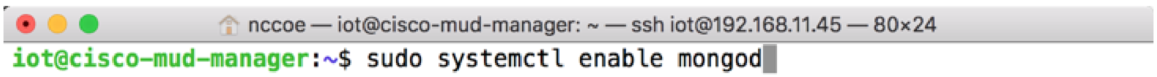
2.1.3.6 Start MUD Manager and FreeRADIUS Server

1. Start and enable the database by executing the following commands:

```
sudo systemctl start mongod
```

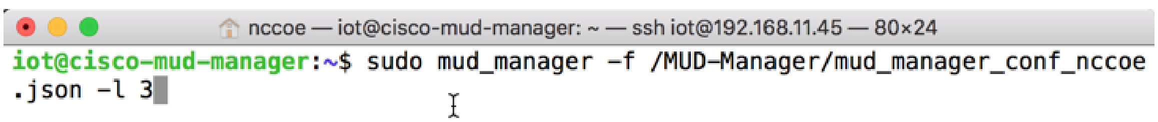



```
sudo systemctl enable mongod
```

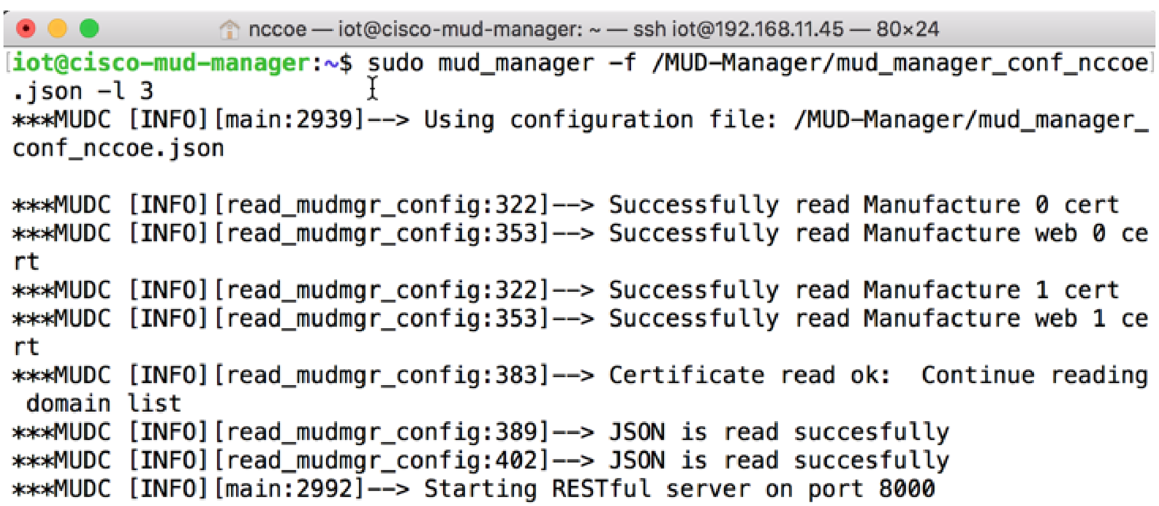


2. Start the MUD manager in the foreground with logging enabled by entering the following command:

```
sudo mud_manager -f /MUD-Manager/mud_manager_conf_nccoe.json -l 3
```



The following output should appear if the service started successfully:



3. Start the FreeRADIUS service in the foreground with logging enabled by entering the following command:

```
sudo radiusd -Xxx
```

At this point all the processes required to support MUD are running on the server side, and the next step is to configure the Cisco Catalyst switch. Once the switch configuration detailed in the Cisco Switch–Catalyst 3850-S setup section is completed, any DHCP activity on the network should appear in the output of the FreeRADIUS and MUD manager logs.

2.2 MUD File Server

2.2.1 MUD File Server Overview

For this build, the NCCoE built a MUD file server hosted within the lab infrastructure. This file server signs and stores the MUD files along with their corresponding signature files for the MUD-capable IoT devices used in the build. The MUD file server is also responsible for serving the MUD file and the corresponding signature file upon request from the MUD manager.

2.2.2 Configuration Overview

The following subsections document the software and network configurations for the MUD file server.

2.2.2.1 Network Configuration

This server was hosted in the NCCoE's virtual environment, functioning as a cloud service. Its IP address was statically assigned.

2.2.2.2 Software Configuration

For this build, the server ran on the CentOS 7 operating system. The MUD files and signatures were hosted by an Apache web server and configured to use secure sockets layer/Transport Layer Security (SSL/TLS) encryption.

2.2.2.3 Hardware Configuration

The MUD file server was hosted in the NCCoE's virtual environment, functioning as a cloud service.

2.2.3 Setup

The following subsections describe the process for configuring the MUD file server.

2.2.3.1 Apache Web Server

The Apache web server was set up by using the official Apache documentation at https://httpd.apache.org/docs/current/install.html. After that, SSL/TLS encryption was set up by using the digital certificate and key obtained from DigiCert. This was set up by using the official Apache documentation, found at https://httpd.apache.org/docs/current/ssl/ssl_howto.html.

2.2.3.2 MUD File Creation and Signing

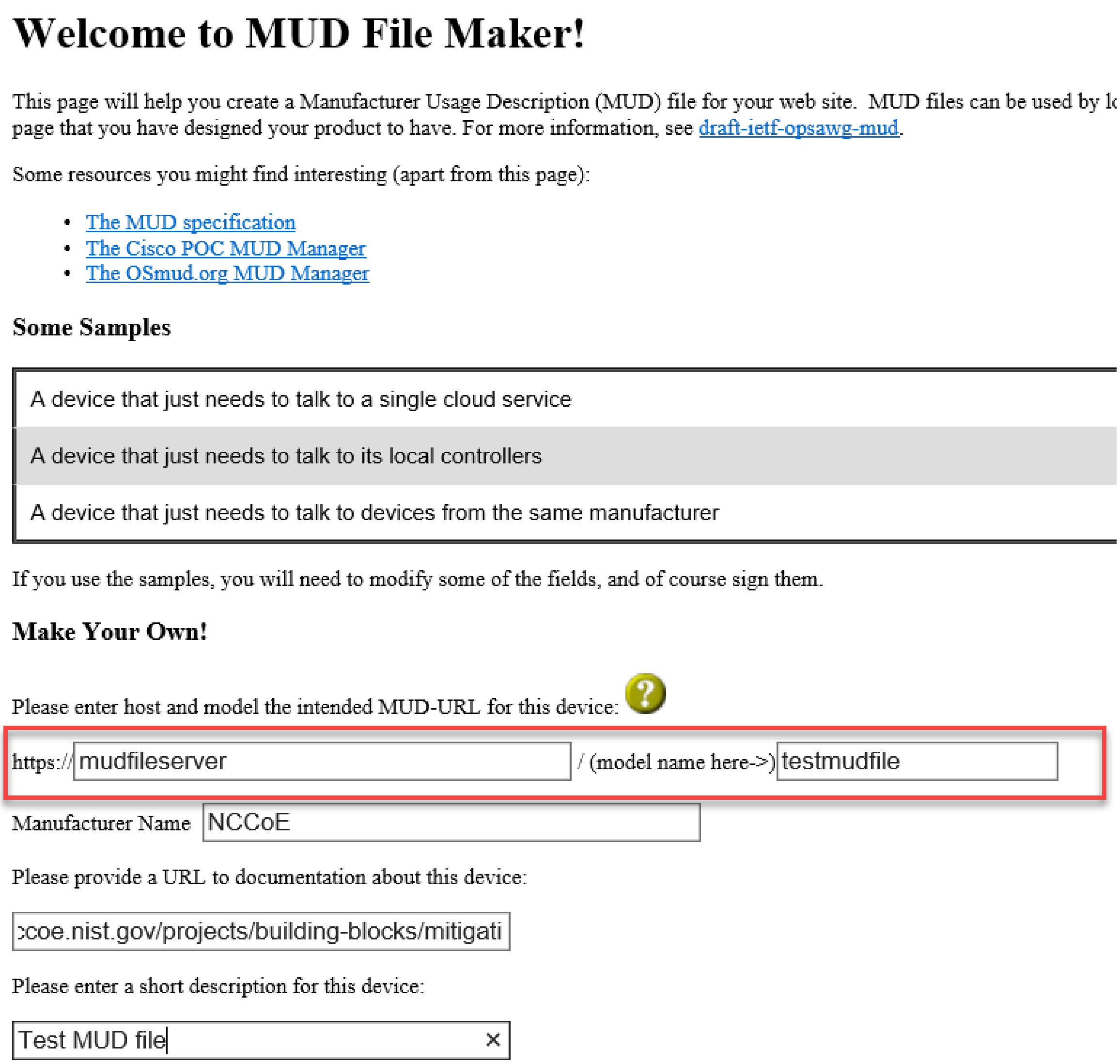
This section details creating and signing a MUD file on the MUD file server. The MUD specification does not mandate that this signing process be performed on the MUD file server itself.

2.2.3.2.1 MUD File Creation

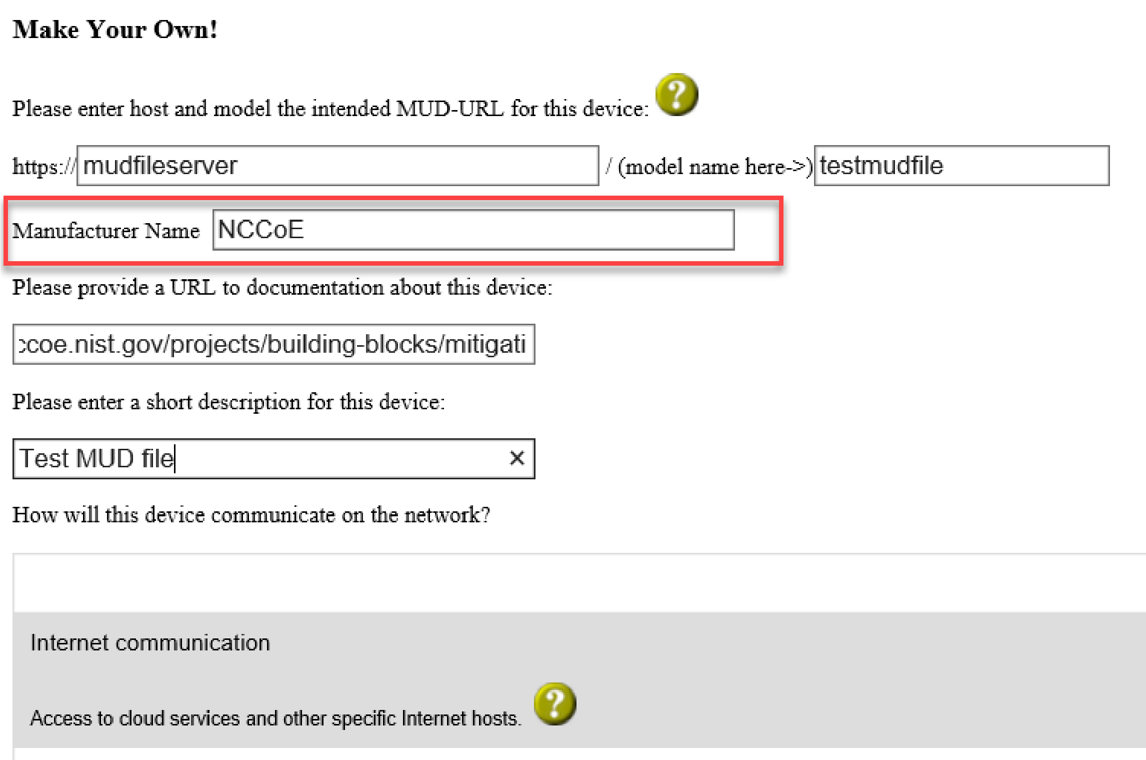
An online tool called MUD Maker was used to build MUD files. Once the permitted communications have been defined for the IoT device, proceed to www.mudmaker.org to leverage the online tool. There is also a list of sample MUD files on the site, which can be used as a reference. Upon navigating to www.mudmaker.org, complete the following steps to create a MUD file:

1. Specify the host that will be serving the MUD file and the model name of the device in the appropriate input fields, which are outlined in red in the screenshot below (Note: This will result in the MUD URL for this device):

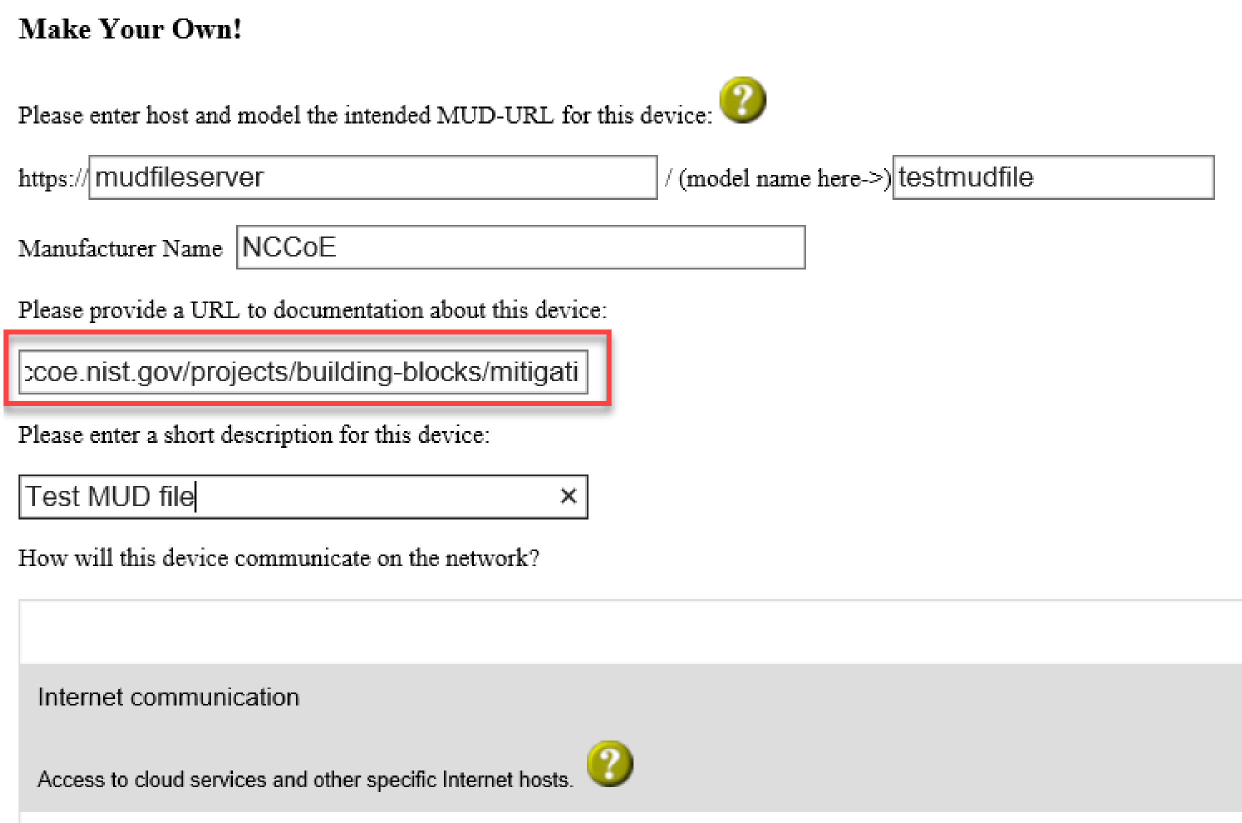
   Sample input: mudfileserver, testmudfile



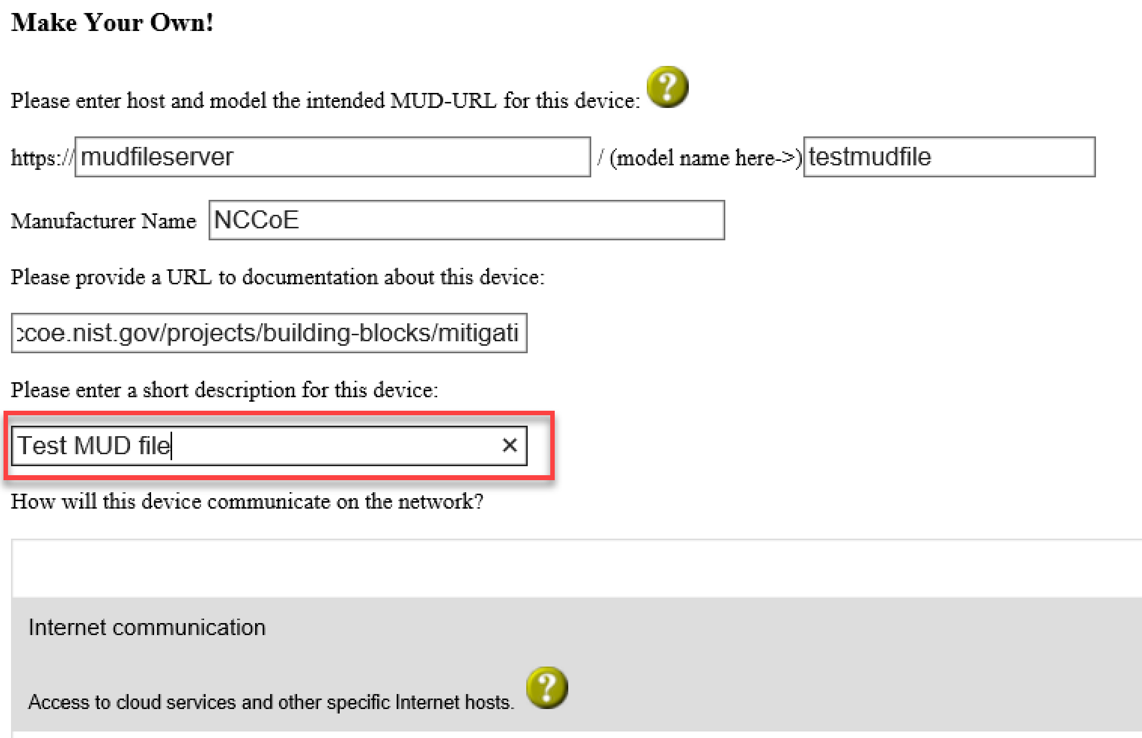
2. Specify the Manufacturer Name of the device in the appropriate input field, which is outlined in red in the screenshot below:


Make Your Own!
Please enter host and model the intended MUD-URL for this device:
https:// mudfileserver / (model name here->) testmudfile
Manufacturer Name NCCoE
Please provide a URL to documentation about this device:
:coe.nist.gov/projects/building-blocks/mitigati
Please enter a short description for this device:
Test MUD file
How will this device communicate on the network?
Internet communication
Access to cloud services and other specific Internet hosts.

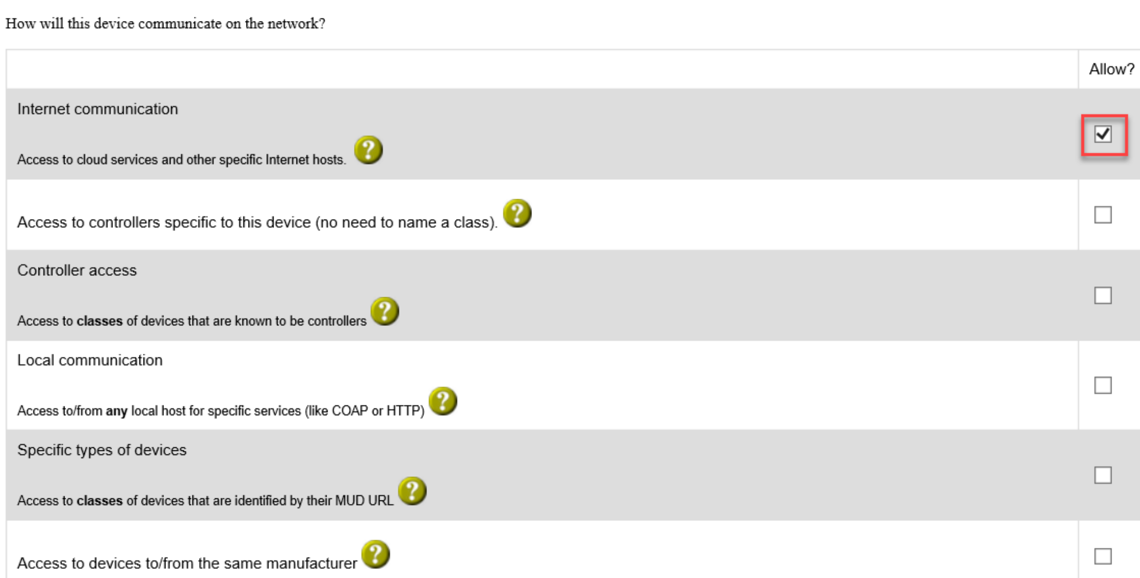
3. Include a URL to provide documentation about this device in the appropriate input field, which is outlined in red in the screenshot below:



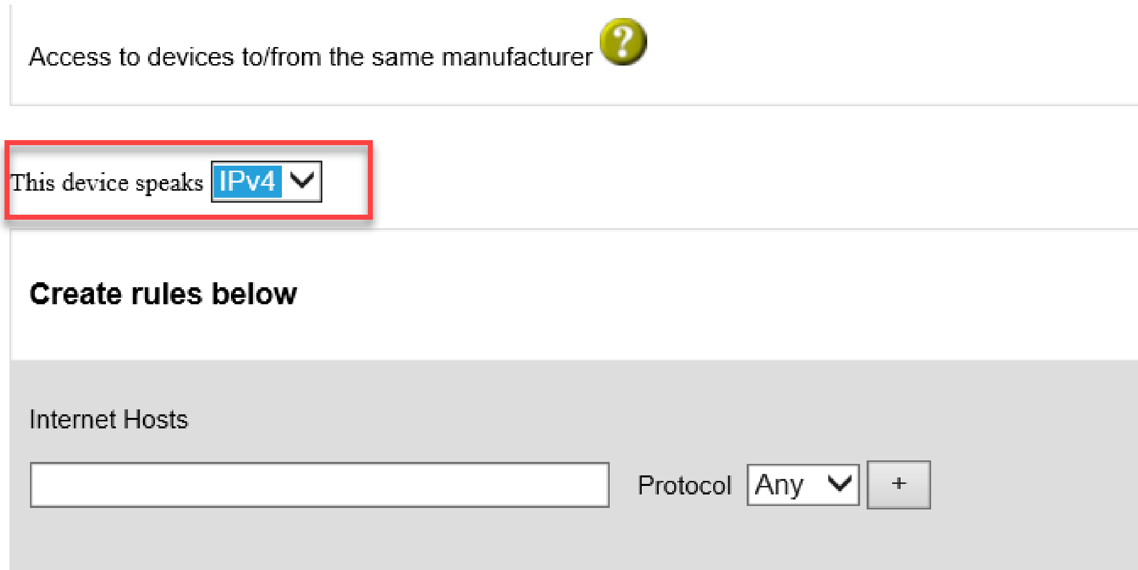
4. Include a short description of the device in the appropriate input field, which is outlined in red in the screenshot below:



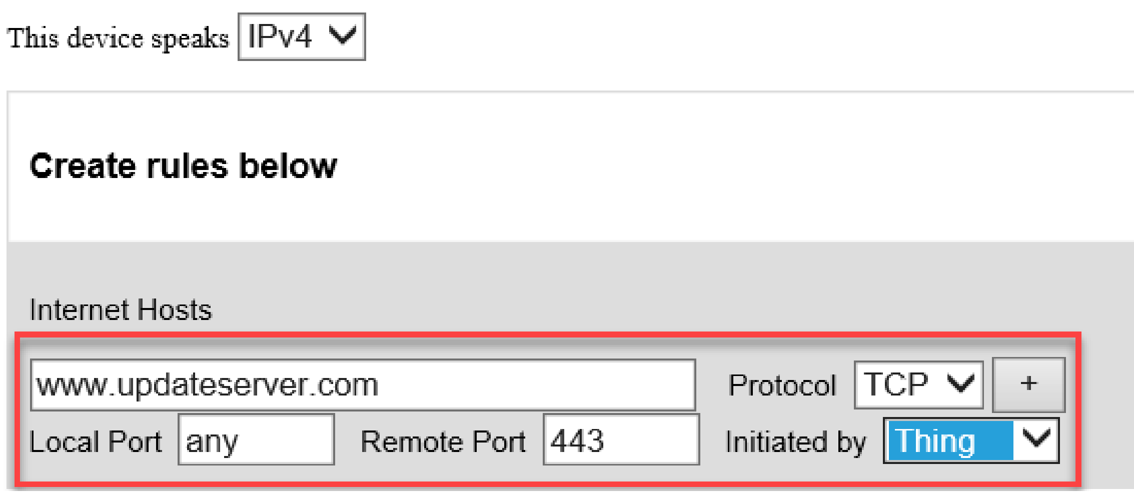
5. Check the boxes for the types of network communication that are allowed for the device:



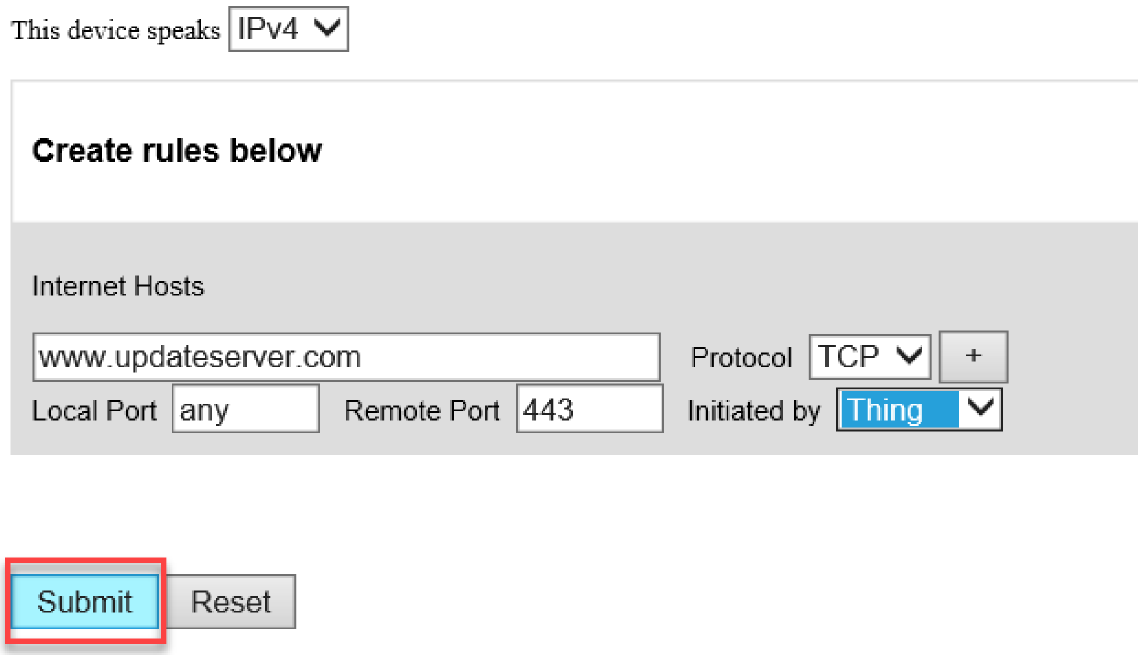
6. Specify the internet protocol version that the device leverages:



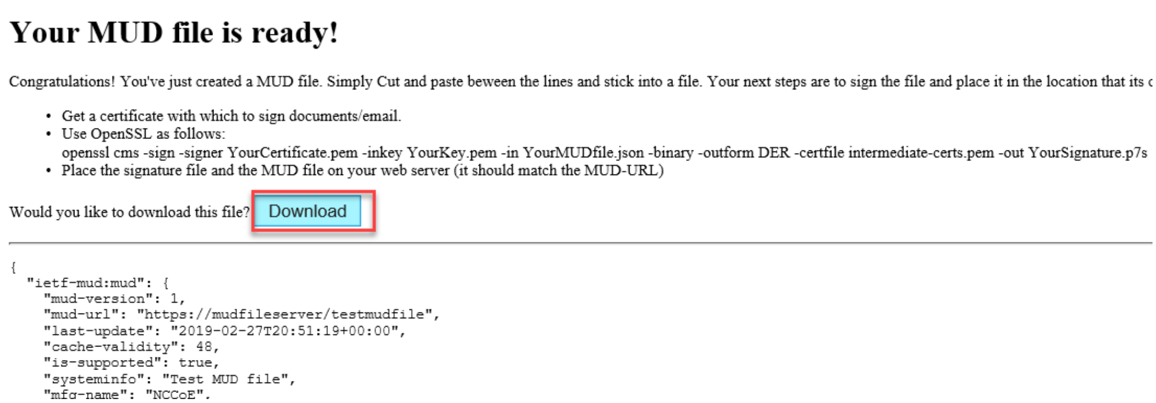
7. Specify values for the fields (Internet Hosts, Protocol, Local Port, Remote Port, and Initiated by) that describe the communications that will be permitted for the device:



8. Click **Submit** to generate the MUD file:



9. Once completed, the page will redirect to the following page that outputs the MUD file on the screen. Click **Download** to download the MUD file, which is a .JSON file:



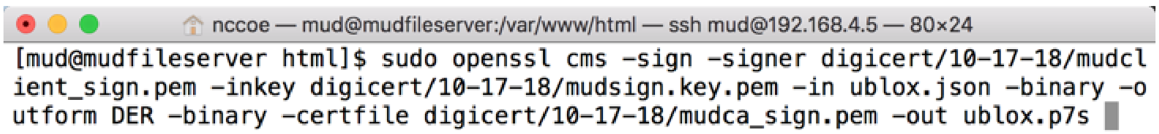
10. Click **Save** to store a copy of the MUD file:

2.2.3.2.2 MUD File Signature Creation and Verification

In this build, OpenSSL is used to sign and verify MUD files. This example uses the MUD file created in the previous section, which is named *ublox.json*; the Signing Certificate; the Private Key for the Signing Certificate; the Intermediate Certificate for the Signing Certificate; and the Certificate of the Trusted Root Certificate Authority (CA) for the Signing Certificate.

1. Sign the MUD file by using the following command:

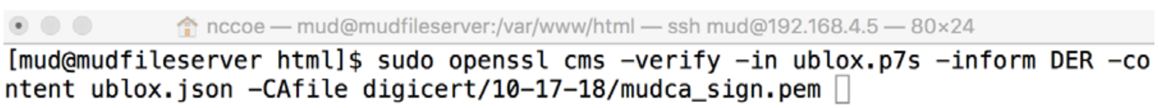
```
sudo openssl cms -sign -signer <Signing Certificate> -inkey <Private Key for
Signing Certificate> -in <Name of MUD File> -binary -outform DER -binary -
certfile <Intermediate Certificate for Signing Certificate> -out <Name of MUD
File without the .json file extension>.p7s
```



   This will create a signature file for the MUD file that has the same name as the MUD file but ends with the .p7s file extension, i.e., in our case *ublox.p7s*.

2. Manually verify the MUD file signature by using the following command:

```
sudo openssl cms -verify -in <Name of MUD File>.p7s -inform DER -content <Name
of MUD File>.json -CAfile <Certificate of Trusted Root Certificate Authority
for Signing Certificate>
```



If a valid file signature was created successfully, a corresponding message should appear. Both the MUD file and MUD file signature should be placed on the MUD file server in the Apache server directory.

2.3 Cisco Switch–Catalyst 3850-S

2.3.1 Cisco 3850-S Catalyst Switch Overview

The switch used in this build is an enterprise-class, layer 3 switch. It is a Cisco Catalyst 3850-S that had been modified to support MUD functionality as a proof-of-concept implementation. In addition to providing DHCP services, the switch acts as a broker for connected IoT devices for authentication, authorization, and accounting through a FreeRADIUS server. The Link Layer Discovery Protocol (LLDP) is enabled on ports that MUD-capable devices are plugged into to help facilitate recognition of connected IoT device features, capabilities, and neighbor relationships at layer 2. Additionally, an access session policy is configured on the switch to enable port control for multihost authentication and port monitoring. The combined effect of these switch configurations is a dynamic access list, which has been generated by the MUD manager, being active on the switch to permit or deny access to and from MUD-capable IoT devices.

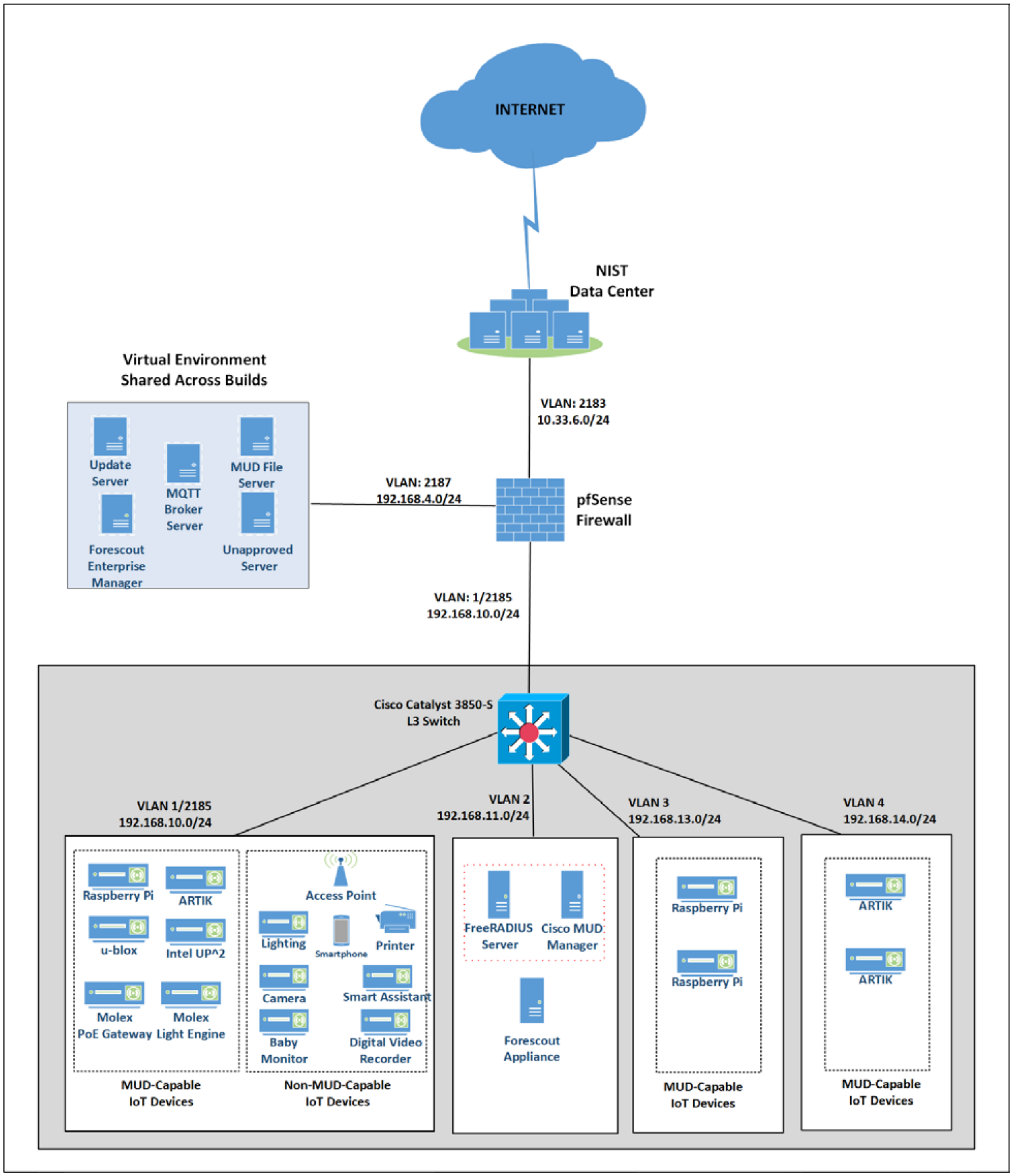
2.3.2 Configuration Overview

The following subsections document the network, software, and hardware configurations for the Cisco Catalyst 3850-S switch.

2.3.2.1 Network Configuration

This section describes how to configure the required Cisco Catalyst 3850-S switch to support the build. A special image for the Catalyst 3850-S was provided by Cisco to support MUD-specific functionality. In our build, the switch is integrated with a DHCP server and a FreeRADIUS server, which together support delivery of the MUD URL to the MUD manager via either DHCP or LLDP. The MUD manager is also able to generate and send a dynamic access list to the switch, via the RADIUS server, to permit or deny access to and from the IoT devices. In addition to hosting directly connected IoT devices on VLANs 1, 3, and 4, the switch hosts both the MUD manager and the FreeRADIUS servers on VLAN 2. As illustrated in Figure 2-1, each locally configured VLAN is protected by a firewall that connects the lab environment to the NIST data center, which provides internet access for all connected devices.

850 **Figure 2-1 Physical Architecture–Build 1**



2.3.2.2 Software Configuration

The prototype, MUD-capable Cisco 3850-S used in this build is running internetwork operating system (IOS) version 16.09.02.

2.3.2.3 Hardware Configuration

The Catalyst 3850-S switch configured in the lab consists of 24 one-gigabit Ethernet ports with two optional 10-gigabit Ethernet uplink ports. A customized version of Cat-OS is installed on the switch. The versions of the OS are as follows:

- Cat3k_caa-guestshell.16
- Cat3k_caa-rpbase.16.06
- Cat3k_caa-rpcore.16.06
- Cat3k_caa-srdriver.16.06.0
- Cat3k_caa-webui.16.06.0

2.3.3 Setup

Table 2-1 lists the Cisco 3850-S switch running configuration used for the lab environment. In addition to the IOS version and a few generic configuration items, configuration items specifically relating to integration with the MUD manager and IoT devices are highlighted in bold fonts; these include DHCP, LLDP, AAA, RADIUS, and policies regarding access session. Table 2-1 also provides a description of each configuration item for ease of understanding.

Table 2-1 Cisco 3850-S Switch Running Configuration

Configuration Item	Description
version 16.9 no service pad service timestamps debug datetime msec service timestamps log datetime msec service call-home no platform punt-keepalive disable-kernel-core ! hostname Build1 !	general overview of configuration information needed to configure AAA to use RADIUS and configure the RADIUS server itself. Note that the FreeRADIUS and AAA passwords must match.
aaa new-model **!**	enables AAA
aaa authentication dot1x default group radius	creates an 802.1X AAA authentication method list

Configuration Item	Description
aaa authorization network default group radius	configures network authorization via RADIUS, including network-related services such as VLAN assignment
aaa accounting identity default start-stop group radius	enables accounting method list for session-aware networking subscriber services
aaa accounting network default start-stop group radius !	enables accounting for all network-related service requests
aaa server radius dynamic-author **client 192.168.11.45 server-key cisco** **server-key cisco** ! aaa session-id common	enables dynamic authorization local server configuration mode and specifies a RADIUS client/key from which a device accepts change of authorization (CoA) and disconnect requests
radius server AAA **address ipv4 192.168.11.45 auth-port 1812**	enables AAA server from the list of multiple AAA servers configured
acct-port 1813 **key cisco**	uses the IP address and ports on which the FreeRADIUS server is listening
ip routing !	
ip dhcp excluded-address 192.168.10.1 192.168.10.100 **!**	DHCP server configuration to exclude selected addresses from pool
ip dhcp pool NCCOE-V3 **network 192.168.13.0 255.255.255.0** **default-router 192.168.13.1** **dns-server 8.8.8.8** **lease 0 12** !	DHCP server configuration to assign IP address to devices on VLAN 3
ip dhcp pool NCCOE-V4 **network 192.168.14.0 255.255.255.0** **default-router 192.168.14.1** **dns-server 8.8.8.8** **!**	DHCP server configuration to assign IP address to devices on VLAN 4
ip dhcp pool NCCOE **network 192.168.10.0 255.255.255.0** **default-router 192.168.10.2** **dns-server 8.8.8.8** **lease 0 12** !	DHCP server configuration to assign IP address to devices on VLAN 1
ip dhcp snooping **ip dhcp snooping vlan 1,3**	enables DHCP snooping globally

Configuration Item	Description
!	specifically enables DHCP snooping on VLANs 1 and 3
access-session attributes filter-list list mudtest **lldp** **dhcp** **access-session accounting attributes filter-spec include list mudtest** **access-session monitor** !	configures access-session attributes to cause LLDP Time Length Values (including the MUD URL) to be forwarded in an accounting message to the AAA server
dot1x logging verbose	global configuration command to filter 802.1x authentication verbose messages
ldp run **!**	enables LLDP, a discovery protocol that runs over layer 2 (the data link layer) to gather information on non-Cisco-manufactured devices
policy-map type control subscriber mud-mab-test **event session-started match-all** **10 class always do-until-failure** **10 authenticate using mab** **!**	configures identity control policies that define the actions that session-aware networking takes in response to specified conditions and subscriber events
template mud-mab-test **switchport mode access** **mab** **access-session port-control auto** **service-policy type control subscriber mud-mab-test** !	enables policy-map (mud-mab-test) and template to cause media access control (MAC) address bypass (MAB) to happen dynamically applies an interface template to a target sets the authorization state of a port. The default value is force-authorized. applies the above previously configured control policy called mud-mab-test
interface GigabitEthernet1/0/13 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device
interface GigabitEthernet1/0/14 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device
interface GigabitEthernet1/0/15 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device

Configuration Item	Description
interface GigabitEthernet1/0/16 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device
interface GigabitEthernet1/0/17 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device
interface GigabitEthernet1/0/18 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device
interface GigabitEthernet1/0/19 **source template mud-mab-test** **!**	statically applies an interface template to a target, i.e., an IoT device
interface GigabitEthernet1/0/20 **source template mud-mab-test**	statically applies an interface template to a target, i.e., an IoT device
interface Vlan1 **ip address 192.168.10.2 255.255.255.0** !	configure and address VLAN1 interface for inter-VLAN routing
interface Vlan2 **ip address 192.168.11.1 255.255.255.0** !	configure and address VLAN2 interface for inter-VLAN routing
interface Vlan3 **ip address 192.168.13.1 255.255.255.0** !	configure and address VLAN3 interface for inter-VLAN routing
interface Vlan4 **ip address 192.168.14.1 255.255.255.0** !	configure and address VLAN4 interface for inter-VLAN routing
interface Vlan5 **ip address 192.168.15.1 255.255.255.0** !	configure and address VLAN5 interface for inter-VLAN routing
! ip default-gateway 192.168.10.1 ip forward-protocol nd ip http server ip http authentication local ip http secure-server ip route 0.0.0.0 0.0.0.0 192.168.10.1 ip route 192.168.12.0 255.255.255.0 192.168.5.1 !	

2.4 DigiCert Certificates

2.4.1 DigiCert CertCentral® Overview

DigiCert's CertCentral® web-based platform allows provisioning and management of publicly trusted X.509 certificates for a variety of purposes. After establishing an account, clients can log in, request, renew, and revoke certificates by using only a browser. For this build, two certificates were provisioned: a private TLS certificate for the MUD file server to support the https connection from the MUD manager to the MUD file server, and a Premium Certificate for signing the MUD files.

2.4.2 Configuration Overview

This section typically documents the network, software, and hardware configurations, but that is not necessary for this component.

2.4.3 Setup

DigiCert allows certificates to be requested through its web-based platform, CertCentral. A user account is needed to access CertCentral. For details on creating a user account and setting up an account, follow the steps described here: https://docs.digicert.com/get-started/

2.4.3.1 TLS Certificate

For this build, we leveraged DigiCert's private TLS certificate because the MUD file server is hosted internally. This certificate supports https connections to the MUD file server, which are required by the MUD manager. Additional information about the TLS certificates offered by DigiCert can be found at https://www.digicert.com/security-certificate-support/.

For instructions on how to order a TLS certificate, proceed to the DigiCert documentation found here, and follow the process for the specific TLS certificate being requested: https://docs.digicert.com/manage-certificates/order-your-ssltls-certificates/

Once requested, integrate the certificate onto the MUD file server as described in Section 2.2.3.1.

2.4.3.2 Premium Certificate

To sign MUD files according to the MUD specification, a client certificate is required. For this implementation, we leveraged DigiCert's Premium Certificate to sign MUD files. This certificate supports signing or encrypting Secure/Multipurpose Internet Mail Extensions messages, which is required by the specification.

For detailed instructions on how to request and implement a Premium Certificate, proceed to the DigiCert documentation found here: https://docs.digicert.com/manage-certificates/client-certificates-guide/.

Once requested, sign MUD files as described in Section 2.2.3.2.2.

2.5 IoT Devices

2.5.1 Molex PoE Gateway and Light Engine

This section provides configuration details of the MUD-capable Molex PoE Gateway and Light Engine used in the build. This component emits a MUD URL that uses LLDP.

2.5.1.1 Configuration Overview

The Molex PoE Gateway runs firmware created and provided by Molex. This firmware was modified by Molex to emit a MUD URL that uses an LLDP message.

2.5.1.1.1 Network Configuration

The Molex PoE Gateway is connected to the network over a wired Ethernet connection. The IP address is assigned dynamically by using DHCP.

2.5.1.1.2 Software Configuration

For this build, the Molex PoE Gateway is configured with Molex's PoE Gateway firmware, version 1.6.1.8.4.

2.5.1.1.3 Hardware Configuration

The Molex PoE Gateway used in this build is model number 180993-0001, dated March 2017.

2.5.1.2 Setup

The Molex PoE Gateway is controlled via the Constrained Application Protocol (CoAP), and CoAP commands were used to ensure that device functionality was maintained during the MUD process.

2.5.1.2.1 DHCP Client Configuration

The device uses the default DHCP client included in the Molex PoE Gateway firmware.

2.5.2 IoT Development Kits–Linux Based

This section provides configuration details for the Linux-based IoT development kits used in the build, which emit MUD URLs by using DHCP. It also provides information regarding a basic IoT application used to test the MUD process.

2.5.2.1 Configuration Overview

The devkits run various flavors of Linux-based operating systems and are configured to emit a MUD URL during a typical DHCP transaction. They also run a Python script that allows the devkits to receive and process commands by using the MQTT protocol, which can be sent to peripherals connected to the devkits.

2.5.2.1.1 Network Configuration

The devkits are connected to the network over a wired Ethernet connection. The IP address is assigned dynamically by using DHCP.

2.5.2.1.2 Software Configuration

For this build, the Raspberry Pi is configured on Raspbian 9, the Samsung ARTIK 520 is configured on Fedora 24, and the Intel UP Squared Grove is configured on Ubuntu 16.04 LTS. The devkits also utilized dhclient as the default DHCP client. This DHCP client is installed natively on many Linux distributions and can be installed using a preferred package manager if not currently present.

2.5.2.1.3 Hardware Configuration

The hardware used for these devkits included the Raspberry Pi 3 Model B, Samsung ARTIK 520, and Intel UP Squared Grove.

2.5.2.2 Setup

The following subsection describes setting up the devkits to send a MUD URL during the DHCP transaction and to act as a connected device by leveraging an MQTT broker server (we describe setting up the MQTT broker server in Section 2.8).

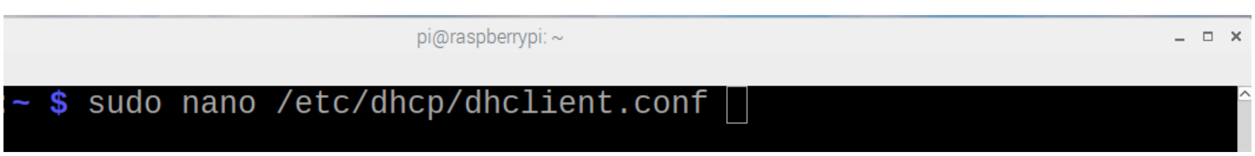
2.5.2.2.1 DHCP Client Configuration

We leveraged dhclient as the default DHCP client for these devices due to the availability of the DHCP client on different Linux platforms and the ease of emitting MUD URLs via DHCP.

To set up the dhclient configuration:

1. Open a terminal on the device.
2. Ensure that any other conflicting DHCP clients are disabled or removed.
3. Install the dhclient package (if needed).
4. Edit the *dhclient.conf* file by entering the following command:

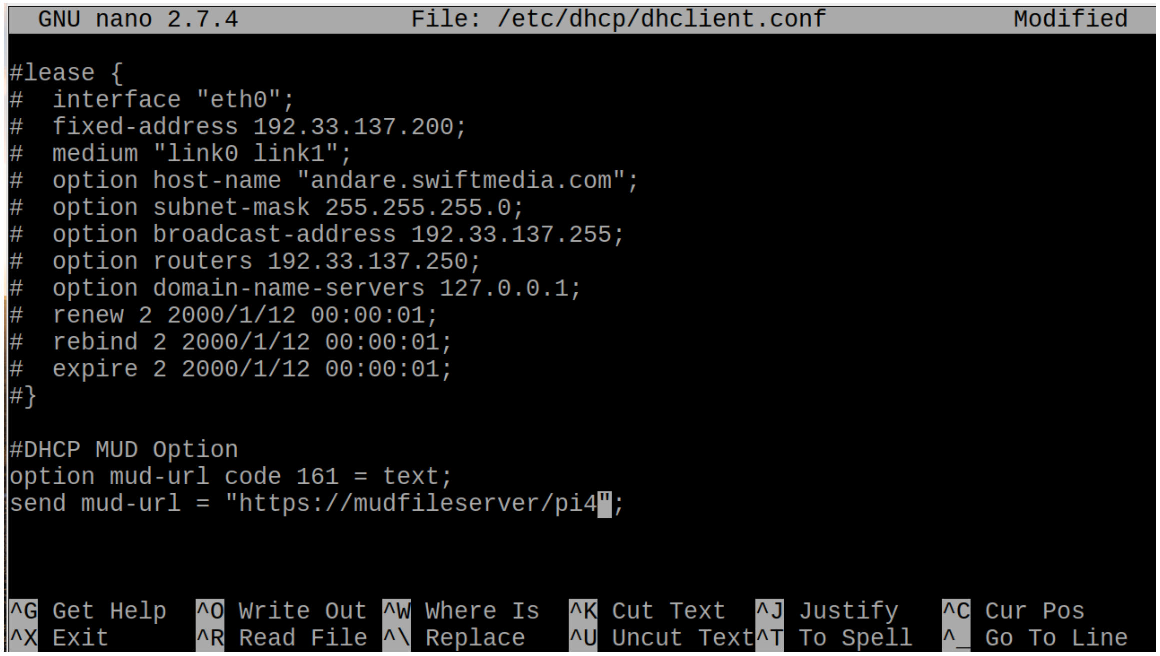
   ```
   sudo nano /etc/dhcp/dhclient.conf
   ```

5. Add the following lines:

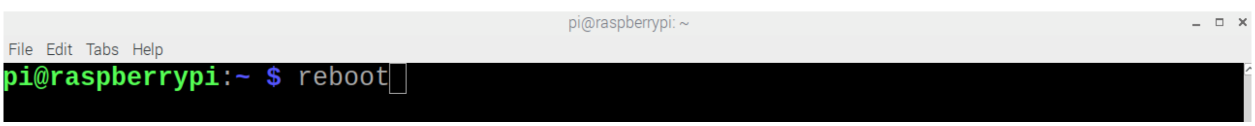
   ```
   option mud-url code 161 = text;
   ```

```
send mud-url = "<insert URL for MUD File here>";
```



6. Save and close the file.
7. Reboot the device:

```
Reboot
```



8. Open a terminal.
9. Execute the dhclient:

```
sudo dhclient -v
```

2.5.2.2.2 IoT Application for Testing

The following Python application was created by the NCCoE to enable the devkits to act as basic IoT devices:

```
#Program:                    IoTapp.
```

```
#Version:                 1.0
#Purpose:                 Provide IoT capabilities to devkit.
#Protocols:         MQTT.
#Functionality:     Allow remote control of LEDs on connected breadboard.

#Libraries
import paho.mqtt.client as mqttClient
import time
import RPi.GPIO as GPIO

#Global Variables
BrokerAddress = "192.168.1.87"   #IP address of Broker(Server), change as needed. Best
practice would be a registered domain name that can be queried for appropriate server
address.
BrokerPort = "1883"        #Default port used by most MQTT Brokers. Would be 1883 if
using Transport Encryption with TLS.
ConnectionStatus = "Disconnected"#Status of connection to Broker. Should be either
"Connected" or "Disconnected".
LED = 26

#Supporting Functions
def on_connect(client, userdata, flags, rc):   #Function for connection status to
Broker.
	if rc == 0:
		ConnectionStatus = "Connected to Broker!"
		print(ConnectionStatus)
	else:
		ConnectionStatus = "Connection Failed!"
		print(ConnectionStatus)

def on_message(client, userdata, msg):         #Function for parsing message data.
	if "ON" in msg.payload:
		print("ON!")
		GPIO.output(LED, 1)

	if "OFF" in msg.payload:
		print("OFF!")
		GPIO.output(LED, 0)

def  MQTTapp():
	client = mqttClient.Client()     #New instance.
	client.on_connect = on_connect
	client.on_message = on_message
	client.connect(BrokerAddress, BrokerPort)
	client.loop_start()
	client.subscribe("test")
	try:
		while True:
			time.sleep(1)
	except KeyboardInterrupt:
		print("8")
		client.disconnect()
```

```
                client.loop_stop()

#Main Function
def main():

        GPIO.setmode(GPIO.BCM)
        GPIO.setup(LED, GPIO.OUT)

        print("Main function has been executed!")
        MQTTapp()

if __name__ == "__main__":
        main()
```

2.5.3 IoT Development Kit–u-blox C027-G35

This section details configuration of a u-blox C027-G35, which emits a MUD URL by using DHCP, and a basic IoT application used to test MUD rules.

2.5.3.1 Configuration Overview

This devkit runs the Arm Mbed-OS and is configured to emit a MUD URL during a typical DHCP transaction. It also runs a basic IoT application to test MUD rules.

2.5.3.1.1 Network Configuration

The u-blox C027-G35 is connected to the network over a wired Ethernet connection. The IP address is assigned dynamically by using DHCP.

2.5.3.1.2 Software Configuration

For this build, the u-blox C027-G35 was configured on the Mbed-OS 5.10.4 operating system.

2.5.3.1.3 Hardware Configuration

The hardware used for this devkit is the u-blox C027-G35.

2.5.3.2 Setup

The following subsection describes setting up the u-blox C027-G35 to send a MUD URL in the DHCP transaction and to act as a connected device by establishing network connections to the update server and other destinations.

2.5.3.2.1 DHCP Client Configuration

To add MUD functionality to the Mbed-OS DHCP client, the following two files inside Mbed-OS require modification:

- *mbed-os/features/lwipstack/lwip/src/include/lwip/prot/dhcp.h*
    - **NOT** *mbed-os/features/lwipstack/lwip/src/include/lwip/dhcp.h*

- *mbed-os/features/lwipstack/lwip/src/core/ipv4/lwip_dhcp.c*

Changes to include/lwip/prot/dhcp.h:

1. Add the following line below the greatest DCHP option number (67) on line 170:

```
#define DHCP_OPTION_MUD_URL_V4    161 /*MUD: RFC-ietf-opsawg-mud-25 draft-ietf-opsawg-mud-08,
Manufacturer Usage Description*/
```

Changes to core/ipv4/lwip_dhcp.c:

1. Change within container around line 141:

   To enum dhcp_option_idx (at line 141) before the first #if, add

```
DHCP_OPTION_IDX_MUD_URL_V4, /*MUD: DHCP MUD URL Option*/
```

   It should now look like the screenshot below:

```
enum dhcp_option_idx {
  DHCP_OPTION_IDX_OVERLOAD = 0,
  DHCP_OPTION_IDX_MSG_TYPE,
  DHCP_OPTION_IDX_SERVER_ID,
  DHCP_OPTION_IDX_LEASE_TIME,
  DHCP_OPTION_IDX_T1,
  DHCP_OPTION_IDX_T2,
  DHCP_OPTION_IDX_SUBNET_MASK,
  DHCP_OPTION_IDX_ROUTER,
  DHCP_OPTION_IDX_MUD_URL_V4, /*MUD: DHCP MUD URL Option*/
#if LWIP_DHCP_PROVIDE_DNS_SERVERS
  DHCP_OPTION_IDX_DNS_SERVER,
  DHCP_OPTION_IDX_DNS_SERVER_LAST = DHCP_OPTION_IDX_DNS_SERVER +
LWIP_DHCP_PROVIDE_DNS_SERVERS - 1,
#endif /* LWIP_DHCP_PROVIDE_DNS_SERVERS */
#if LWIP_DHCP_GET_NTP_SRV
  DHCP_OPTION_IDX_NTP_SERVER,
  DHCP_OPTION_IDX_NTP_SERVER_LAST = DHCP_OPTION_IDX_NTP_SERVER +
LWIP_DHCP_MAX_NTP_SERVERS - 1,
#endif /* LWIP_DHCP_GET_NTP_SRV */
  DHCP_OPTION_IDX_MAX
};
```

2. Change within the function around line 975:

a. To the list of local variables for `static err_t dhcp_discover(struct netif *netif)`, add the desired MUD URL (*www.example.com* used here):

```
char* mud_url = "https://www.example.com";    /*MUD: MUD URL*/
```

NOTE: The MUD URL must be less than 255 octets/bytes/characters long.

b. Within if (result == ERR_OK) after

```
dhcp_option(dhcp, DHCP_OPTION_PARAMETER_REQUEST_LIST,
LWIP_ARRAYSIZE(dhcp_discover_request_options));
   for (i = 0; i < LWIP_ARRAYSIZE(dhcp_discover_request_options); i++) {
     dhcp_option_byte(dhcp, dhcp_discover_request_options[i]);
   }
```

and before:

```
   dhcp_option_trailer(dhcp);
```

add:

```
   /*MUD: Begin - Add Option and URL to DISCOVER/REQUEST*/
#if (DHCP_DEBUG != LWIP_DBG_OFF)
   if (strlen(mud_url) > 255)
     LWIP_DEBUGF(DHCP_DEBUG | LWIP_DBG_TRACE, ("dhcp_discover: MUD URL is too large (>255)\n"));
#endif /* DHCP_DEBUG != LWIP_DBG_OFF */

   u8_t mud_url_len = (strlen(mud_url) < 255)? strlen(mud_url) : 255; //Ignores any URL greater than 255
bytes/octets
   dhcp_option(dhcp, DHCP_OPTION_MUD_URL_V4, mud_url_len);
   for (i = 0; i < mud_url_len; i++) {
     dhcp_option_byte(dhcp, mud_url[i]);
   }
   /*MUD: END - Add Option and URL to DISCOVER/REQUEST */
```

3. Change within the function around line 1486:

Within the following function:

```
static err_t
dhcp_parse_reply(struct dhcp *dhcp, struct pbuf *p)
```

Within switch(op) before default, add the following case (around line 1606):

```
case(DHCP_OPTION_MUD_URL_V4): /* MUD Testing */
 LWIP_ERROR("len == 0", len == 0, return ERR_VAL;);
 decode_idx = DHCP_OPTION_IDX_MUD_URL_V4;
 break;
```

4. Compile by using the following command:

```
mbed compile -m ublox_c027 -t gcc_arm
```

2.5.3.2.2 IoT Application for Testing

The following application was created by the NCCoE to enable the devkit to test the build as a MUD-capable device:

```
#include "mbed.h"
#include "EthernetInterface.h"

//DigitalOut led1(LED1);
PwmOut led2(LED2);
Serial pc(USBTX, USBRX);

float brightness = 0.0;

// Network interface
EthernetInterface net;

// Socket demo
int main() {
  int led1 = true;

  for (int i = 0; i < 4; i++) {

    led2 = (led1)? 0.5 : 0.0;

    led1 = !led1;
    wait(0.5);
  }

  for (int i = 0; i < 8; i++) {

    led2 = (led1)? 0.5 : 0.0;

    led1 = !led1;
    wait(0.25);
  }

  for (int i = 0; i < 8; i++) {

    led2 = (led1)? 0.5 : 0.0;

    led1 = !led1;
    wait(0.125);
```

```
  }
  TCPSocket socket;
  char sbuffer[] = "GET / HTTP/1.1\r\nHost: www.updateserver.com\r\n\r\n";
  char bbuffer[] = "GET / HTTP/1.1\r\nHost: www.unapprovedserver.com\r\n\r\n";
  int scount, bcount;
  char rbuffer[64];
  char brbuffer[64];
  int rcount, brcount;

  /* By default grab an IP address*/
  // Bring up the ethernet interface
  pc.printf("Ethernet socket example\r\n");
  net.connect();
  // Show the network address
  const char *ip = net.get_ip_address();
  pc.printf("IP address is: %s\r\n", ip ? ip : "No IP");
  socket.open(&net);
  /* End of default IP address */

  pc.printf("Press U to turn LED1 brightness up, D to turn it down, G to get IP, R to
release IP, H for HTTP request, B for blocked HTTP request\r\n");

  while(1) {
    char c = pc.getc();
    if((c == 'u') && (brightness < 0.5)) {
      brightness += 0.01;
      led2 = brightness;
    }
    if((c == 'd') && (brightness > 0.0)) {
      brightness -= 0.01;
      led2 = brightness;
    }
    if(c == 'g'){
      // Bring up the ethernet interface
      pc.printf("Sending DHCP Request...\r\n");
      net.connect();
      // Show the network address
      const char *ip = net.get_ip_address();
      pc.printf("IP address is: %s\r\n", ip ? ip : "No IP");
    }
    if(c == 'r'){
      socket.close();
      net.disconnect();
      pc.printf("IP Address Released\r\n");
    }
    if(c == 'h'){

     pc.printf("Sending HTTP Request...\r\n");
     // Open a socket on the network interface, and create a TCP connection
     socket.open(&net);
     socket.connect("www.updateserver.com", 80);
     // Send a simple http request
     scount = socket.send(sbuffer, sizeof sbuffer);
     pc.printf("sent %d [%.*s]\r\n", scount, strstr(sbuffer, "\r\n")-sbuffer, sbuffer);
     // Receive a simple http response and print out the response line
     rcount = socket.recv(rbuffer, sizeof rbuffer);
```

```
      pc.printf("recv %d [%.*s]\r\n", rcount, strstr(rbuffer, "\r\n")-rbuffer, rbuffer);
      socket.close();
     }
     if(c == 'b'){
      pc.printf("Sending Blocked HTTP Request...\r\n");
      // Open a socket on the network interface, and create a TCP connection
      socket.open(&net);
      socket.connect("www.unapprovedserver.com", 80);
      // Send a simple http request
      bcount = socket.send(bbuffer, sizeof bbuffer);
      pc.printf("sent %d [%.*s]\r\n", bcount, strstr(bbuffer, "\r\n")-bbuffer, bbuffer);

      // Receive a simple http response and print out the response line
      brcount = socket.recv(brbuffer, sizeof brbuffer);
      pc.printf("recv %d [%.*s]\r\n", brcount, strstr(brbuffer, "\r\n")-brbuffer,
brbuffer);
      socket.close();
     }
   }
}
```

2.5.4 IoT Devices–Non-MUD-Capable

This section details configuration of non-MUD-capable IoT devices attached to the implementation network. These include several types of devices, such as cameras, mobile phones, lighting, a connected assistant, a printer, a baby monitor, a wireless access point, and a digital video recorder. These devices did not emit a MUD URL or have MUD capabilities of any kind.

2.5.4.1 Configuration Overview

These non-MUD-capable IoT devices are unmodified and still retain the default manufacturer configurations.

2.5.4.1.1 Network Configuration

These IoT devices are configured to obtain an IP address via DHCP.

2.5.4.1.2 Software Configuration

The software on these devices is configured according to standard manufacturer instructions.

2.5.4.1.3 Hardware Configuration

The hardware used in these devices is unmodified from manufacturer specifications.

2.5.4.2 Setup

These devices were set up according to the manufacturer instructions and connected to the Cisco switch via Ethernet cable or connected wirelessly through the wireless access point.

2.5.4.2.1 DHCP Client Configuration

These IoT devices used the default DHCP clients provided by the original manufacturer and were not modified in any way.

2.6 Update Server

This section describes how to implement a server that will act as an update server. It will attempt to access and be accessed by the IoT device, in this case one of the development kits we built in the lab.

2.6.1 Update Server Overview

The update server is an Apache web server that hosts mock software update files to be served as software updates to our IoT device devkits. When the server receives an http request, it sends the corresponding update file.

2.6.2 Configuration Overview

The following subsections document the software, hardware, and network requirements for the update server.

2.6.2.1 Network Configuration

The IP address was statically assigned.

2.6.2.2 Software Configuration

For this build, the update server was configured on the Ubuntu 18.04 LTS operating system.

2.6.2.3 Hardware Configuration

The update server was hosted in the NCCoE's virtual environment, functioning as a cloud service.

2.6.3 Setup

The Apache web server was set up by using the official Apache documentation at https://httpd.apache.org/docs/current/install.html. After completing the process, the SSL/TLS encryption was set up by using the digital certificate and key obtained from DigiCert. This was set up by using the official Apache documentation, found at https://httpd.apache.org/docs/current/ssl/ssl_howto.html.

The following configurations were made to the server to host the update file:

1. Open a terminal.
2. Change directories to the Hypertext Markup Language (HTML) folder:

```
cd /var/www/html/
```

3. Create the update file (Note: this is a mock update file):

```
touch IoTsoftwareV2.tar.gz
```

2.7 Unapproved Server

This section describes how to implement a server that will act as an unapproved server. It will attempt to access and to be accessed by an IoT device, in this case one of the MUD-capable devices on the implementation network.

2.7.1 Unapproved Server Overview

The unapproved server is an internet host that is not explicitly authorized in the MUD file to communicate with the IoT device. When the IoT device attempts to connect to this server, the router or switch should not allow this traffic because it is not an approved internet service as defined by the corresponding MUD file. Likewise, when the server attempts to connect to the IoT device, this traffic should be denied at the router or switch.

2.7.2 Configuration Overview

The following subsections document the software, hardware, and network configurations for the unapproved server.

2.7.2.1 Network Configuration

The unapproved server hosts a web server that is accessed via transmission control protocol (TCP) port 80. Any applications that request access to this server need to be able to connect on this port. Use firewall-cmd, iptables, or any other system utility for manipulating the firewall to open this port.

2.7.2.2 Software Configuration

For this build, the CentOS 7 OS was leveraged with an Apache web server.

2.7.2.3 Hardware Configuration

The unapproved server was hosted in the NCCoE's virtual environment, functioning as a cloud service. The IP address was statically assigned.

2.7.3 Setup

The following subsection describes the setup process for configuring the unapproved server.

2.7.3.1 Apache Web Server

The Apache web server was set up by using the official Apache documentation at https://httpd.apache.org/docs/current/install.html. SSL/TLS encryption was not used for this server.

2.8 MQTT Broker Server

2.8.1 MQTT Broker Server Overview

For this build, the open-source tool Mosquitto was used as the MQTT broker server. The server communicates publish and subscribe messages among multiple clients. For our implementation, this server allows mobile devices set up with the appropriate application to communicate with the MQTT-enabled IoT devices in the build. The messages exchanged by the devices are on and off messages, which allow the mobile device to control the LED light on the MQTT-enabled IoT device.

2.8.2 Configuration Overview

The following subsections document the software, hardware, and network requirements for the MQTT broker server.

2.8.2.1 Network Configuration

The MQTT broker server was hosted in the NCCoE's virtual environment, functioning as a cloud service. The IP address was statically assigned.

The server is accessed via TCP port 1883. Any clients that require access to this server need to be able to connect on this port. Use firewall-cmd, iptables, or any other system utility for manipulating the firewall to open this port.

2.8.2.2 Software Configuration

For this build, the MQTT broker server was configured on an Ubuntu 18.04 LTS operating system.

2.8.2.3 Hardware Configuration

This server was hosted in the NCCoE's virtual environment, functioning as a cloud service. The IP address was statically assigned.

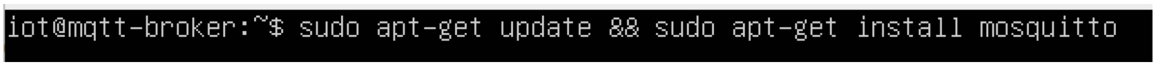
2.8.3 Setup

In this section we describe setting up the MQTT broker server to communicate messages to and from the controlling application and the IoT device.

2.8.3.1 Mosquitto Setup

1. Install the open-source MQTT broker server, Mosquitto, by entering the following command:

```
sudo apt-get update && sudo apt-get install mosquitto
```



Following the installation, this implementation leveraged the default configuration of the Mosquitto server. The MQTT broker server was set up by using the official Mosquitto documentation at https://mosquitto.org/man/.

2.9 Forescout–IoT Device Discovery

This section describes how to implement Forescout's appliance and enterprise manager to provide device discovery on the network.

2.9.1 Forescout Overview

The Forescout appliance discovers, catalogs, profiles, and classifies the devices that are connected to the demonstration network. When a device is added to or removed from the network, the Forescout appliance is updated and actively monitors these devices on the network. The administrator will be able to manage multiple Forescout appliances from a central point by integrating the appliance with the enterprise manager.

2.9.2 Configuration Overview

The following subsections document the software, hardware, and network requirements for the Forescout appliance and enterprise manager.

2.9.2.1 Network Configuration

The virtual Forescout appliance was hosted on VLAN 2 of the Cisco switch. It was set up with just the monitor interface. The network configuration for the Forescout appliance was completed by using the official Forescout documentation at https://www.Forescout.com/wp-content/uploads/2018/10/CounterACT_Installation_Guide_8.0.1.pdf (see Chapters 2 and 8).

The virtual enterprise manager was hosted in the virtual environment that is shared across each build.

2.9.2.2 Software Configuration

The build leveraged a virtual Forescout appliance VCT-R version 8.0.1 along with a virtual enterprise manager VCEM-05 version 8.0.1. Both virtual appliances were built on a Linux OS supported by Forescout.

Forescout provides software for managing the appliances on the network. The Forescout console is software that allows management of the Forescout appliance/enterprise manager and visualization of the data gathered by the appliances.

2.9.2.3 Hardware Configuration

The build leveraged a virtual Forescout appliance, which was set up in the lab environment on a dedicated machine hosting the local virtual machines in Build 1.

The virtual enterprise manager was hosted in the NCCoE's virtual environment with a static IP assignment.

2.9.3 Setup

In this section we describe setting up the virtual Forescout appliance and the virtual enterprise manager.

2.9.3.1 Forescout Appliance Setup

The virtual Forescout appliance was set up by using the official Forescout documentation at https://www.Forescout.com/wp-content/uploads/2018/10/CounterACT_Installation_Guide_8.0.1.pdf (see Chapters 3 and 8).

2.9.3.2 Enterprise Manager Setup

The enterprise manager was set up by using the official Forescout documentation at https://www.Forescout.com/wp-content/uploads/2018/10/CounterACT_Installation_Guide_8.0.1.pdf (see Chapters 4 and 8).

Using the enterprise manager, we configured the following modules:

- Endpoint
- Network
- Authentication
- Core Extension
- Device Profile Library—https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_Device_Profile_Library.pdf

- IoT Posture Assessment Library—https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_IoT_Posture_Assessment_Library-1.pdf
- Network Interface Card (NIC) Vendor DB—https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_NIC_Vendor_DB_17.0.12.pdf
- Windows Applications—https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_Windows_Applications.pdf
- Windows Vulnerability Database (DB)—https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_Windows_Vulnerability_DB_18.0.2.pdf
- Open Integration Module—https://www.Forescout.com/wp-content/uploads/2018/08/CounterACT_Open_Integration_Module_Overview_1.1.pdf

3 Build 2 Product Installation Guides

This section of the practice guide contains detailed instructions for installing and configuring the products used to implement Build 2. For additional details on Build 2's logical and physical architectures, please refer to NIST SP 1800-15B.

3.1 Yikes! MUD Manager

This section describes the Yikes! MUD manager version v1.1.3, which is a software package deployed on the Yikes! router. It should not require configuration as it should be fully functioning upon connecting the Yikes! router to the network.

3.1.1 Yikes! MUD Manager Overview

The Yikes! MUD manager is a software package supported by MasterPeace within the Yikes! physical router. The version of the Yikes! router used in this implementation supports IoT devices that leverage DHCP as their default MUD emission method.

3.1.2 Configuration Overview

At this implementation, no additional network, software, or hardware configuration was required to enable the Yikes! MUD manager capability on the Yikes! router.

3.1.3 Setup

At this implementation, no setup was required to enable the Yikes! MUD manager capability on the Yikes! router. See the Yikes! Router section for details on the router setup.

3.2 MUD File Server

3.2.1 MUD File Server Overview

For this build, the NCCoE leveraged a MUD file server hosted by MasterPeace. This file server hosts MUD files along with their corresponding signature files for the MUD-capable IoT devices used in Build 2. The MUD file server is responsible for serving the MUD file and the corresponding signature file upon request from the MUD manager. These files were created by the NCCoE and provided to MasterPeace to host due to the Yikes! cloud component requirement that the MUD file server be internet accessible to display the contents of the MUD file in the Yikes! user interface (UI).

To build an on-premises MUD file server and to create MUD files for MUD-capable IoT devices, please follow the instructions in Build 1’s MUD File Server section.

3.3 Yikes! DHCP Server

This section describes the Yikes! DHCP server, which should also be fully functional out of the box and should not require any modification upon receipt.

3.3.1 Yikes! DHCP Server Overview

The Yikes! DHCP server is MUD capable and, like the Yikes! MUD manager and Yikes! threat-signaling agent, is a logical component within the Yikes! router. In addition to dynamically assigning IP addresses, it recognizes the DHCP option (161) and logs DHCP events that include this option to a log file. This log file is monitored by the Yikes! MUD manager, which is responsible for handling the MUD requests.

3.3.2 Configuration Overview

At this implementation, no additional network, software, or hardware configuration was required to enable the Yikes! DHCP server capability on the Yikes! router.

3.3.3 Setup

At this implementation, no additional setup was required.

3.4 Yikes! Router

This section describes how to implement and configure the Yikes! router, which requires minimal configuration from a user standpoint.

3.4.1 Yikes! Router Overview

The Yikes! router is a customized original equipment manufacturer product, which at implementation was a preproduction product. It is a self-contained router, Wi-Fi access point, and firewall that communicates locally with Wi-Fi devices and wired devices. The Yikes! router leveraged in this implementation was developed on an OpenWRT base router with the Yikes! capabilities added on. The Yikes! router hosts all the software necessary to enable a MUD infrastructure on premise. It also communicates with the Yikes! cloud and threat-signaling services to support additional capabilities in the network.

At this implementation, the Yikes! MUD manager, DHCP server, and GCA threat-signaling components all reside on the Yikes! router and are configured to function without any additional configuration.

3.4.2 Configuration Overview

3.4.2.1 Network Configuration

Implementation of a Yikes! router requires an internet source such as a Digital Subscriber Line (DSL) or cable modem.

3.4.2.2 Software Configuration

At this implementation, no additional software configuration was required to set up the Yikes! router.

3.4.2.3 Hardware Configuration

At this implementation, no additional hardware configuration was required to set up the Yikes! router.

3.4.3 Setup

As stated earlier, the version of the Yikes! router used in Build 2 was preproduction, so MasterPeace may have performed some setup and configuration steps that are not documented here. Those additional steps, however, are not expected to be required to set up the production version of the router. The following setup steps were performed:

1. Unbox the Yikes! router and provided accessories.
2. Connect the Yikes! router's wide area network port to an internet source (e.g., cable modem or DSL).
3. Plug the power supply into the Yikes! router.
4. Power on the Yikes! router.

After powering on the router, the network password must be provided so the router can authenticate itself to the network. In addition, best security practices (not documented here), such as changing the router's administrative password, should be followed in accordance with the security policies of the user.

3.5 DigiCert Certificates

DigiCert's CertCentral web-based platform allows provisioning and management of publicly trusted X.509 certificates for a variety of purposes. After establishing an account, clients can log in, request, renew, and revoke certificates by using only a browser. For Build 2, the Premium Certificate created in Build 1 was leveraged for signing the MUD files. To request and implement DigiCert certificates, follow the documentation in Build 1's DigiCert Certificates section and subsequent sections.

3.6 IoT Devices

3.6.1 IoT Development Kits—Linux Based

3.6.1.1 Configuration Overview

This section provides configuration details for the Linux-based IoT development kits used in the build, which emit MUD URLs by using DHCP. It also provides information regarding a basic IoT application used to test the MUD process.

3.6.1.1.1 Network Configuration

The devkits are connected to the network over both a wired Ethernet connection and wirelessly. The IP address is assigned dynamically by using DHCP.

3.6.1.1.2 Software Configuration

For this build, Raspberry Pi is configured on Raspbian 9, the Samsung ARTIK 520 is configured on Fedora 24, the NXP i.MX 8m is configured on Yocto Linux, and the BeagleBone Black is configured on Debian 9.5. The devkits also utilized a variety of DHCP clients, including dhcpcd and dhclient (see Build 1's IoT Development Kits–Linux Based section for dhclient configurations). This build introduced dhcpcd as a method for emitting a MUD URL for all devkits in this build, apart from the NXP i.MX 8m, which leveraged dhclient. Dhcpcd is installed natively on many Linux distributions and can be installed using a preferred package manager if not currently present.

3.6.1.1.3 Hardware Configuration

The hardware used for these devkits included the Raspberry Pi 3 Model B, Samsung ARTIK 520, NXP i.MX 8m, and BeagleBone Black.

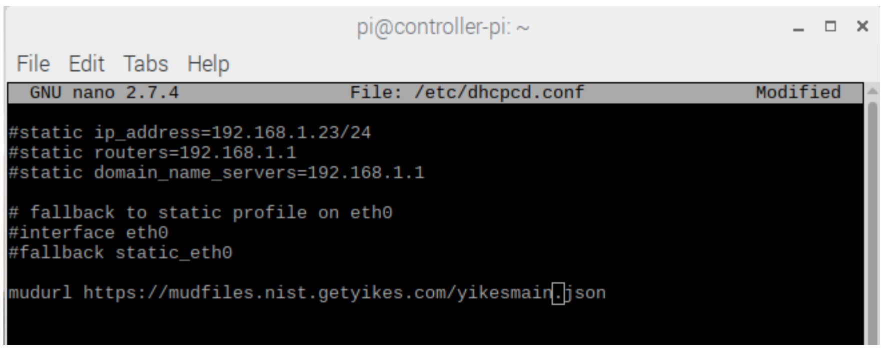
3.6.1.2 Setup

The following subsection describes setting up the devkits to send a MUD URL during the DHCP transaction using dhcpcd as the DHCP client on the Raspberry Pi. For dhclient instructions, see Build 1's Setup and DHCP Client Configuration sections.

3.6.1.2.1 DHCP Client Configuration

These devkits utilized dhcpcd version 7.2.3. Configuration consisted of adding the following line to the file located at */etc/dhcpcd.conf*:

```
mudurl https://<example-url>
```



3.7 Update Server

Build 2 leveraged the preexisting update server that is described in Build 1's Update Server section. To implement a server that will act as an update server, see the documentation in Build 1's Update Server section. The update server will attempt to access and be accessed by the IoT device, which, in this case, is one of the development kits we built in the lab.

3.8 Unapproved Server

Build 2 leverages the preexisting unapproved server that is described in Build 1's Unapproved Server section. To implement a server that will act as an unapproved server, see the documentation in Build 1's Unapproved Server section. The unapproved server will attempt to access and to be accessed by an IoT device, which, in this case, is one of the MUD-capable devices on the implementation network.

3.9 Yikes! IoT Device Discovery, Categorization, and Traffic Policy Enforcement (Yikes! Cloud and Yikes! Mobile Application)

This section describes how to implement and configure Yikes! IoT device discovery, categorization, and traffic policy enforcement, which is a capability supported by the Yikes! router, Yikes! cloud, and Yikes! mobile application.

3.9.1 Yikes! IoT Device Discovery, Categorization, and Traffic Policy Enforcement Overview

The Yikes! router provides an IoT device discovery service for Build 2. Yikes! discovers, inventories, profiles, and classifies devices connected to the local network consistent with each device's type and allows traffic enforcement policies to be configured by the user through the Yikes! mobile application.

Yikes! isolates every device on the network so that, by default, no device is permitted to communicate with any other device. Devices added to the network are automatically identified and categorized based on information such as DHCP header, MAC address, operating system, manufacturer, and model.

Using the Yikes! mobile application, users can define fine-grained device filtering. The enforcement can be set to enable specific internet access (north/south) and internal network access to specific devices (east/west) as determined by category-specific rules.

3.9.2 Configuration Overview

3.9.2.1 Network Configuration

No network configurations outside Yikes! router network configurations are required to enable this capability.

3.9.2.2 Software Configuration

MasterPeace performed some software configuration on the Yikes! router after it was deployed as part of Build 2. Aside from this, no additional software configuration was required to support device discovery. When the production version of the Yikes! router is available, it is not expected to require configuration. The Yikes! mobile application was still in development during deployment. The build used the web-based Yikes! mobile application from a laptop in the lab environment to display and configure device information and traffic policies.

3.9.2.3 Hardware Configuration

At this implementation, the Yikes! mobile application was not published in an application store. For this reason, a desktop was leveraged to load the web page hosting the "mobile application."

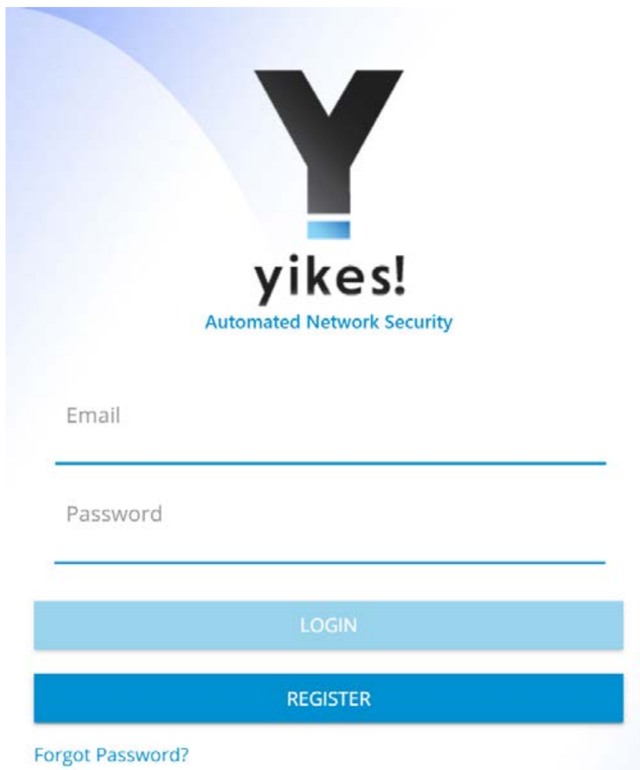
3.9.3 Setup

Once devices have been added to the network on the Yikes! router, they will appear in the Yikes! cloud inventory, which is accessible via the Yikes! mobile application. At this implementation, the Yikes! mobile application and the processes associated with the Yikes! cloud service were under development. It is possible that the design of the UI and the workflow will change for the final implementation of the mobile application.

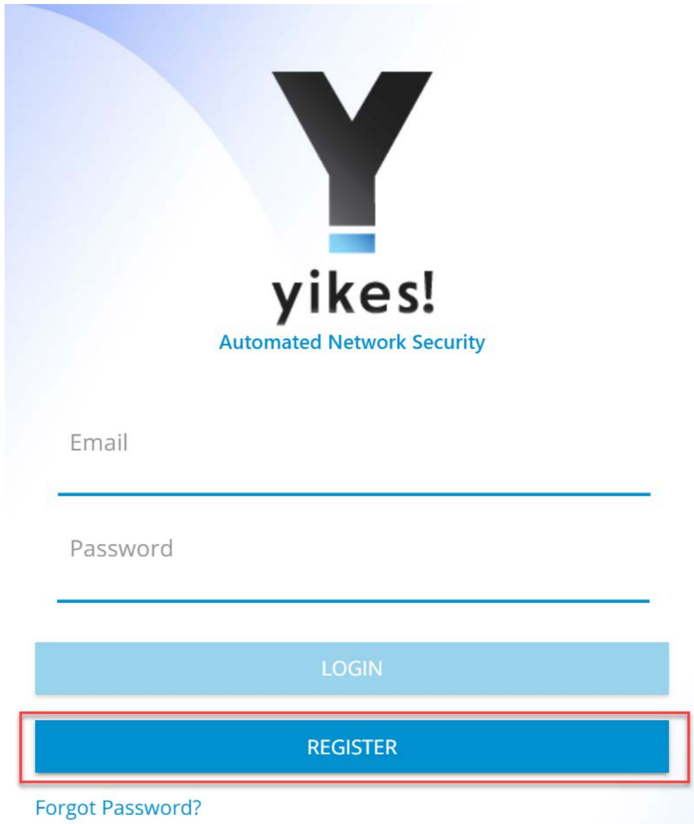
3.9.3.1 Yikes! Router and Account Cloud Registration

At this implementation, the Yikes! router and cloud account registration processes were under development. As a result, this section will not describe how to associate a Yikes! router with a Yikes! cloud instance. The steps below show the process for account registration at this implementation.

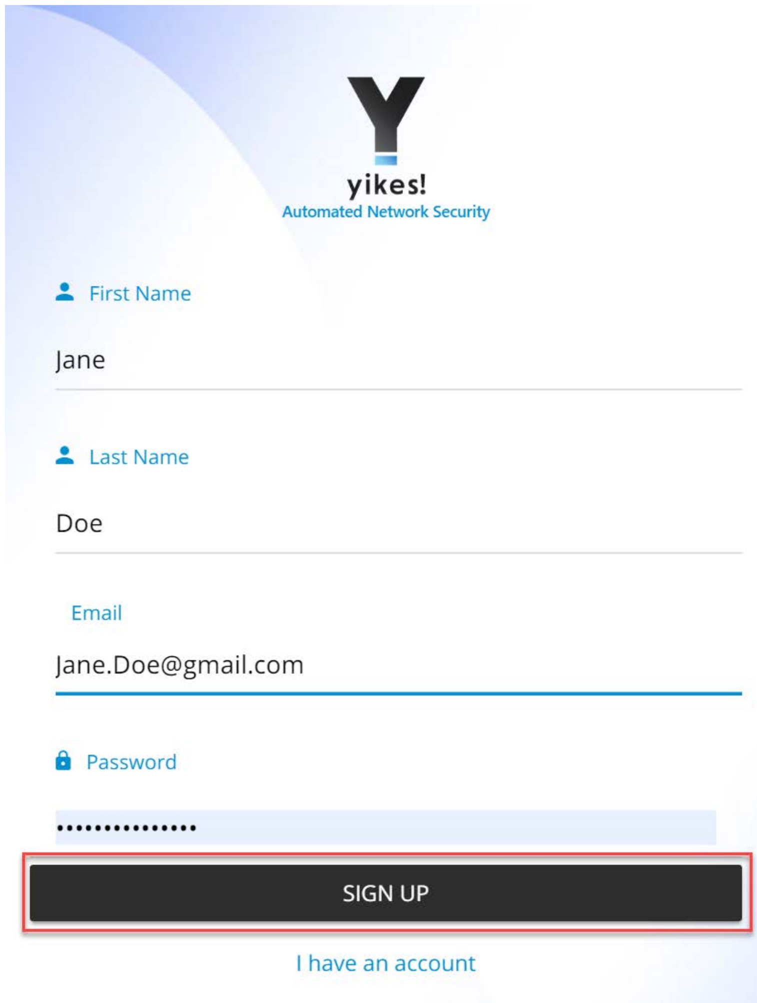
1. Open a browser and access the Yikes! UI (In the preproduction version of the router, accessing the UI required inputting a URL provided by MasterPeace):



2. Click on the **Register** button to sign up for an account:

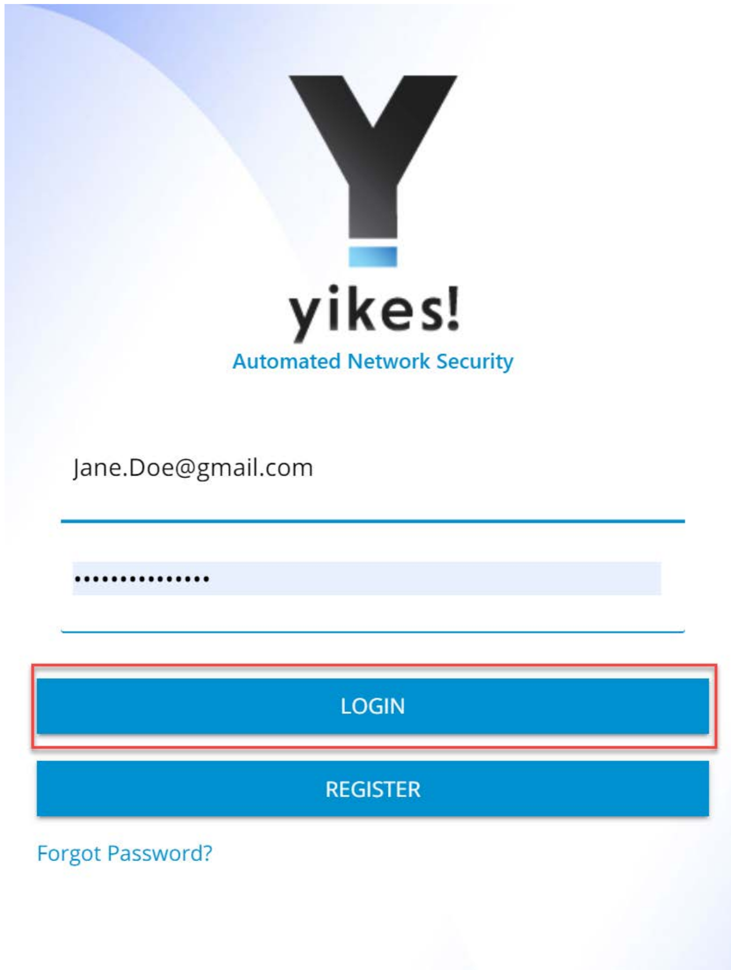


3. Populate the requested information for the account: First Name, Last Name, Email, and Password. Click **Sign Up:**

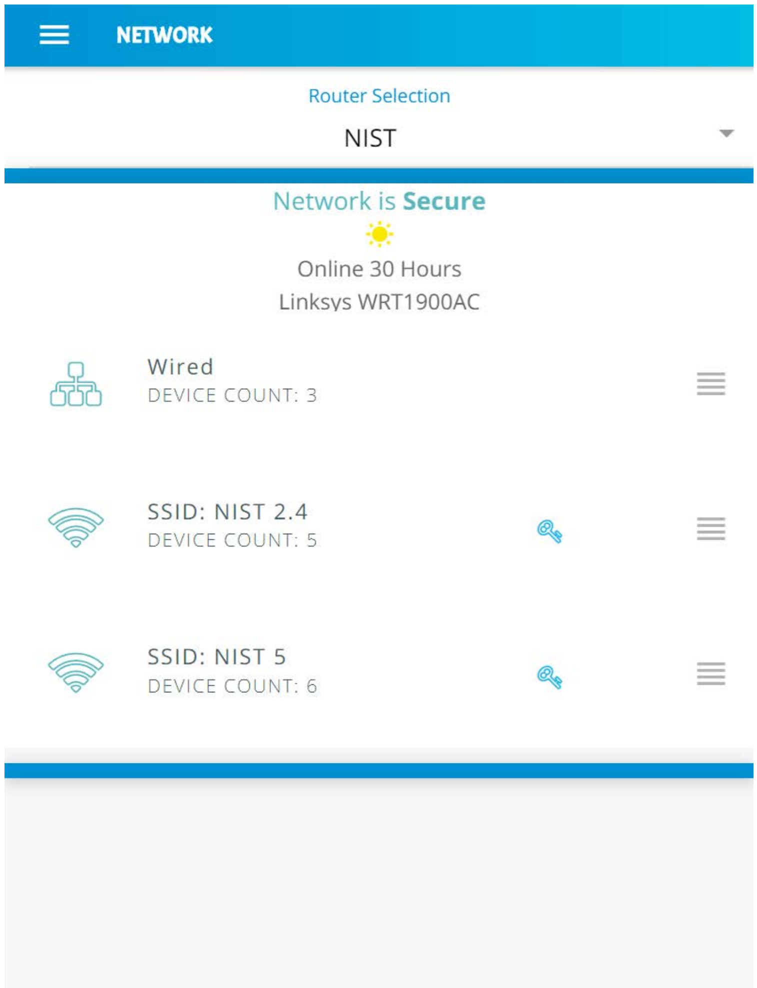


Note: There will be additional steps related to associating the Yikes! router with the Yikes! account being created. However, at this implementation, this process was still under development.

4. Once the account is approved and linked to the Yikes! router, **Log in** with the credentials created in step 3:



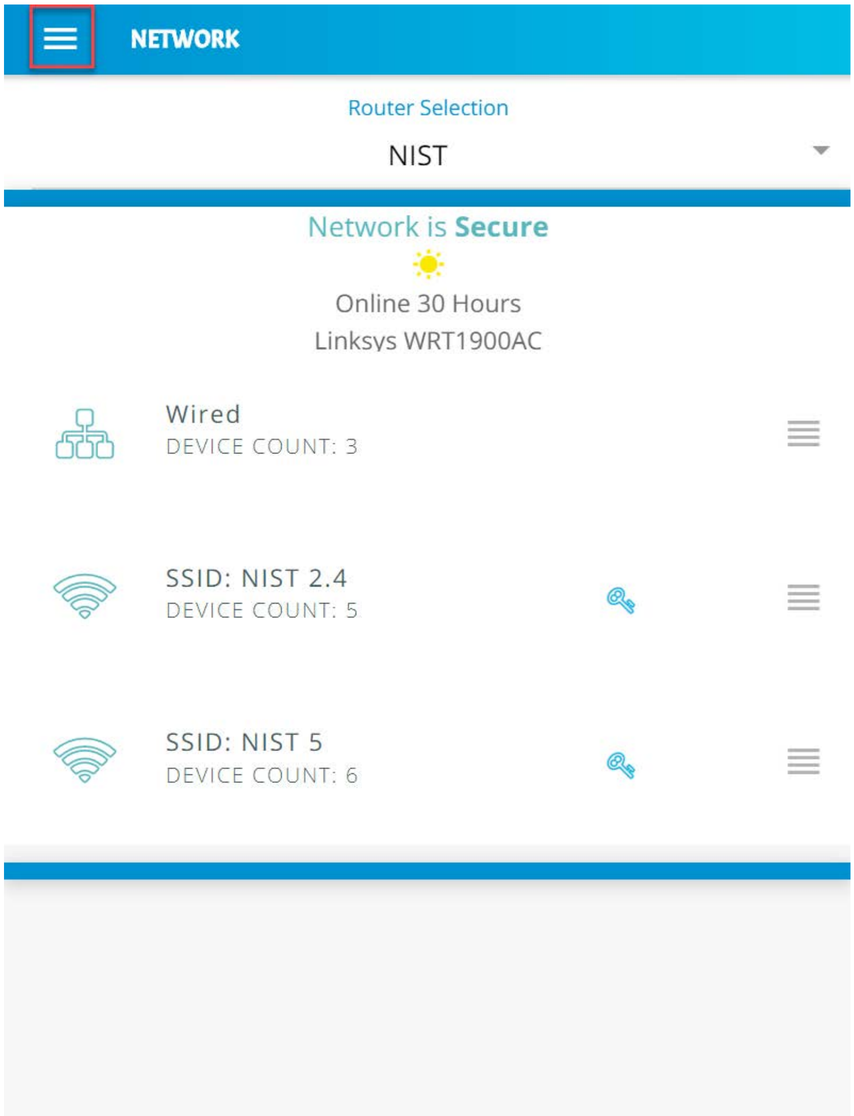
5. The home screen will show the network overview:



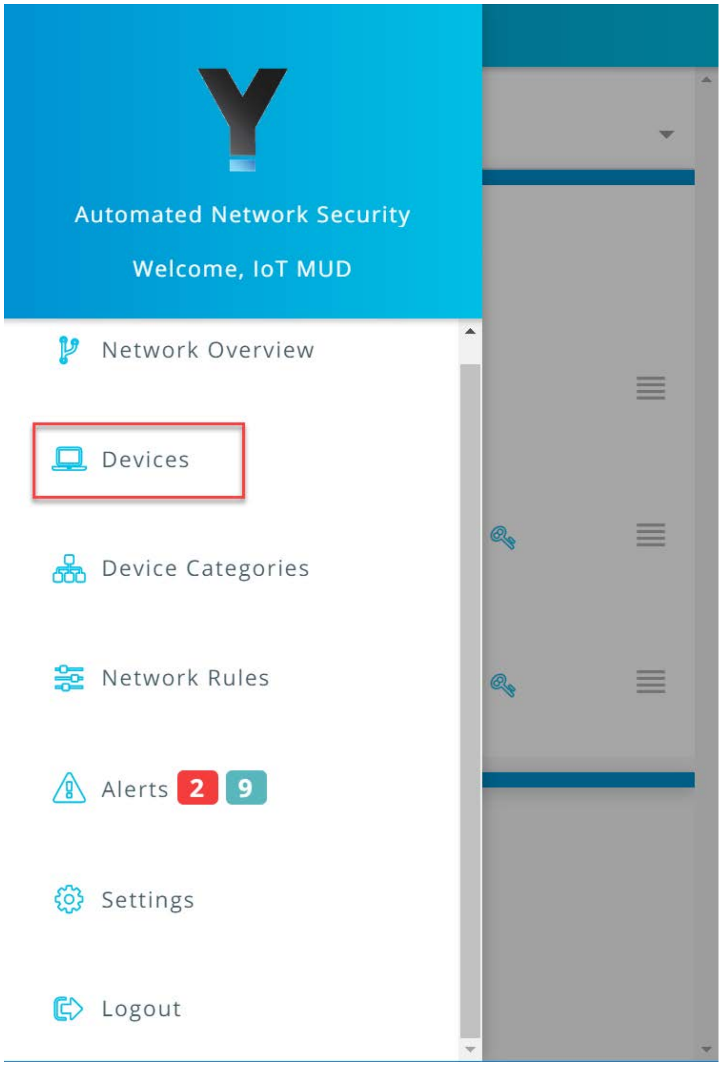
3.9.3.2 Yikes! MUD-Capable IoT Device Discovery

This section details the Yikes! MUD-capable IoT device discovery capability. This feature is accessible through the Yikes! mobile application and identifies all MUD-capable IoT devices that are connected to the network.

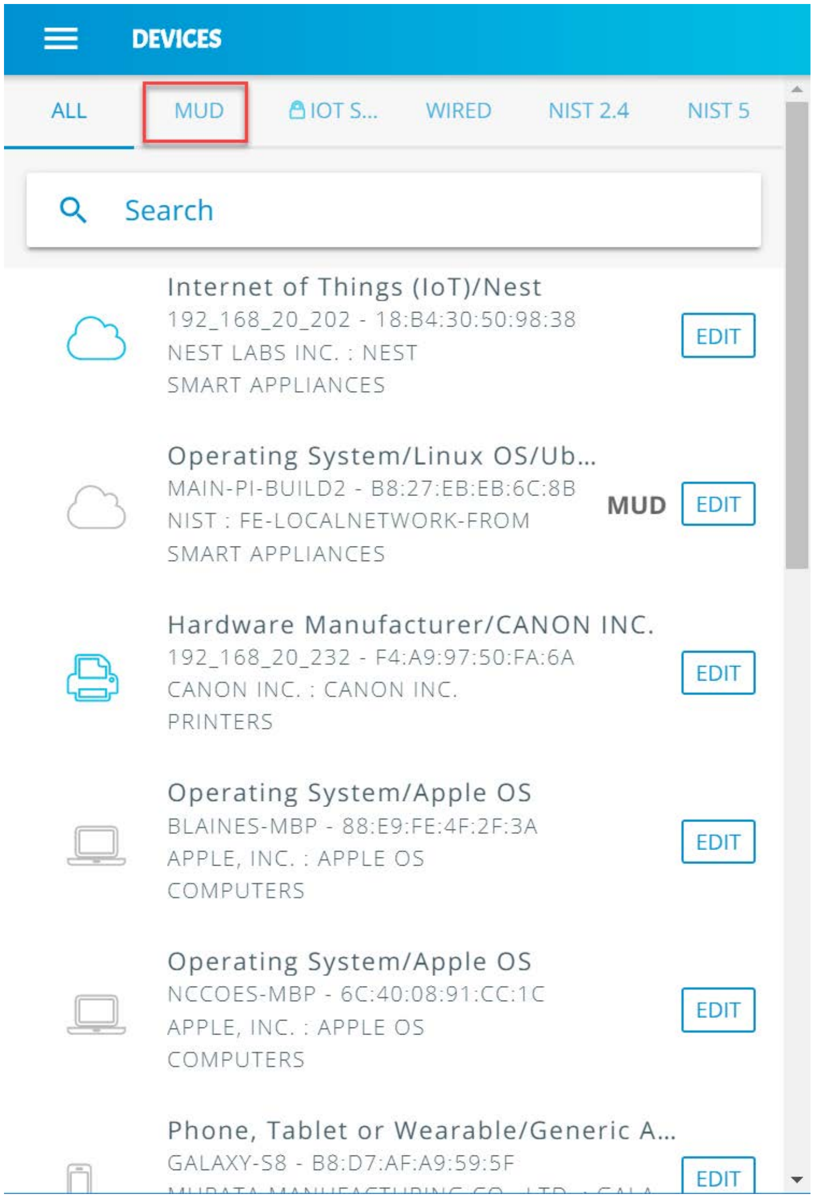
1. Open the menu pane in the UI:



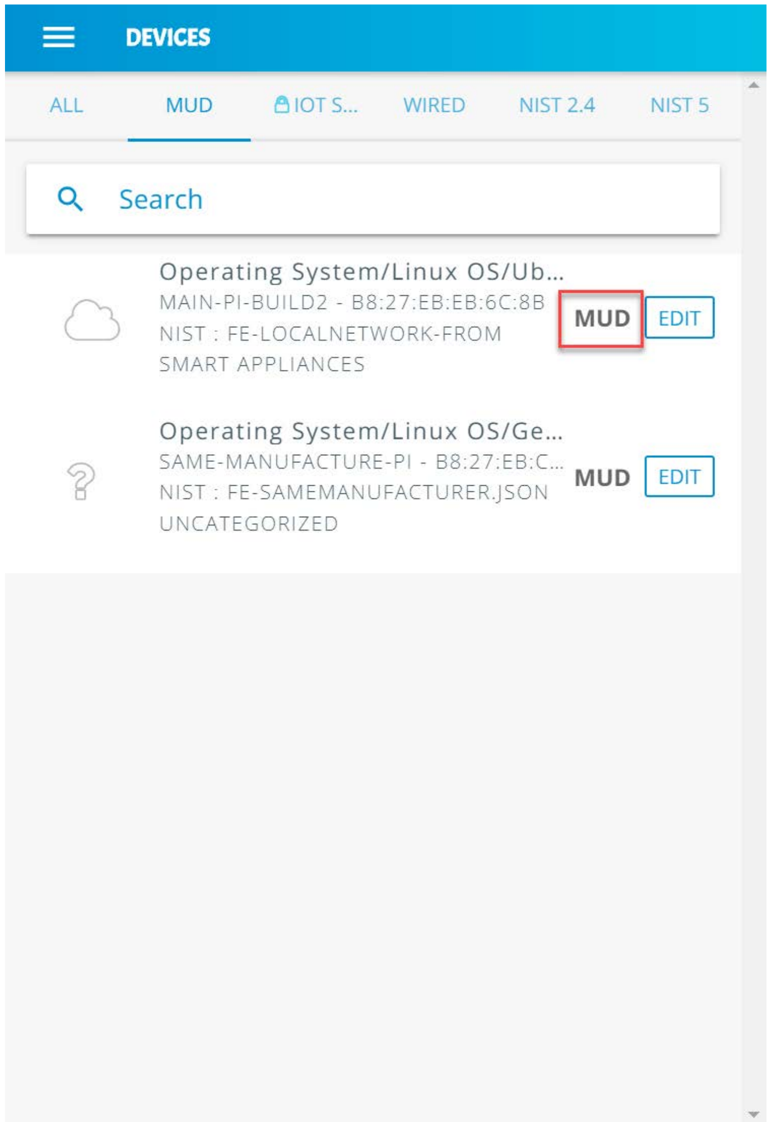
1551 2. Click the **Devices** button to open the devices menu:

1552 

3. Click the **MUD** tab to switch from the **ALL** device view to review the MUD-capable IoT devices connected to the network:



4. All MUD-capable devices on the network will have the **MUD** label as seen below:

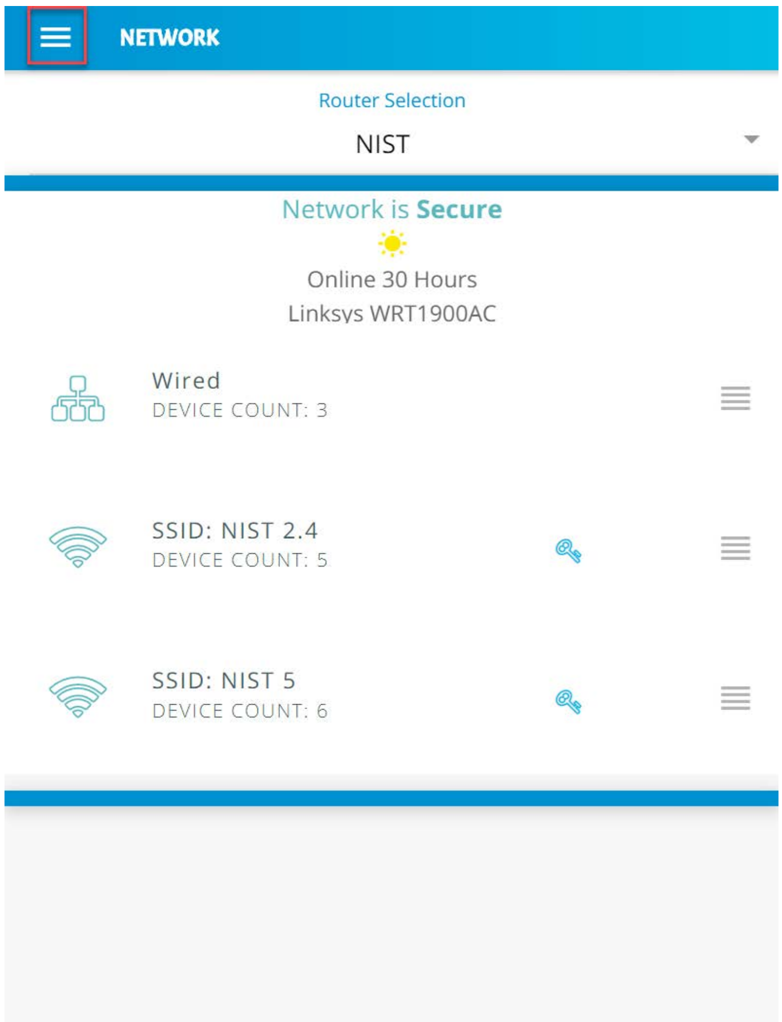


3.9.3.3 Yikes! Alerts

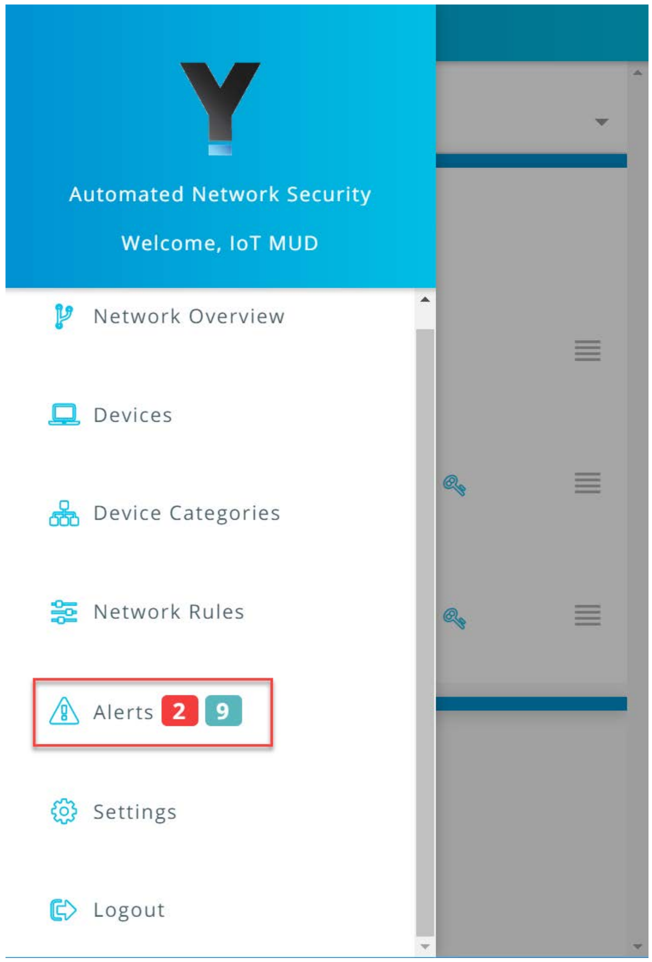
This section details the Yikes! alerting capability. This feature is accessible through the Yikes! mobile application and notifies users when new devices have been connected to the network. Additionally, this feature alerts the user when new devices are not recognized as known devices and are placed in the uncategorized device category by the Yikes! cloud.

From the Yikes! mobile application, the user can edit the information about the device (e.g., name, make, and model) and modify the device's category or can choose to ignore the alert by removing the notification.

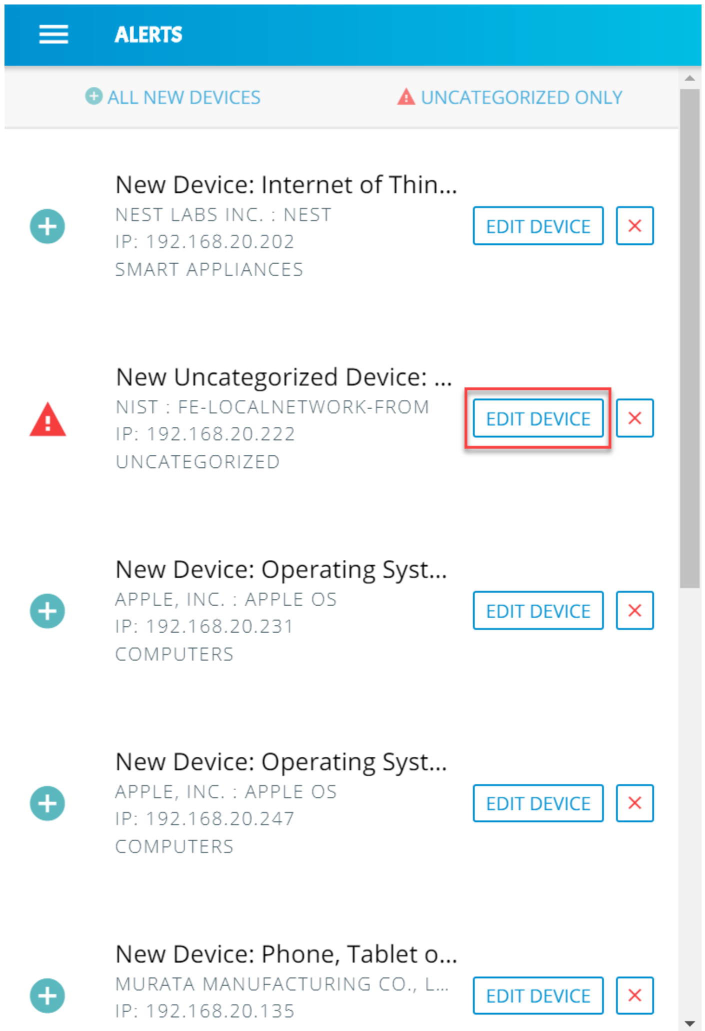
1. Open the menu pane in the UI:



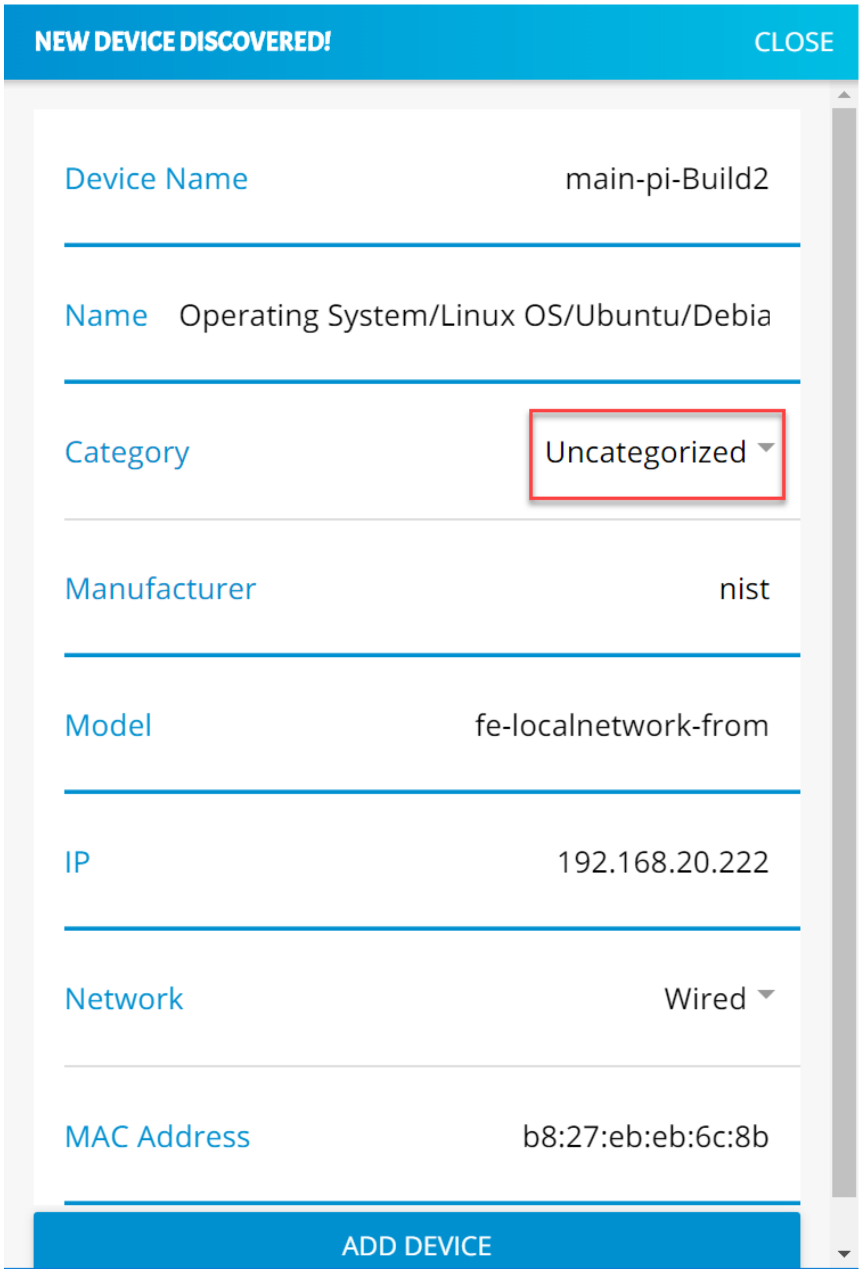
2. Click **Alerts** to open the Alerts menu:



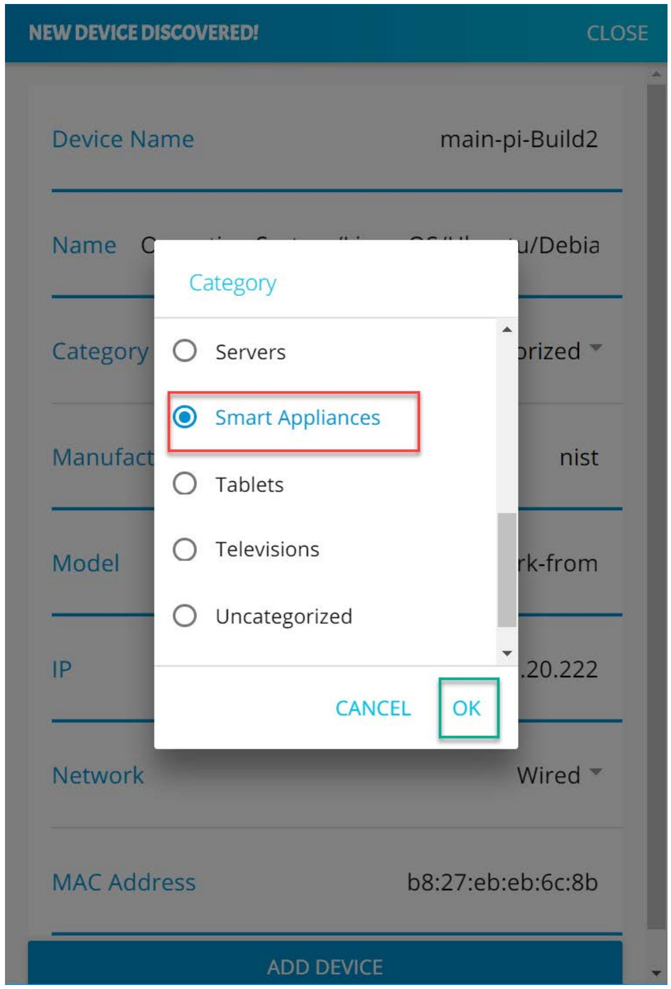
3. Select a device to edit the device information and category by clicking **Edit Device:**



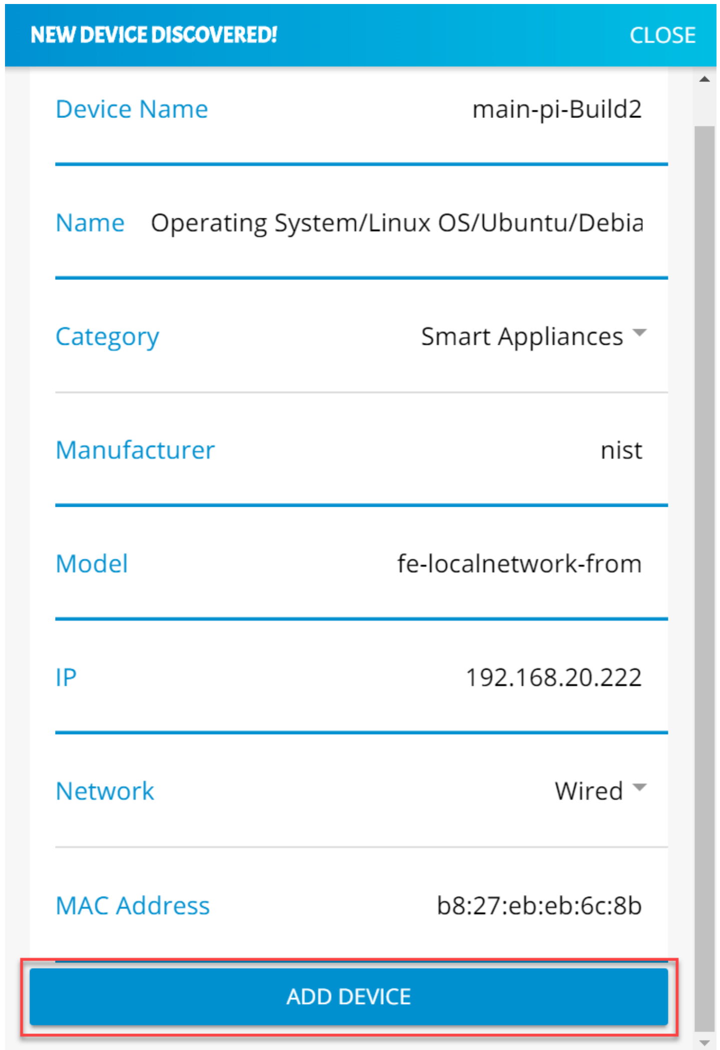
4. Modify the **Category** of the device by clicking the device's current category:



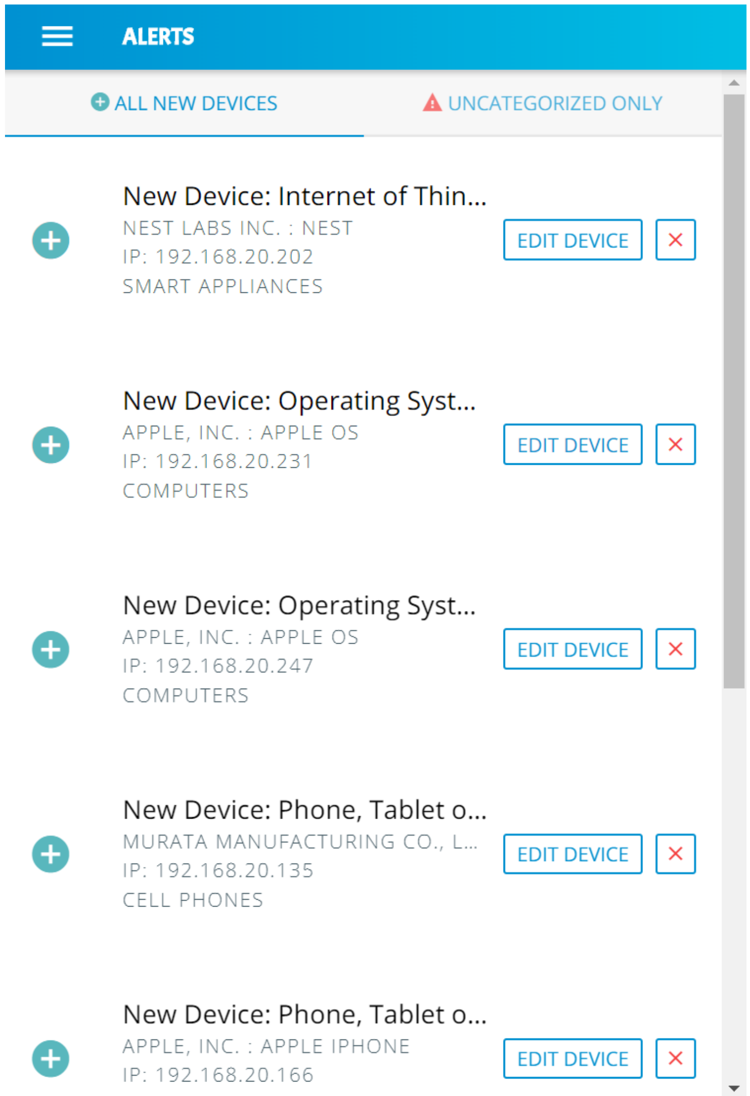
5. Select the desired category, in this case **Smart Appliances,** and click **OK:**



6. The device **Category** will update to reflect the new selection. Click **Add Device** to complete the process:



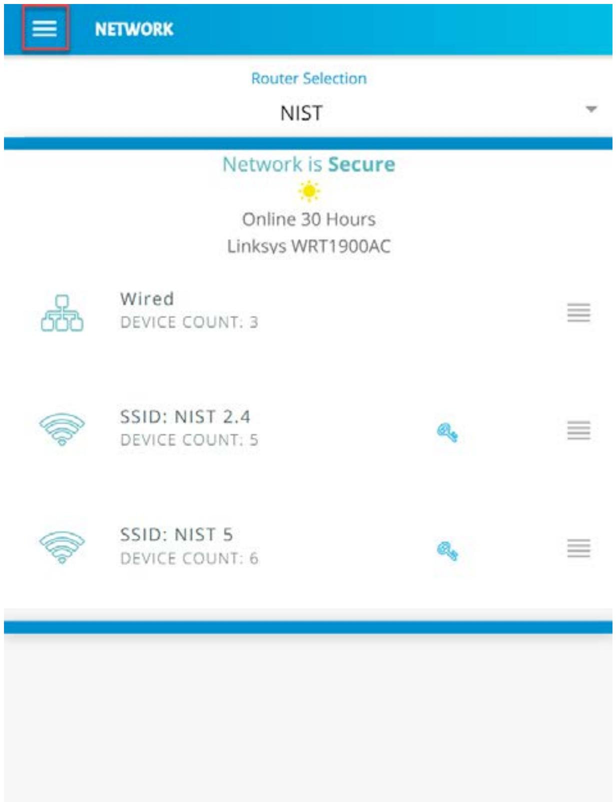
7. The alerts menu will update and no longer include the device that was just modified and added:



3.9.3.4 Yikes! Device Categories and Setting Rules

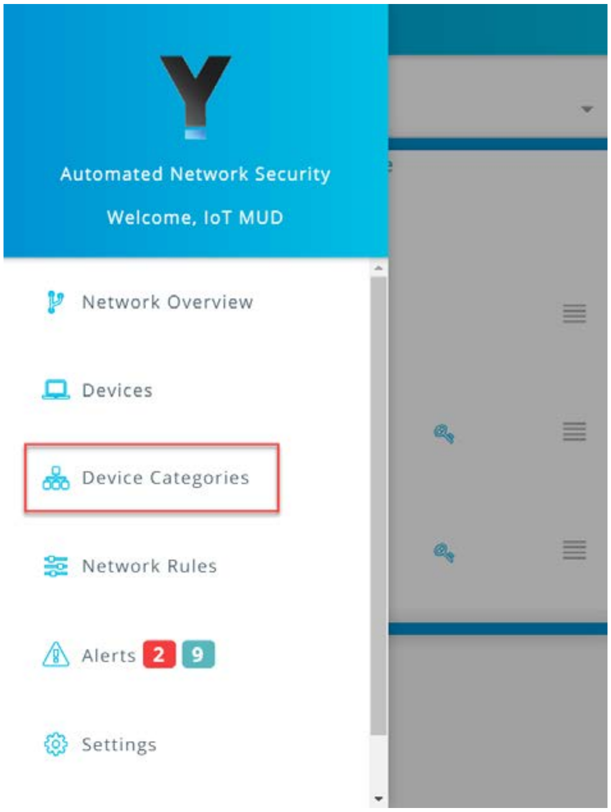
The Yikes! mobile application provides the capability to view predefined device categories and set rules for local communication between categories of devices on the local network and internet rules for all devices in a selected category.

1585 1. Click the menu bar to open the menu pane:



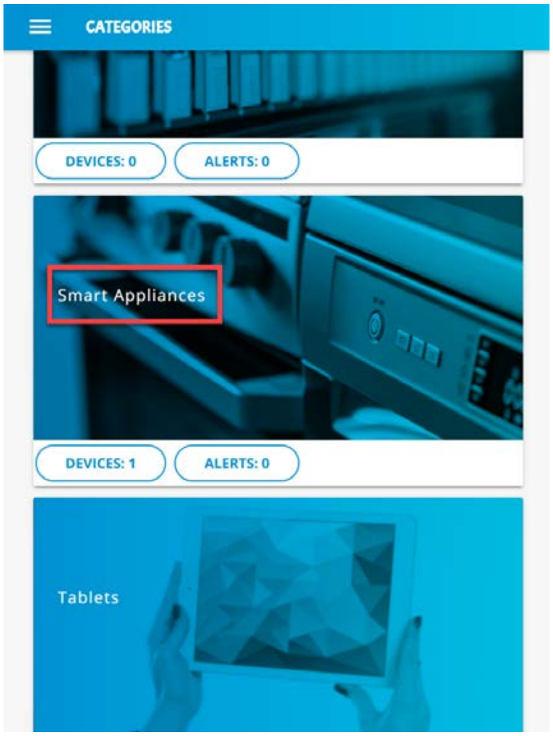
1586

1587 2. Click the **Device Categories** option to view all device categories:

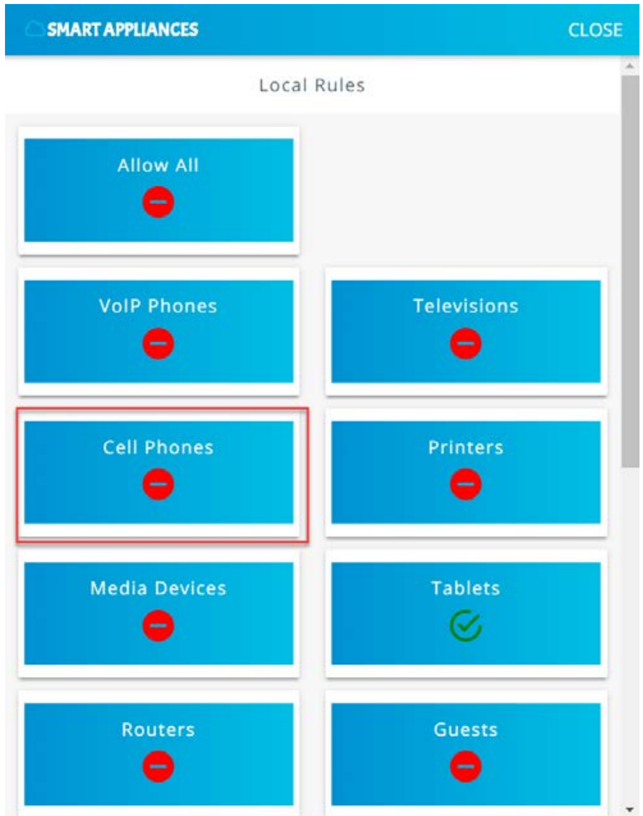


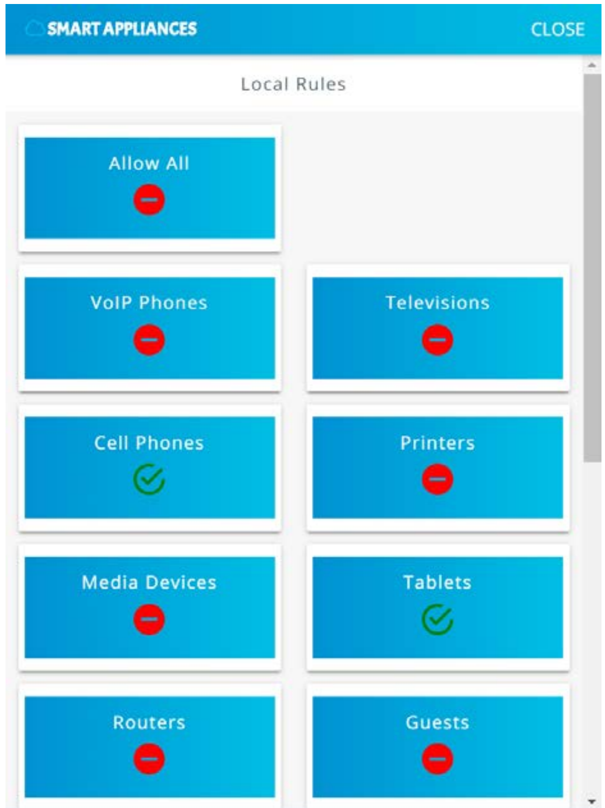
1588

3. Select the category of device to view and configure rules:

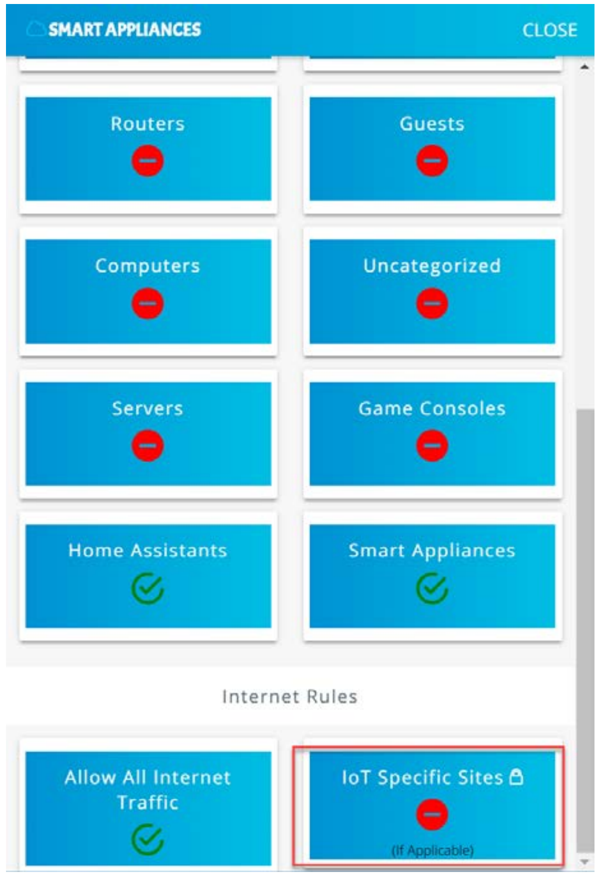


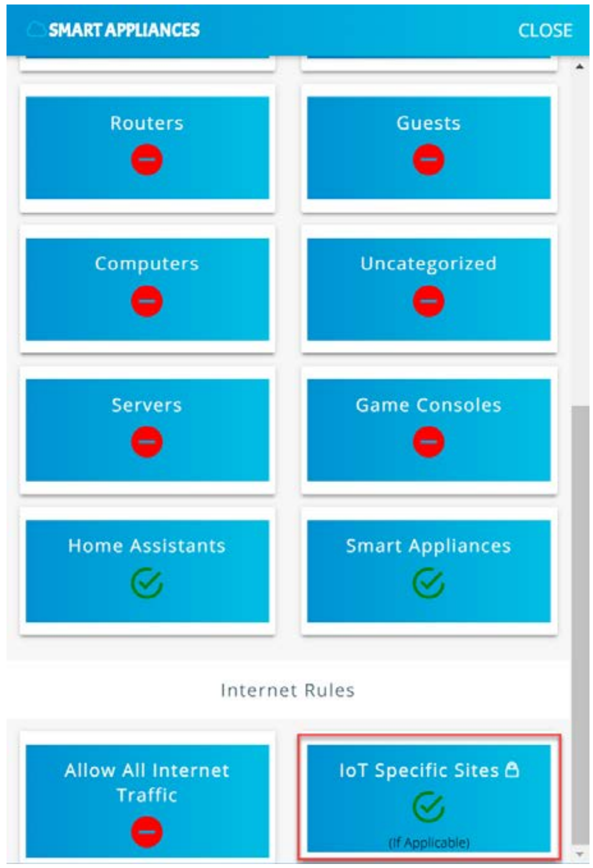
4. Modify local rules by clicking on the category of devices with which the selected category is permitted to communicate:



1594 
SMART APPLIANCES
CLOSE
Local Rules
Allow All
VoIP Phones
Televisions
Cell Phones
Printers
Media Devices
Tablets
Routers
Guests

5. Scroll to the bottom of the page to view the current **Internet Rules** for this category, and change the permissions by clicking on **IoT Specific Sites:**

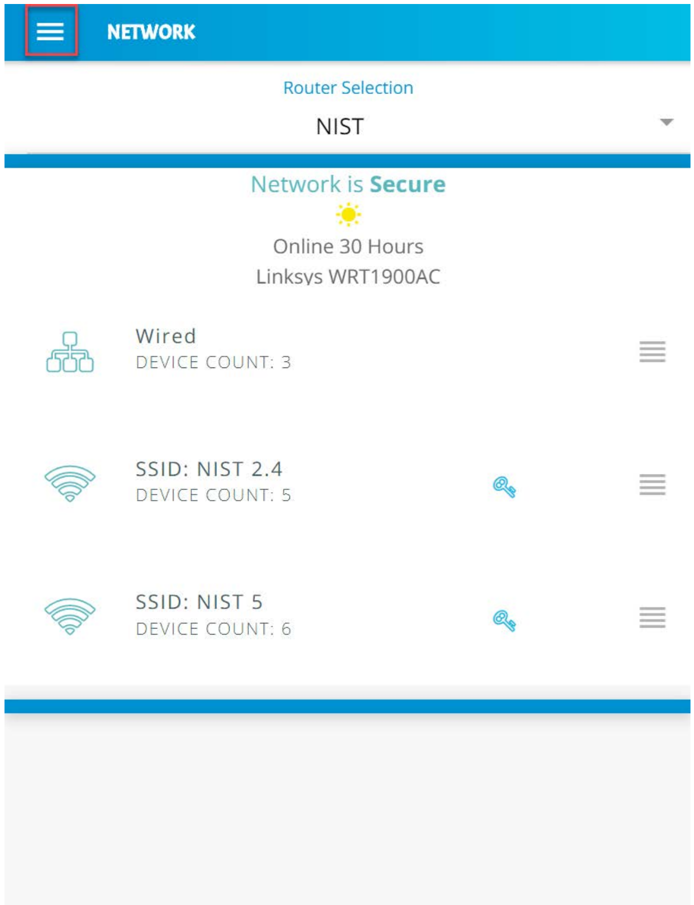
1597 
SMART APPLIANCES
CLOSE
Routers
Guests
Computers
Uncategorized
Servers
Game Consoles
Home Assistants
Smart Appliances
Internet Rules
Allow All Internet Traffic
IoT Specific Sites
(If Applicable)



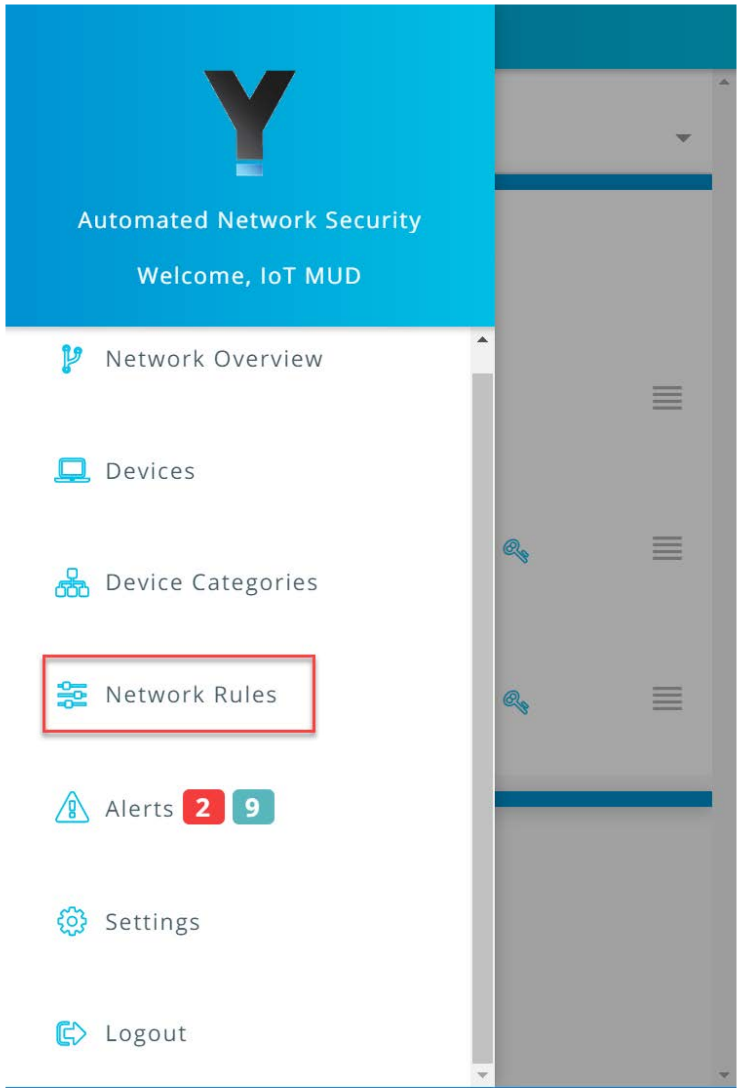
Smart appliances should now be permitted to communicate locally to Smart Appliances, Home Assistants, Tablets, Cell Phones, and, externally, to IoT Specific Sites.

3.9.3.5 Yikes! Network Rules

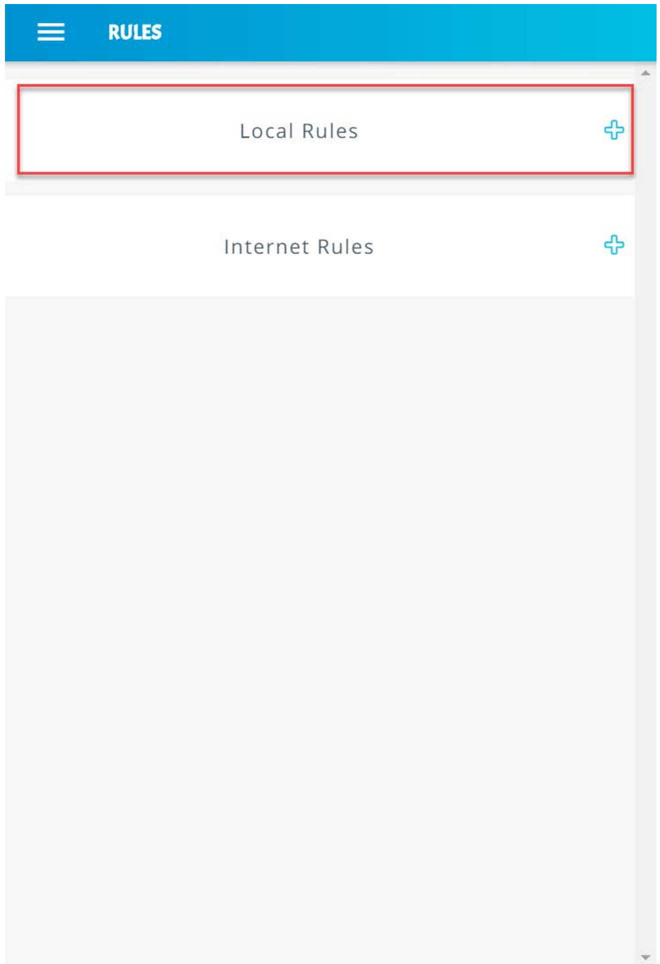
1. The Yikes! mobile application allows reviewing the rules that have been implemented on the network. These rules are divided into two main sections: Local Rules and Internet Rules. Local rules display the local communications permitted for each category of devices. Internet rules display the internet communications permitted for each category of devices. This section reviews the rules defined for Smart Appliances in Yikes! Device Categories and Setting Rules UI:


NETWORK
Router Selection
NIST
Network is Secure
Online 30 Hours
Linksys WRT1900AC
Wired
DEVICE COUNT: 3
SSID: NIST 2.4
DEVICE COUNT: 5
SSID: NIST 5
DEVICE COUNT: 6

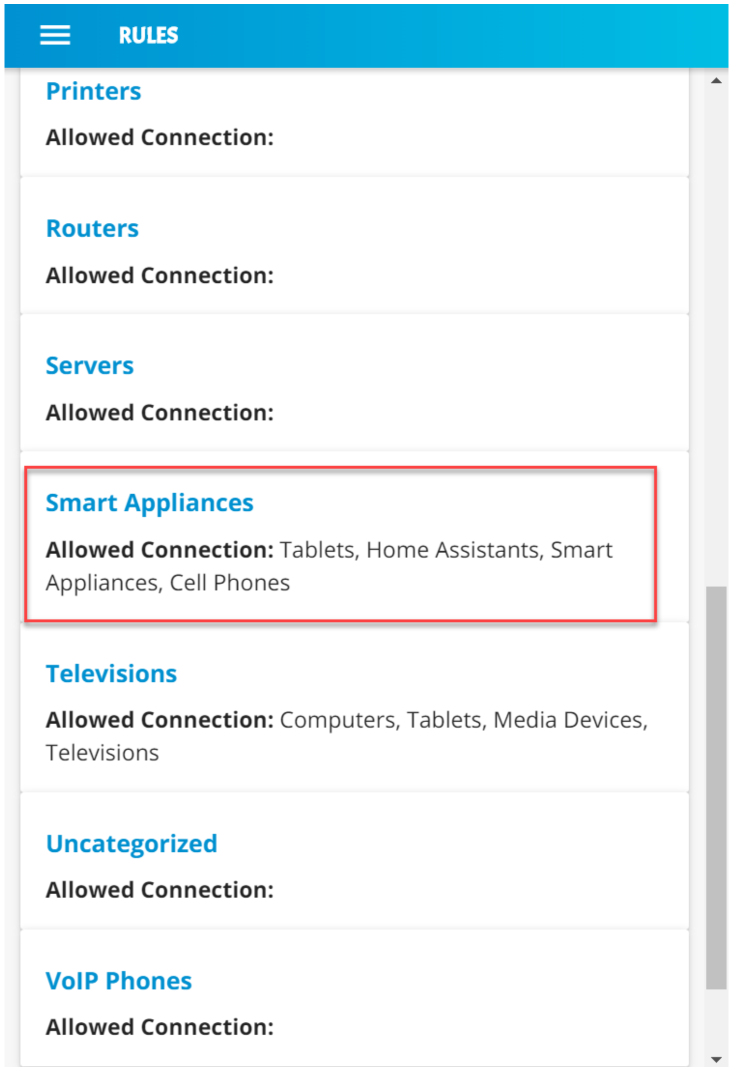
2. Click **Network Rules** to navigate to the rules menu:



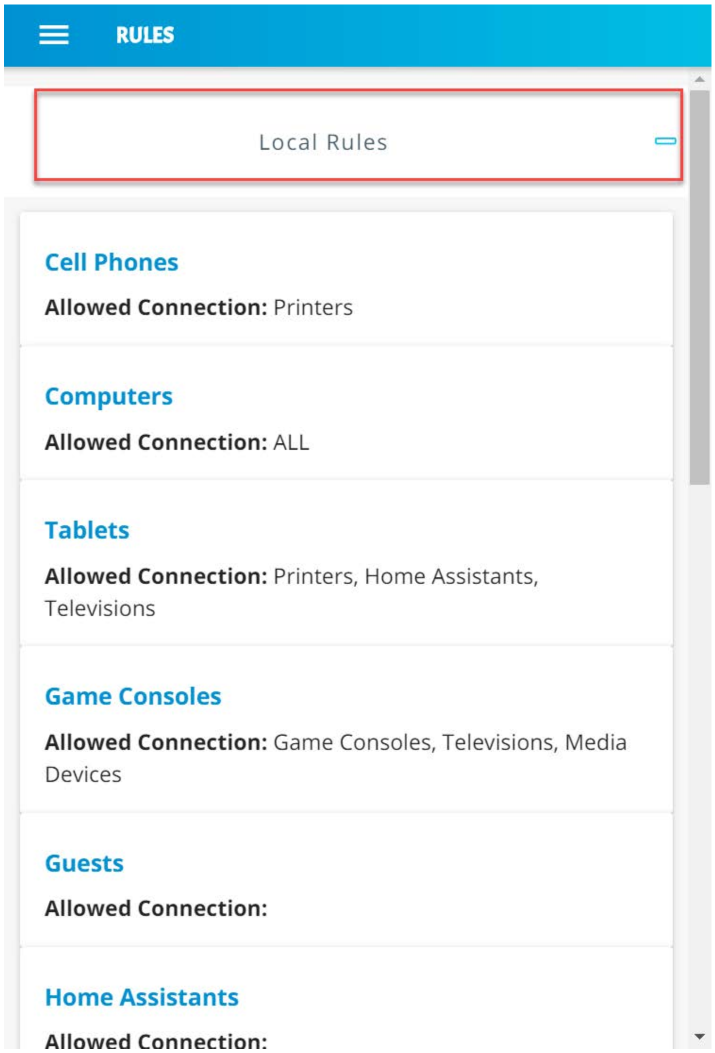
1610 3. Click **Local Rules** to view the permitted local communications for each device category:

1611 

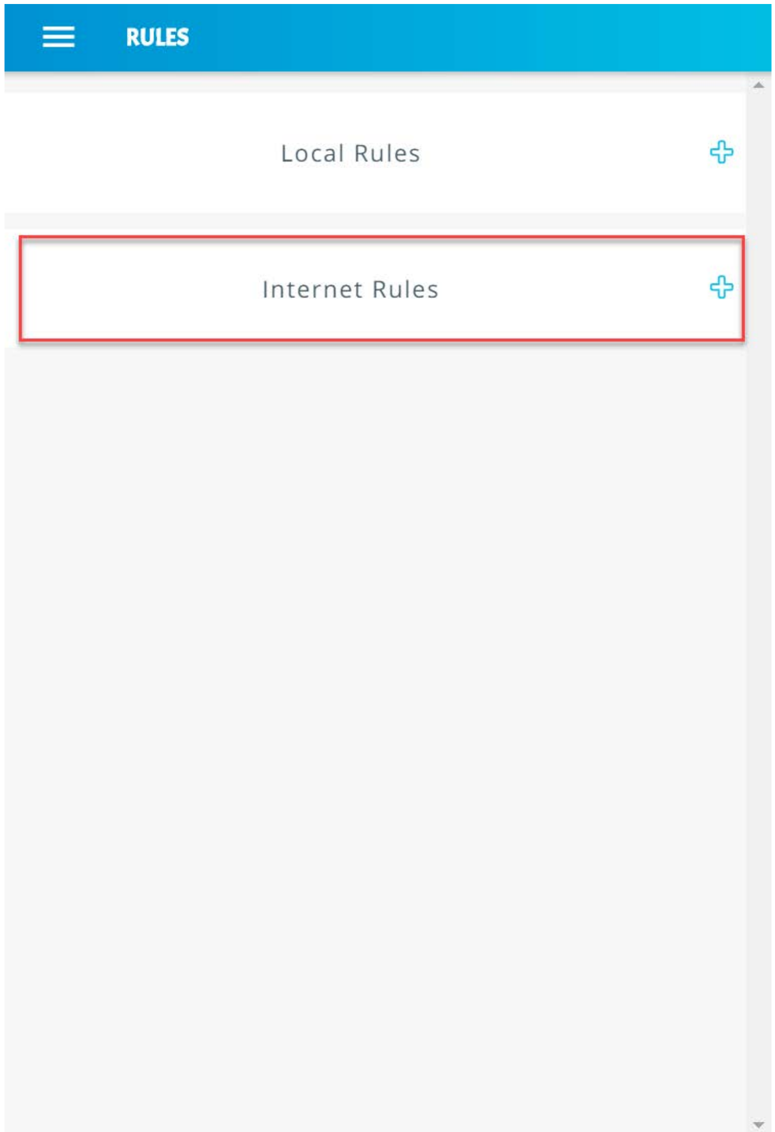
4. Scroll down to view the local rules for the **Smart Appliances** category:



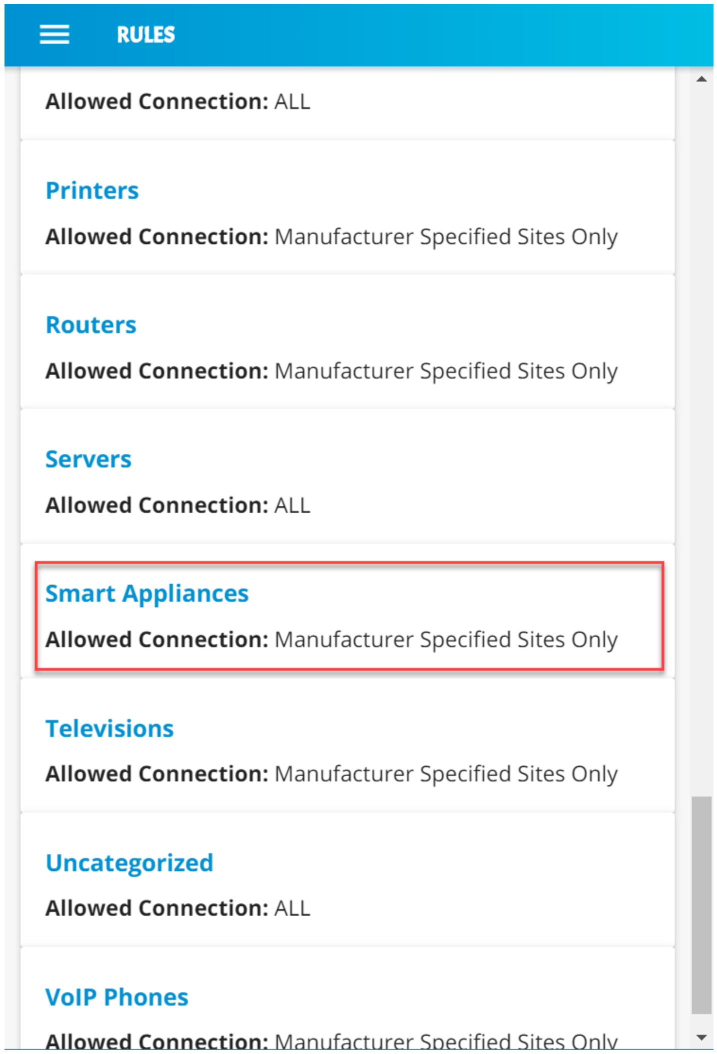
5. Minimize the rules by clicking the **Local Rules** button:



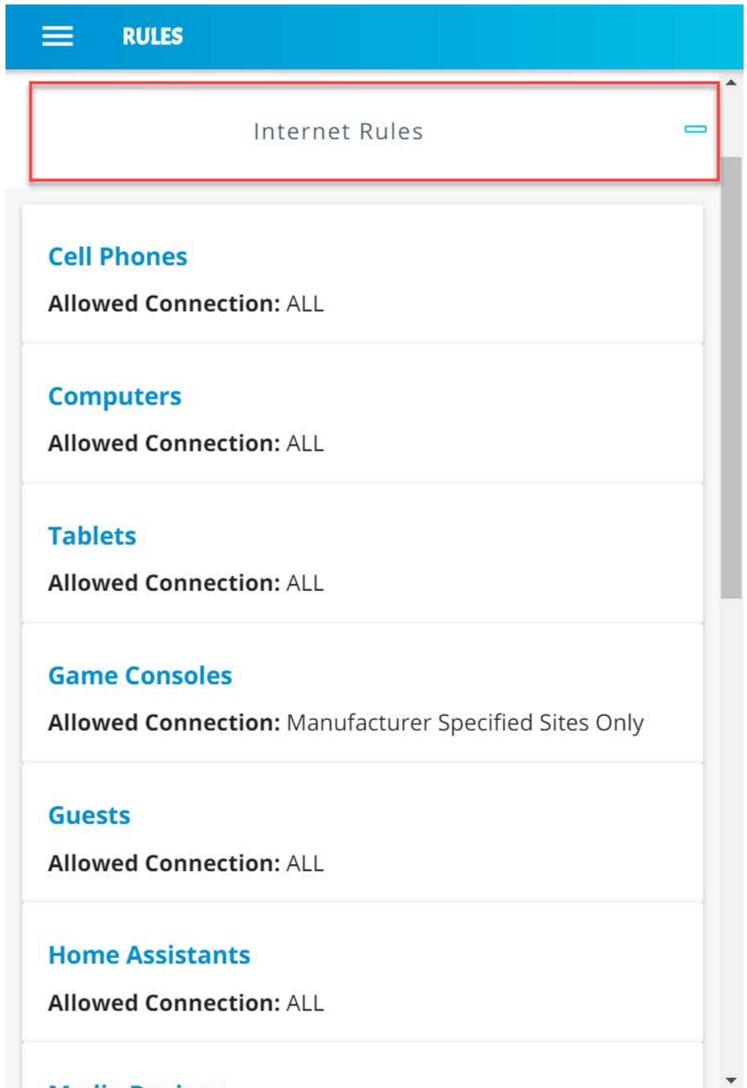
6. Expand the rules that show internet rules for device categories by clicking **Internet Rules:**



7. Scroll down to view the internet rules for the **Smart Appliances** category:



8. Minimize the rules by clicking the **Internet Rules** button:



3.10 GCA Quad9 Threat Signaling in Yikes! Router

This section describes the threat-signaling service provided by GCA in the Yikes! router. This capability should not require configuration because the Quad9 Active Threat Response (Q9Thrt) open-source software should be fully functional when the Yikes! router to connects to the network. Please see the Q9Thrt GitHub page for details on this software: https://github.com/osmud/q9thrt#q9thrt.

3.10.1 GCA Quad9 Threat Signaling in Yikes! Router Overview

The GCA Q9Thrt leverages DNS traffic by using Quad9 DNS services and threat intelligence from ThreatSTOP. As detailed in NIST SP 1800-15B, Q9Thrt is integrated into the Yikes! router and relies on the availability of three third-party services in the cloud: Quad9 DNS service, Quad9 threat API, and ThreatSTOP threat MUD file server. The Yikes! router is integrated with GCA Q9Thrt capabilities implemented, configured, and enabled out of the box.

3.10.2 Configuration Overview

At this implementation, no additional network, software, or hardware configuration was required to enable GCA Q9Thrt on the Yikes! router.

3.10.3 Setup

At this implementation, no additional setup was required to enable GCA Q9Thrt on the Yikes! router. See the Yikes! Router section for details on the router setup.

To take advantage of threat signaling, the Yikes! router uses the Quad9 DNS services for domain name resolution. GCA Quad threat signaling depends upon the Quad9 DNS services to be up and running. The Quad9 threat API must also be available to provide the Yikes! router with information regarding specific threats. In addition, for any given threat that is found, the MUD file server provided by the threat intelligence service that has flagged that threat as potentially dangerous must also be available. These are third-party services that GCA Q9Thrt relies upon to be set up, configured, and available.

It is possible to implement the Q9Thrt feature onto a non-Yikes! router. To integrate the Q9Thrt feature onto an existing router, see the open-source software on GitHub: https://github.com/osmud/q9thrt.

This software was designed for and has been integrated successfully using the OpenWRT platform but has the potential to be integrated into various networking environments. Instructions on how to deploy Q9thrt onto an existing router can be found on https://github.com/osmud/q9thrt#q9thrt.

4 Build 3 Product Installation Guides

This section of the practice guide contains detailed instructions for installing, configuring, and integrating the products used to implement Build 3. For additional details on Build 3's logical and physical architectures, please refer to NIST SP 1800-15B.

4.1 Product Installation

4.1.1 DigiCert Certificates

DigiCert's CertCentral web-based platform allows provisioning and management of publicly trusted X.509 certificates for a variety of purposes. After establishing an account, clients can log in, request, renew, and revoke certificates by using only a browser. For Build 3, the Premium Certificate created in Build 1 was leveraged for signing the MUD files. Additionally, this implementation leveraged a standard SSL certificate to secure the cloud servers. You will need to request standard SSL certificates for each of the servers in your implementation. For this build we requested standard SSL certificates for two servers—the MUD file server and the Micronets service provider cloud server. To request and implement DigiCert certificates, follow the documentation in Build 1's DigiCert Certificates section and subsequent sections.

Once you have received the requested certificates, you can store these on the respective servers in your desired location. For this demonstration, we simply stored them in the workspace directory on the appropriate servers, but it is likely these would be stored in the /usr/lib or /etc/lib directories.

4.1.2 MUD Manager

This section describes the CableLabs MUD manager, which, for this implementation, is a cloud-provided service. This implementation leveraged the nccoe-build-3 branch of CableLabs MUD manager Git release. This service can be hosted by the implementer or another party. This documentation describes setting up your own MUD manager.

4.1.2.1 MUD Manager Overview

The CableLabs MUD manager is used by the Micronets Manager as a utility service to retrieve MUD files from a passed URL, parse the MUD file, and produce device communication restriction declarations that can be passed to the associated Micronets Gateway Service.

This Micronets MUD manager is hosted in the service provider cloud and for this implementation is on the same server as the other Micronets services. The MUD manager is responsible for retrieving MUD files and their associated signature files and executing verification as outlined in the MUD specification. It generates the ACLs for the device based on the MUD file and provides this information to the Micronets Manager.

4.1.2.2 Configuration Overview

The following subsections document the software and network configurations for the MUD manager. Please note that the MUD manager, Micronets Manager, Websocket Proxy, MUD registry, and MSO portal are all implemented on the same server, nccoe-server1.micronets.net.

4.1.2.2.1 Network Configuration

The nccoe-server1.micronets.net server was hosted outside the lab environment on a Linode cloud-hosted Linux server. Its IP address was statically assigned.

4.1.2.2.2 Software Configuration

For this build, the server ran on an Ubuntu 18.04 LTS operating system. The MUD manager runs in its own docker container and is configured to use SSL/TLS encryption.

The following software is required to install, configure, and operate the MUD manager:

- an Ubuntu 18.04 LTS server reachable by the server hosting the Micronets Manager instances and any Micronets gateways
- docker (v18.06 or higher)
- curl
- NGINX

4.1.2.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the MUD manager:

- 4 gigabyte (GB) of RAM
- 50 GB of free disk space

4.1.2.3 Setup

The subsequent sections describe installing, configuring, and confirming general operation for the MUD manager.

4.1.2.3.1 Install and Set Up Dependencies

1. Make directory for downloading micronets-related scripts and packages:

   ```
   mkdir Projects/micronets/
   ```

2. Install **docker, curl,** and **NGINX** by entering the following command:

   ```
   sudo apt install docker curl nginx
   ```

3. Create an NGINX config file for this server (Note: If you are following the architecture for this implementation, all Micronets cloud components will be hosted on this server, and this will be the same config file that will be modified to add routes to the different Micronets services):

   sudo vim /etc/nginx/sites-available/<ServerURL>

   ```
   sudo vim /etc/nginx/sites-available/nccoe-server1.micronets.net
   ```

4. Add the following configuration block to the file and add the path to the certificate and key file received from your DigiCert standard SSL. (Note: Additional locations will be added to this configuration block as you continue to set up the different Micronets services.)

```
server {
      listen 443 ssl;
       listen [::]:443 ssl;

       root /var/www/html;

       index index.html index.htm index.nginx-debian.html;

       server_name nccoe-server1.micronets.net;

       location / {

              try_files $uri $uri/ =404;

       }

       ssl_certificate /home/micronets-dev/Projects/micronets/cert/nccoe-
       server1_micronets_net.crt;

       ssl_certificate_key /home/micronets-dev/Projects/micronets/cert/nccoe-
       server1_micronets_net.key;

}
```

5. Enable the file by creating a link from it to the sites-enabled directory, which NGINX reads from during start-up:

```
sudo ln -s /etc/nginx/sites-available/nccoe-server1.micronets.net
/etc/nginx/sites-enabled/nccoe-server1.micronets.net
```

6. Next, test to make sure that there are no syntax errors in the NGINX files:

```
sudo nginx -t
```

You should see output similar to the following:

```
[sudo] password for micronets-dev:
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

7. If there are no problems, restart NGINX to enable your changes:

```
sudo systemctl restart nginx
```

4.1.2.3.2 Installing MUD Manager

1. Change directory to the Projects/micronets/ folder:

```
cd Projects/micronets/
```

2. Download the management script by executing the following command:

```
curl -O https://raw.githubusercontent.com/cablelabs/micronets-mud-tools/nccoe-build-3/bin/micronets-mud-manager
```

3. Install and execute the management script:

```
sudo install -v -o root -m 755 -D -t /etc/micronets/micronets-mud-manager.d/ micronets-mud-manager
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 -D
-t /etc/micronets/micronets-mud-manager.d/ micronets-mud-manager
[sudo] password for micronets-dev:
install: creating directory '/etc/micronets/micronets-mud-manager.d'
'micronets-mud-manager' -> '/etc/micronets/micronets-mud-manager.d/micronets-mud-man
ager'
```

4. Open the management script to configure it for your implementation by entering the following command:

```
sudo vim /etc/micronets/micronets-mud-manager.d/micronets-mud-manager
```

5. Once the file is opened, modify the default variables in the management script to point to the server hosting our Micronets manager by changing the **DEF_CONTROLLER_ADDRESS** variable:

```
DEF_CONTROLLER_ADDRESS=nccoe-server1.micronets.net
```

```
#!/bin/bash

set -e

# Uncomment this to debug the script
# set -x

shortname="${0##*/}"
longname="MUD manager"
script_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"

DOCKER_CMD="docker"
DEF_IMAGE_LOCATION="community.cablelabs.com:4567/micronets-docker/micronets-mud-manager"
DEF_IMAGE_TAG=nccoe-build-3
DEF_CONTAINER_NAME=micronets-mud-manager-service
DEF_MUD_CACHE_PATH=/var/cache/micronets-mud
DEF_BIND_PORT=8888
DEF_BIND_ADDRESS=127.0.0.1
DEF_CONTROLLER_ADDRESS=nccoe-server1.micronets.net
```

6. Download the docker image by entering the following command:

```
/etc/micronets/micronets-mud-manager.d/micronets-mud-manager docker-pull
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/micronets-mud-manag
er docker-pull
Pulling docker image from community.cablelabs.com:4567/micronets-docker/micronets-mu
d-manager:nccoe-build-3
nccoe-build-3: Pulling from micronets-docker/micronets-mud-manager
8ec398bc0356: Already exists
3db8034857a2: Already exists
ba5f9fbce982: Already exists
5ab2a4e50325: Already exists
65fe15d554b2: Already exists
1e57fecf78cc: Already exists
d0f7704112f2: Pull complete
5f15715d4210: Pull complete
074bf77546db: Pull complete
Digest: sha256:273f455fb3482c5f6089c72491488528df69b0113b676019b88d6ef66dbb9402
Status: Downloaded newer image for community.cablelabs.com:4567/micronets-docker/mic
ronets-mud-manager:nccoe-build-3
community.cablelabs.com:4567/micronets-docker/micronets-mud-manager:nccoe-build-3
```

7. Next, set up the MUD cache directory by using the management script and entering the following command:

```
sudo /etc/micronets/micronets-mud-manager.d/micronets-mud-manager setup-cache-
dir
```

8. Last, start the MUD manager by entering the following command to run the docker container:

```
/etc/micronets/micronets-mud-manager.d/micronets-mud-manager docker-run
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/micronets-mud-manag
er.d/micronets-mud-manager docker-run
Starting container "micronets-mud-manager-service" from community.cablelabs.com:4567
/micronets-docker/micronets-mud-manager:nccoe-build-3 (on 127.0.0.1:8888)
06be09836aa016a02c3709a776079f432b9aad4946f6b1a3311e0f15fff2c2ac
```

9. Verify that the MUD manager is running by using the following command and reviewing the logs:

```
/etc/micronets/micronets-mud-manager.d/micronets-mud-manager docker-logs
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/micronets-mud-manag
er.d/micronets-mud-manager docker-logs
Showing logs for container "micronets-mud-manager-service"
2020-05-05T15:56:13.635640286Z 2020-05-05 15:56:13,635 micronets-mud-manager: INFO B
ind address: 0.0.0.0
2020-05-05T15:56:13.635942956Z 2020-05-05 15:56:13,635 micronets-mud-manager: INFO B
ind port: 8888
2020-05-05T15:56:13.636184595Z 2020-05-05 15:56:13,636 micronets-mud-manager: INFO C
A path: /etc/ssl/certs
2020-05-05T15:56:13.636417304Z 2020-05-05 15:56:13,636 micronets-mud-manager: INFO A
dditional CA certs: None
2020-05-05T15:56:13.636626114Z 2020-05-05 15:56:13,636 micronets-mud-manager: INFO M
UD cache directory: /mud-cache-dir
2020-05-05T15:56:13.636794154Z 2020-05-05 15:56:13,636 micronets-mud-manager: INFO C
ontroller: None
2020-05-05T15:56:13.637894702Z 2020-05-05 15:56:13,637 asyncio: DEBUG Using selector
: EpollSelector
2020-05-05T15:56:13.641757712Z Running on https://0.0.0.0:8888 (CTRL + C to quit)
2020-05-05T15:56:13.641778932Z [2020-05-05 15:56:13,641] ASGI Framework Lifespan err
or, continuing without Lifespan support
2020-05-05T15:56:13.641931411Z 2020-05-05 15:56:13,641 quart.serving: WARNING ASGI F
ramework Lifespan error, continuing without Lifespan support
```

10. Set up a proxy pass to the MUD manager by adding the following entry to the NGINX `server` block:

    a. Open the NGINX sites-available file for the server:

    ```
    sudo vim /etc/nginx/sites-available/nccoe-server1.micronets.net
    ```

    b. Add the following location to the server block:

    ```
    location /micronets/mud-manager/ {
           proxy_pass     http://localhost:8888/;
    }
    ```

```
server {
        listen 443 ssl;
        listen [::]:443 ssl;
        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;
        server_name nccoe-server1.micronets.net;

        location / {
                try_files $uri $uri/ =404;
        }

        location /micronets/mud-manager/ {
        proxy_pass      http://localhost:8888/;
        }

        ssl_certificate /home/micronets-dev/Projects/micronets/cert/nccoe-server1_micronets_n
et.crt;
        ssl_certificate_key /home/micronets-dev/Projects/micronets/cert/nccoe-server1_microne
ts_net.key;

}
~
```

11. Reload the NGINX server by executing the following command:

```
sudo nginx -s reload
```

4.1.2.3.3 Operation

In this section, we test general operation of the MUD manager.

1. Test the MUD manager by retrieving a MUD file and using the following command (replace the MUD manager URL with the URL you created in Section 4.1.2.3.1):

```
curl -q -X POST -H "Content-Type: application/json" \
 https://nccoe-server1.micronets.net/micronets/mud-manager/getMudFile \
 -d '{"url": "https://alpineseniorcare.com/micronets-mud/ciscopi.json"}'
```

   You should see the MUD file requested printed in the terminal:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ curl -q -X POST -H "Content-Type:
application/json" \
> https://nccoe-server1.micronets.net/micronets/mud-manager/getMudFile \
[> -d '{"url": "https://alpineseniorcare.com/micronets-mud/ciscopi.json"}'        ]
{
    "ietf-mud:mud": {
        "mud-version": 1,
        "mud-url": "https://mudfileserver/ciscopi2",
        "last-update": "2018-12-05T19:42:01+00:00",
        "cache-validity": 24,
        "is-supported": true,
        "systeminfo": "ingress/egress ",
        "from-device-policy": {
            "access-lists": {
                "access-list": [
                    {
                        "name": "mud-81726-v4fr"
                    }
                ]
            }
        },
        "to-device-policy": {
            "access-lists": {
                "access-list": [
                    {
                        "name": "mud-81726-v4to"
                    }
                ]
            }
        }
    },
```

2. Check the MUD file cache directory to confirm that the MUD file requested is stored in the cache:

```
ls -1  /var/cache/micronets-mud/
```

You should see the MUD file you just requested stored in the cache directory:

```
[micronets-dev@nccoe-server1:~/Projects/micronets$ ls -l /var/cache/micronets-mud/    ]
total 12
-rw-r--r-- 1 root root 6307 May  5 19:31 alpineseniorcare.com_micronets-mud_ciscopi.
json
-rw-r--r-- 1 root root   49 May  5 19:31 alpineseniorcare.com_micronets-mud_ciscopi.
json.md
```

3. Now that the MUD manager has successfully retrieved its first MUD file, you can clear the cache by entering the following command:

```
/etc/micronets/micronets-mud-manager.d/micronets-mud-manager clear-cache-dir
```

You should see the following output once the command above has been executed:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/micronets-mud-manag
er.d/micronets-mud-manager clear-cache-dir
removed '/var/cache/micronets-mud/alpineseniorcare.com_micronets-mud_ciscopi.json'
removed '/var/cache/micronets-mud/alpineseniorcare.com_micronets-mud_ciscopi.json.md
'
```

4. To output a list of additional docker commands supported by the management script, you can execute the following command:

   `/etc/micronets/micronets-mud-manager.d/micronets-mud-manager –`

   You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ /etc/micronets/micronets-mud-manager.d/micronets-mud-manager --
micronets-mud-manager: error: Unrecognized option: --

Usage: micronets-mud-manager <operation>

   operation can be one of:

     docker-pull: Download the micronets-mud-manager docker image
     docker-run: Create and start the micronets-mud-manager docker container
     docker-run-interactive: Start a shell to run micronets-mud-manager (for debugging)
     docker-status: Show the status of the micronets-mud-manager docker container
     docker-kill: Kill the micronets-mud-manager docker container
     docker-restart: Restart the micronets-mud-manager docker container
     docker-logs: Show the logs for micronets-mud-manager docker container
     docker-trace: Watch the logs for the micronets-mud-manager docker container
     docker-address: Print the IP addresses for the micronets-mud-manager docker container
     docker-env: List the environment variables for the micronets-mud-manager docker container
     setup-cache-dir: Create the MUD cache directory
     clear-cache-dir: Clear the MUD cache

   [--docker-image <docker image ID>
       (default "community.cablelabs.com:4567/micronets-docker/micronets-mud-manager")
   [--docker-image-tag <docker image tag>
       (default "nccoe-build-3")
   [--docker-name <docker name to assign>
       (default "micronets-mud-manager-service")
   [--mud-cache-path <mud cache directory to mount in container>
       (default "/var/cache/micronets-mud")
   [--bind-address <address to bind micronets-mud-manager to>
       (default "127.0.0.1")
   [--bind-port <port to bind micronets-mud-manager to>
       (default "8888")
   [--controller-address <address of the MUD controller>
       The address to use for any MUD "controller" references
       (default "nccoe-server1.micronets.net")
```

4.1.3 MUD File Server

This section describes the CableLabs MUD file server, which is a cloud-hosted service. The Build 3 implementation is designed a bit differently from the other three builds insofar as it requires a MUD registry to be incorporated in the solution as described in Volume B. We describe the MUD registry in this section of the documentation.

4.1.3.1 MUD File Server Overview

In the absence of a commercial MUD file server for use in this project, the NCCoE leveraged a Linode cloud-hosted Linux server to create the MUD file server that is accessible via the internet. This file server stores the MUD files along with their corresponding signature files for the IoT devices used in the project. Upon receiving a GET request for the MUD files and signatures, it serves the request to the MUD manager by using https.

4.1.3.2 Configuration Overview

The following subsections document the software and network configurations for the MUD file server.

4.1.3.2.1 Network Configuration

This server was hosted outside the lab environment on a Linode cloud-hosted Linux server. Its IP address was statically assigned.

4.1.3.2.2 Software Configuration

For this build, the server ran on an Ubuntu 18.04 LTS operating system. The MUD files and signatures were hosted by an NGINX web server and configured to use SSL/TLS encryption.

4.1.3.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the MUD file server:

- 4 GB of RAM
- 50 GB of free disk space

4.1.3.3 Setup

4.1.3.3.1 NGINX Web Server

1. Update your local package index by entering the following command:

   ```
   sudo apt update
   ```

2. Install NGINX by entering the following command:

   ```
   sudo apt install nginx
   ```

3. Create the directory where the MUD files will be stored on the MUD file server as follows:

   ```
   sudo mkdir -p /var/www/nccoe-server2.micronets.net/html/micronets-mud/
   ```

4. Create an NGINX config file for this server (Note: If you are following the architecture for this implementation, all Micronets cloud components will be hosted on this server, and this will be the same config file that will be modified to add routes to the different Micronets services):

   sudo vim /etc/nginx/sites-available/<ServerURL>

   Below is an example of this command:

   ```
   sudo vim /etc/nginx/sites-available/nccoe-server2.micronets.net
   ```

5. Add the following configuration block to the file (Note: Additional locations will be added to this configuration block as you continue to set up the different Micronets services):

   ```
   server {
       listen 443 ssl;
       listen [::]:443 ssl;
       root /var/www/nccoe-server2.micronets.net/html;
       index index.html index.htm index.nginx-debian.html;
       server_name nccoe-serve2.micronets.net;
        location / {
          # First attempt to serve request as file, then
          # as directory, then fall back to displaying a 404.
          try_files $uri $uri/ =404;
      }
      if ($scheme != "https") {
          return 301 https://$host$request_uri;
      }
      ssl_certificate /home/micronets-dev/Projects/micronets/cert/nccoe-
      server2_micronets_net.crt;
      ssl_certificate_key /home/micronets-dev/Projects/micronets/cert/nccoe-
      server2_micronets_net.key;

   include /etc/nginx/micronets-subscriber-forwards/*.conf;
   }
   ```

6. Enable the file by creating a link from it to the sites-enabled directory, which NGINX reads from during startup:

```
sudo ln -s /etc/nginx/sites-available/nccoe-server2.micronets.net \
/etc/nginx/sites-enabled/nccoe-server2.micronets.net
```

7. Next, test to make sure that there are no syntax errors in any of your NGINX files:

```
sudo nginx -t
```

You should see output similar to the following:

```
[sudo] password for micronets-dev:
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

8. If there are no problems, restart NGINX to enable your changes:

```
sudo systemctl restart nginx
```

4.1.3.3.2 MUD File Creation and Signing

To create MUD files for MUD-capable IoT devices, please follow the instructions in Build 1's MUD File Server. Once MUD files and signature files are created, they can be stored in the web server directory created on the MUD file server in the previous section.

4.1.4 Micronets Gateway

This section describes the CableLabs Micronets Gateway, which, for this implementation, is an on-premise component. This implementation leveraged the nccoe-build-3 tagged version of CableLabs Micronets Gateway Git release. This documentation describes setting up your own Micronets gateway.

4.1.4.1 Micronets Gateway Overview

The Micronets Gateway establishes a connection to the Micronets Manager through the Websocket Proxy and receives traffic flow rules and other configuration information that it applies and enforces. Additionally, the Micronets Gateway supports wired and wireless connections, MUD-defined ACLs, and DPP onboarding.

4.1.4.2 Configuration Overview

The following subsections document the software and network configurations for the Micronets Gateway.

4.1.4.2.1 Network Configuration

Implementation of a Micronets gateway requires an internet source such as a digital subscriber line (DSL) or cable modem.

4.1.4.2.2 Software Configuration

The Micronets Gateway runs an Ubuntu 16.04 LTS server, which can support all the software dependencies and packages that will be installed during setup.

4.1.4.2.3 Hardware Configuration

For this implementation, we leveraged a Shuttle XPC slim DH170 with the following specs:

- x86_64 processor (Intel or AMD)
- at least two Ethernet ports
- wireless adapter with a QUALCOMM Atheros AR9271 chipset
- 2 GB or higher of RAM

4.1.4.3 Setup

4.1.4.3.1 Install Dependencies

1. If Micronets is already installed and running, you should stop the services first by executing the following commands:

```
sudo systemctl stop micronets-gw.service

sudo systemctl stop micronets-hostapd.service
```

2. Update your local package index by entering the following command:

```
sudo apt-get update
```

You should see the following output from this command:

```
micronets-dev@nccoe-gw:~$ sudo apt-get update
Hit:1 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [850 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [1,130 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main i386 Packages [652 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 Packages [912 kB]
Get:9 http://security.ubuntu.com/ubuntu xenial-security/main amd64 DEP-11 Metadata [74.9 kB]
Get:10 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons [84.1 kB]
Get:11 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 DEP-11 Metadata [124 kB]
Get:12 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 DEP-11 Metadata [2,464 B]
Get:13 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 DEP-11 Metadata [322 kB]
Get:14 http://us.archive.ubuntu.com/ubuntu xenial-updates/main DEP-11 64x64 Icons [235 kB]
Get:15 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 DEP-11 Metadata [276 kB]
Get:16 http://us.archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 DEP-11 Metadata [5,980 B]
Get:17 http://us.archive.ubuntu.com/ubuntu xenial-backports/main amd64 DEP-11 Metadata [3,328 B]
Get:18 http://us.archive.ubuntu.com/ubuntu xenial-backports/universe amd64 DEP-11 Metadata [5,320 B]
Fetched 5,001 kB in 1s (3,477 kB/s)
Reading package lists... Done
```

3. Install the **python-pip, virtualenv, dnsmasq, python-six,** and **libnl-route-3-200** packages by executing the following command:

```
sudo apt-get -y install python-pip virtualenv dnsmasq python-six libnl-route-3-200
```

If the packages are not already installed, you should see the following output from this command:

```
micronets-dev@nccoe-gw:~$ sudo apt-get -y install python-pip virtualenv dnsmasq pyth
on-six libnl-route-3-200
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-six is already the newest version (1.10.0-3).
libnl-route-3-200 is already the newest version (3.2.27-1ubuntu0.16.04.1).
dnsmasq is already the newest version (2.75-1ubuntu0.16.04.5).
python-pip is already the newest version (8.1.1-2ubuntu0.4).
virtualenv is already the newest version (15.0.1+ds-3ubuntu1).
The following packages were automatically installed and are no longer required:
  linux-headers-4.15.0-45 linux-headers-4.15.0-45-generic linux-headers-4.15.0-70
  linux-headers-4.15.0-70-generic linux-headers-4.15.0-72
  linux-headers-4.15.0-72-generic linux-headers-4.15.0-74
  linux-headers-4.15.0-74-generic linux-headers-4.15.0-76
  linux-headers-4.15.0-76-generic linux-headers-4.15.0-88
  linux-headers-4.15.0-88-generic linux-image-4.15.0-45-generic
  linux-image-4.15.0-70-generic linux-image-4.15.0-72-generic
  linux-image-4.15.0-74-generic linux-image-4.15.0-76-generic
  linux-image-4.15.0-88-generic linux-modules-4.15.0-45-generic
  linux-modules-4.15.0-70-generic linux-modules-4.15.0-72-generic
  linux-modules-4.15.0-74-generic linux-modules-4.15.0-76-generic
  linux-modules-4.15.0-88-generic linux-modules-extra-4.15.0-45-generic
  linux-modules-extra-4.15.0-70-generic linux-modules-extra-4.15.0-72-generic
  linux-modules-extra-4.15.0-74-generic linux-modules-extra-4.15.0-76-generic
  linux-modules-extra-4.15.0-88-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 91 not upgraded.
```

4. Install openvswitch version 2.9.2 and its dependencies from the CableLabs micronets-gw github repository by executing the following for loop:

```
for package in libopenvswitch_2.9.2-1_amd64.deb \
               openvswitch-common_2.9.2-1_amd64.deb \
               openvswitch-switch_2.9.2-1_amd64.deb ;
do curl -L -O https://github.com/cablelabs/micronets-gw/releases/down-
load/1.0.55/${package};
sudo dpkg -i ${package};
done
```

You should see the following output from this command:

```
micronets-dev@nccoe-gw:~$ for package in libopenvswitch_2.9.2-1_amd64.deb openvswitc
h-common_2.9.2-1_amd64.deb openvswitch-switch_2.9.2-1_amd64.deb ;
> do curl -L -O https://github.com/cablelabs/micronets-gw/releases/download/1.0.55/$
{package};
> sudo dpkg -i ${package} ;
> done
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   645  100   645    0     0   1734      0 --:--:-- --:--:-- --:--:--  1733
100 1141k  100 1141k    0     0  1590k      0 --:--:-- --:--:-- --:--:-- 1590k
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack libopenvswitch_2.9.2-1_amd64.deb ...
Unpacking libopenvswitch:amd64 (2.9.2-1) over (2.9.2-1) ...
Setting up libopenvswitch:amd64 (2.9.2-1) ...
Processing triggers for libc-bin (2.23-0ubuntu11) ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   649  100   649    0     0   1905      0 --:--:-- --:--:-- --:--:--  1903
100  161k  100  161k    0     0   277k      0 --:--:-- --:--:-- --:--:--  277k
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack openvswitch-common_2.9.2-1_amd64.deb ...
Unpacking openvswitch-common (2.9.2-1) over (2.9.2-1) ...
Setting up openvswitch-common (2.9.2-1) ...
Processing triggers for man-db (2.7.5-1) ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   649  100   649    0     0   2284      0 --:--:-- --:--:-- --:--:--  2285
100  253k  100  253k    0     0   475k      0 --:--:-- --:--:-- --:--:--  475k
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack openvswitch-switch_2.9.2-1_amd64.deb ...
Unpacking openvswitch-switch (2.9.2-1) over (2.9.2-1) ...
Setting up openvswitch-switch (2.9.2-1) ...
Processing triggers for systemd (229-4ubuntu21.27) ...
Processing triggers for ureadahead (0.100.0-19) ...
ureadahead will be reprofiled on next reboot
Processing triggers for man-db (2.7.5-1) ...
```

5. Install Python version 3.6 and its dependencies from the CableLabs micronets-gw github repository by executing the following for loop:

```
for package in libpython3.6-minimal_3.6.5-5.16.04.york1_amd64.deb \
               libpython3.6-stdlib_3.6.5-5.16.04.york1_amd64.deb \
               python3.6-minimal_3.6.5-5.16.04.york1_amd64.deb \
               python3.6_3.6.5-5.16.04.york1_amd64.deb ;
do curl -L -O https://github.com/cablelabs/micronets-gw/releases/down-
load/1.0.55/${package};
```

You should see the following output from this command:

```
micronets-dev@nccoe-gw:~$ for package in libpython3.6-minimal_3.6.5-5.16.04.york1_am
d64.deb libpython3.6-stdlib_3.6.5-5.16.04.york1_amd64.deb python3.6-minimal_3.6.5-5.
16.04.york1_amd64.deb python3.6_3.6.5-5.16.04.york1_amd64.deb ;
> do curl -L -O https://github.com/cablelabs/micronets-gw/releases/download/1.0.55/$
{package};
> sudo dpkg -i ${package} ;
> done
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   663  100   663    0     0   1762      0 --:--:-- --:--:-- --:--:--  1763
100  560k  100  560k    0     0   727k      0 --:--:-- --:--:-- --:--:--  727k
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack libpython3.6-minimal_3.6.5-5.16.04.york1_amd64.deb ...
Unpacking libpython3.6-minimal:amd64 (3.6.5-5~16.04.york1) over (3.6.5-5~16.04.york1
) ...
Setting up libpython3.6-minimal:amd64 (3.6.5-5~16.04.york1) ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   662  100   662    0     0   2271      0 --:--:-- --:--:-- --:--:--  2274
100 1942k  100 1942k    0     0  2566k      0 --:--:-- --:--:-- --:--:-- 10.3M
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack libpython3.6-stdlib_3.6.5-5.16.04.york1_amd64.deb ...
Unpacking libpython3.6-stdlib:amd64 (3.6.5-5~16.04.york1) over (3.6.5-5~16.04.york1)
 ...
Setting up libpython3.6-stdlib:amd64 (3.6.5-5~16.04.york1) ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   660  100   660    0     0   2396      0 --:--:-- --:--:-- --:--:--  2391
100 1672k  100 1672k    0     0  2216k      0 --:--:-- --:--:-- --:--:-- 2216k
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack python3.6-minimal_3.6.5-5.16.04.york1_amd64.deb ...
Unpacking python3.6-minimal (3.6.5-5~16.04.york1) over (3.6.5-5~16.04.york1) ...
Setting up python3.6-minimal (3.6.5-5~16.04.york1) ...
Processing triggers for man-db (2.7.5-1) ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   652  100   652    0     0   2252      0 --:--:-- --:--:-- --:--:--  2256
100  224k  100  224k    0     0   402k      0 --:--:-- --:--:-- --:--:-- 1000k
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack python3.6_3.6.5-5.16.04.york1_amd64.deb ...
Unpacking python3.6 (3.6.5-5~16.04.york1) over (3.6.5-5~16.04.york1) ...
Setting up python3.6 (3.6.5-5~16.04.york1) ...
Processing triggers for gnome-menus (3.13.3-6ubuntu3.1) ...
Processing triggers for desktop-file-utils (0.22-1ubuntu5.2) ...
Processing triggers for bamfdaemon (0.5.3~bzr0+16.04.20180209-0ubuntu1) ...
Rebuilding /usr/share/applications/bamf-2.index...
Processing triggers for mime-support (3.59ubuntu1) ...
Processing triggers for man-db (2.7.5-1) ...
```

4.1.4.3.2 Install Micronets Packages

1. Enter the following command to download the Micronets hostapd package:

```
curl -L -O https://github.com/cablelabs/micronets-gw/releases/down-
load/1.0.55/micronets-hostapd-1.0.21.deb
```

You should see output similar to the following:

```
micronets-dev@nccoe-gw:~$ curl -L -O https://github.com/cablelabs/micronets-gw/relea
ses/download/1.0.55/micronets-hostapd-1.0.21.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   641  100   641    0     0   2021      0 --:--:-- --:--:-- --:--:--  2022
100 1981k  100 1981k    0     0  2363k      0 --:--:-- --:--:-- --:--:-- 11.5M
```

2. Enter the following command to de-package the Micronets hostapd package:

```
sudo dpkg -i micronets-hostapd-1.0.21.deb
```

You should see output similar to the following:

```
micronets-dev@nccoe-gw:~$ sudo dpkg -i micronets-hostapd-1.0.21.deb
(Reading database ... 431746 files and directories currently installed.)
Preparing to unpack micronets-hostapd-1.0.21.deb ...
Apr 20 12:22:00 nccoe-gw mnhostapd-prerm-111T122200: PRERM: mnhostapd-prerm-111T1222
00
Apr 20 12:22:00 nccoe-gw mnhostapd-prerm-111T122200: Stopping micronets-hostapd serv
ice.
Apr 20 12:22:00 nccoe-gw mnhostapd-pre-111T122200: PREINSTALL: mnhostapd-pre-111T122
200
Unpacking micronets-hostapd (1.0.21) over (1.0.16) ...
Setting up micronets-hostapd (1.0.21) ...
Upgrading from version 1.0.16
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: POSTINSTALL: mnhostapd-post-111T
122200
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: Installing micronets-hostapd ser
vice.
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: Reloading service files.
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: Completed installation.
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: NOTE: Make sure to configure /op
t/micronets-hostapd/lib/hostapd.conf for your system
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: To start hostapd via systemd:
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200:     systemctl start micronets-ho
stapd.service
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: To start hostapd manually:
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200:     /opt/micronets-hostapd/bin/h
ostapd /opt/micronets-hostapd/lib/hostapd.conf
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200: To set hostapd for automatic sta
rtup:
Apr 20 12:22:00 nccoe-gw mnhostapd-post-111T122200:     systemctl enable micronets-h
ostapd.service
```

3. Enter the following command to download the Micronets Gateway package:

```
curl -L -O https://github.com/cablelabs/micronets-gw/releases/down-
load/1.0.55/micronets-gw-1.0.55.deb
```

You should see output similar to the following:

```
micronets-dev@nccoe-gw:~$ curl -L -O https://github.com/cablelabs/micronets-gw/relea
ses/download/1.0.55/micronets-gw-1.0.55.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   636  100   636    0     0   1745      0 --:--:-- --:--:-- --:--:--  1747
100 49784  100 49784    0     0  86219      0 --:--:-- --:--:-- --:--:-- 86219
```

4. Enter the following command to install the Micronets hostapd package:

```
sudo dpkg -i micronets-gw-1.0.55.deb
```

After a bit of a delay, you should see output similar to the following:

```
Apr 20 12:24:21 nccoe-gw mngw-post-111T122420: Installing micronets-gw service.
Apr 20 12:24:21 nccoe-gw mngw-post-111T122420: Reloading service files.
Apr 20 12:24:21 nccoe-gw mngw-post-111T122420: Enabling micronets-gw service.
Apr 20 12:24:21 nccoe-gw mngw-post-111T122420: Starting micronets-gw service.
```

5. Enable autostart for the Micronets hostapd service by entering the following command:

```
sudo systemctl enable micronets-hostapd.service
```

6. Enable autostart for the Micronets Gateway Service by entering the following command:

```
sudo systemctl enable micronets-gw.service
```

7. Start the Micronets hostapd service by entering the following command:

```
sudo systemctl start micronets-hostapd.service
```

8. Start the Micronets Gateway Service by entering the following command:

```
sudo systemctl start micronets-gw.service
```

9. Verify that the gateway service started successfully by running the following command:

```
sudo systemctl status micronets-gw.service
```

10. Verify that the Micronets hostapd service started successfully by running the following command:

```
sudo systemctl status micronets-hostapd.service
```

CableLabs documentation notes that installing the micronets-gw package should produce the following results:

- installation of the Micronets Gateway Service in the /opt/micronets-gw directory
- installation of the ifup/down and dnsmasq extension scripts for configuration of openvswitch and the micronets-gw service via /etc/network/interfaces
- installation of a sample/etc/network/interfaces file in /opt/micronets-gw/doc/interfaces.sample
- installation and start of the micronets-gw-service systemd service

4.1.5 IoT Devices

This section provides configuration details for the Linux-based IoT development kits used in the build, which can be onboarded via DPP. It also provides information regarding a basic IoT application used to test the MUD process.

4.1.5.1 IoT Devices Overview

Build 3, like the other builds in this project, leverages the Raspberry Pi devkit with capabilities developed to make these devices both MUD- and DPP-capable. The Raspberry Pi runs the Raspbian 9 OS and is provisioned with one bootstrapping public/private key pair during device setup. The Micronets Proto-Pi software developed by CableLabs in combination with the added hardware outlined in the configuration section adds DPP capability to these devices. There are two onboarding mechanisms called *modes* supported by the Micronets Proto-Pi software: DPP mode and clinic mode. The clinic mode provides an onboarding mechanism via automated installation of Wi-Fi security certificates, and the DPP mode provides QR code–based device onboarding. For this implementation, we only describe setting up and leveraging the Micronets Proto-Pi software in DPP mode. If you would like to leverage the clinic mode of this software, follow the documentation provided by CableLabs: https://github.com/cablelabs/micronets-pi3/blob/nccoe-build-3/README.md#Installation.

4.1.5.2 Configuration Overview

The following subsections document the software and network configurations for the Micronets Proto-Pi device.

4.1.5.2.1 Network Configuration

The following network configurations are required to install, configure, and operate the Micronets Proto-Pi device:

- wired network connection to a separate access point that provides both initial internet access to self-register the device and remote management access to the device during setup

4.1.5.2.2 Software Configuration

The following software is required to install, configure, and operate the Micronets Proto-Pi device:

- tool for flashing images to Secure Digital (SD) card (This implementation leveraged balenaEtcher: https://www.balena.io/etcher/.)
- latest Raspbian image from:
  - CableLabs at the following link (This image has Secure Shell (SSH) and Visual (vi) preinstalled): https://www.dropbox.com/s/37ygauo02ltxirf/raspbian-buster-ssh-updates.zip?dl=0
  - Or you can download the latest Buster distribution and install packages yourself from the following link: https://www.raspberrypi.org/downloads/raspbian/

4.1.5.2.3 Hardware Configuration

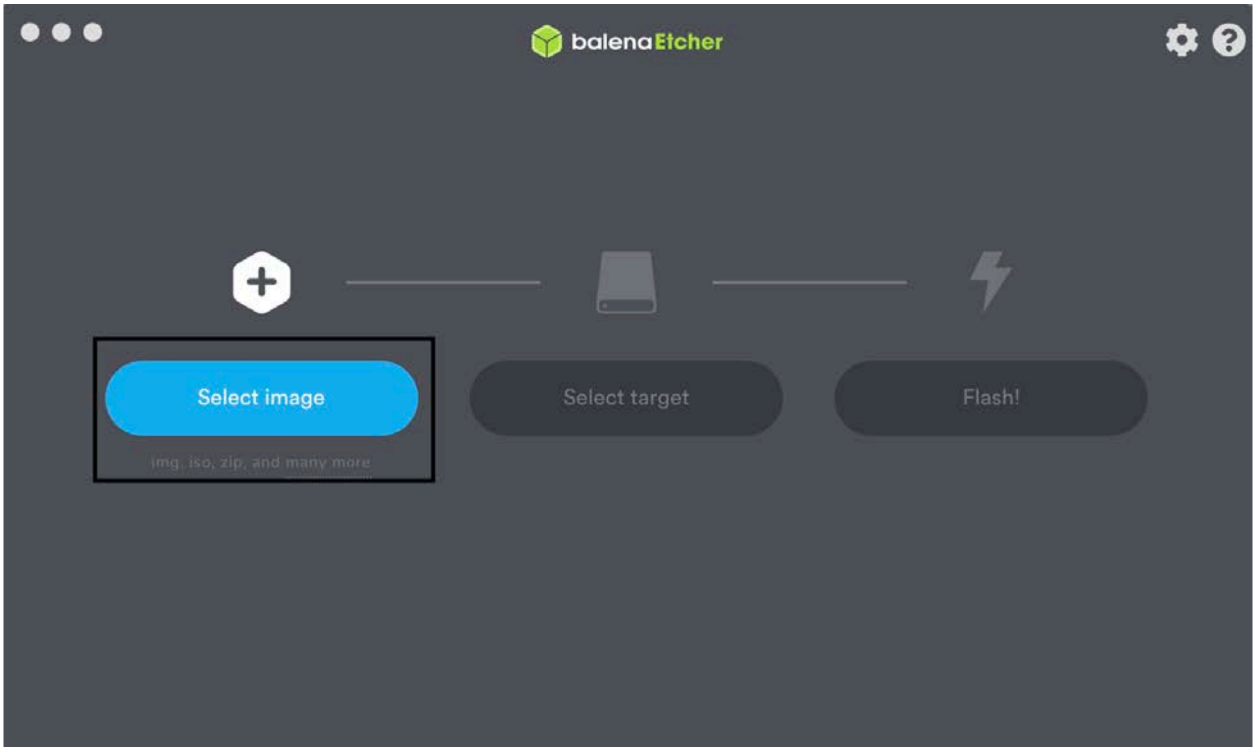
The following hardware is required to install, configure, and operate the Micronets Proto-Pi device:

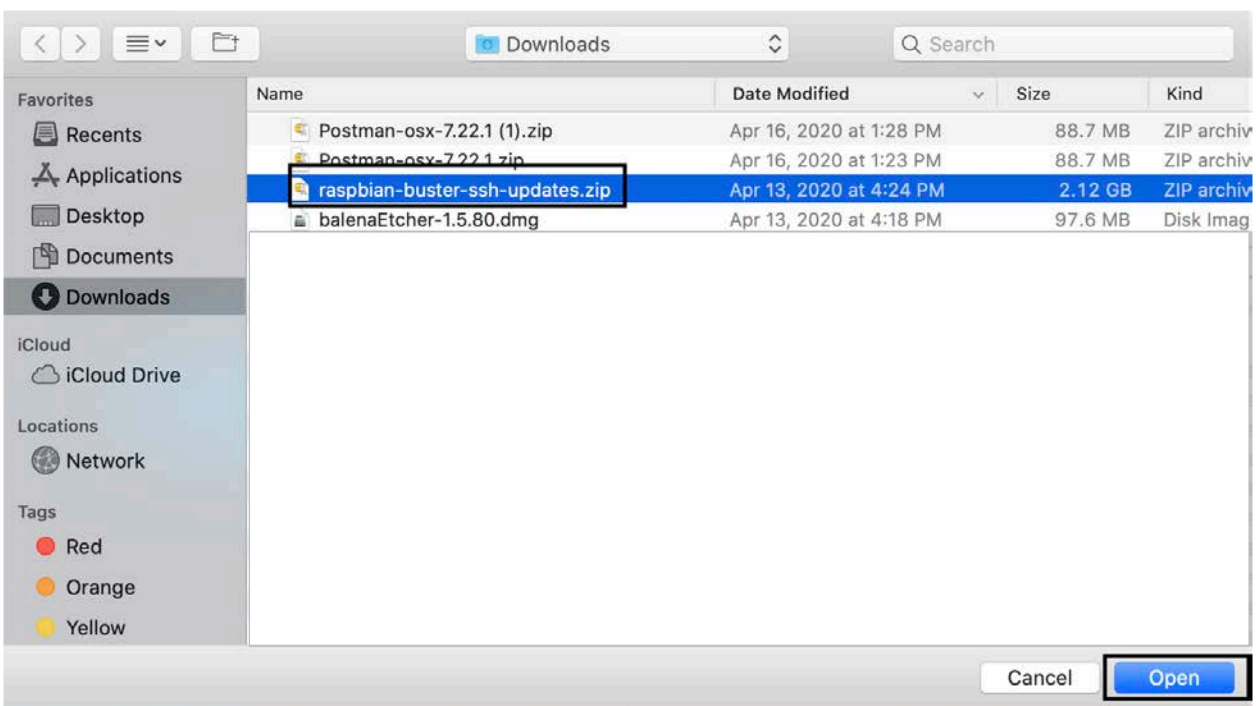
- Raspberry Pi (version 3B+)
- SD card
- Alfa adapter
- Ethernet cable

4.1.5.3 Setup

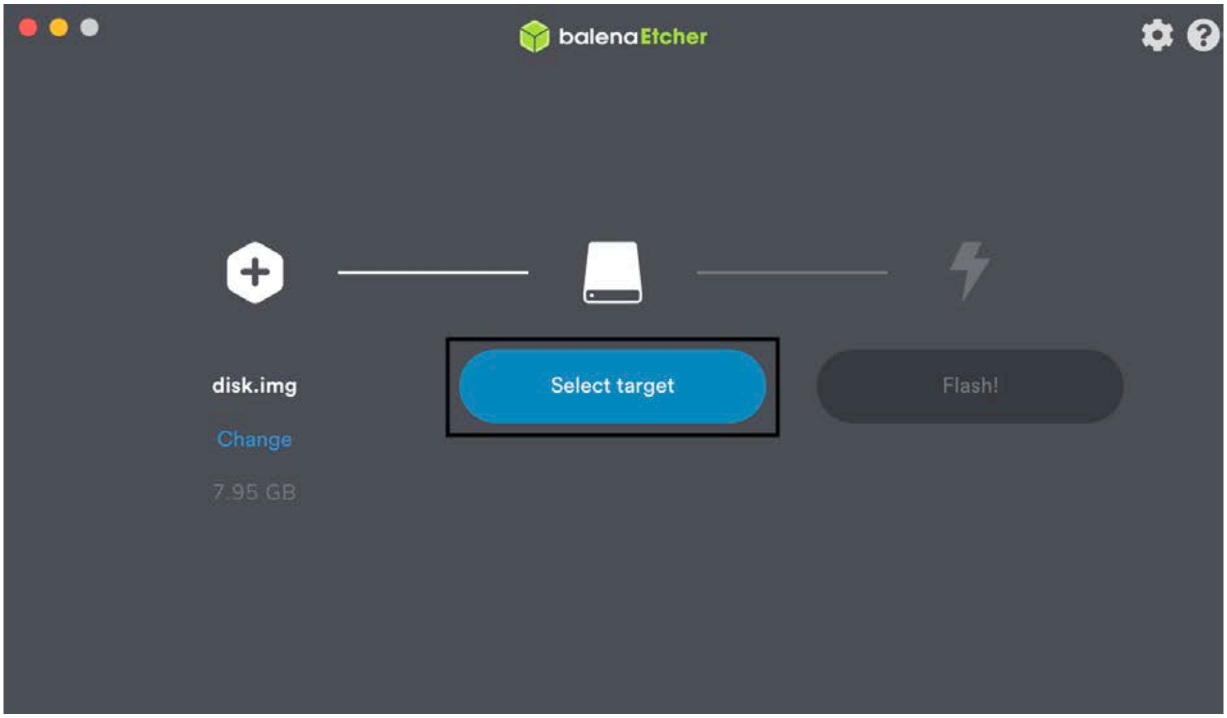
4.1.5.3.1 Install Dependencies

1. Connect the SD card to your computer.
2. Open balenaEtcher (or whatever tool you have downloaded for flashing SD cards).
3. Click **Select image,** and select the Raspbian image you downloaded:

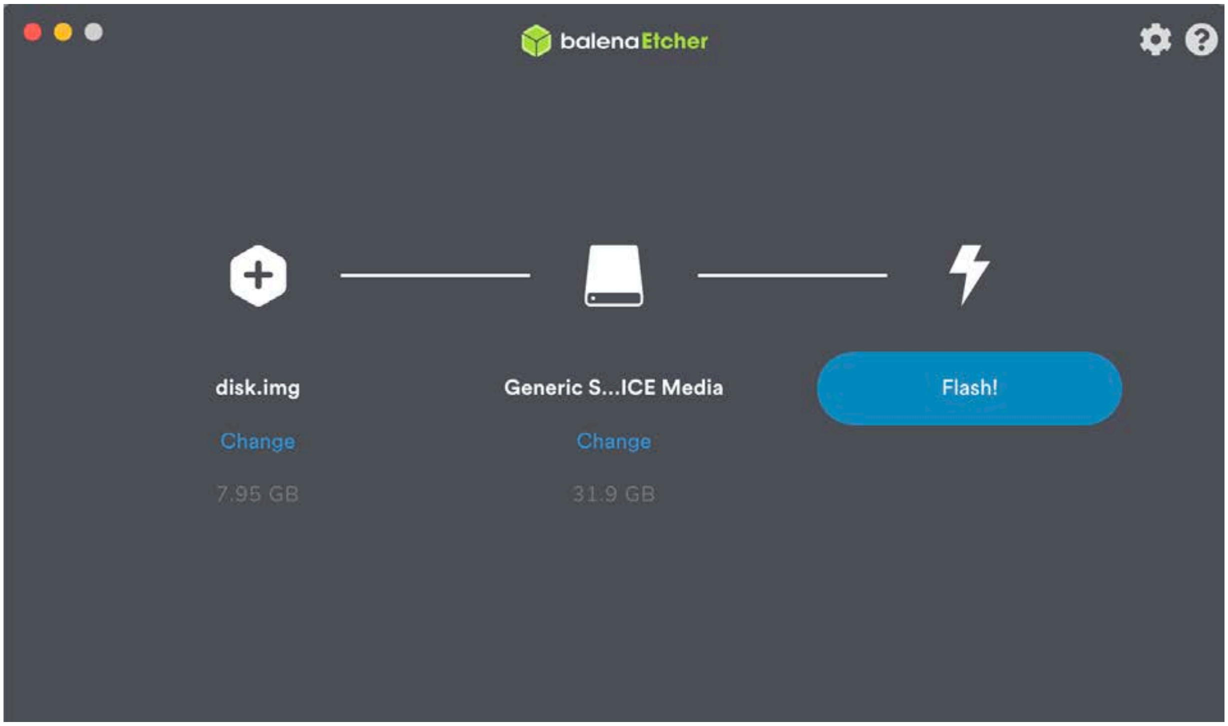

balenaEtcher
Select image
img, iso, zip, and many more
Select target
Flash!


Downloads
Search
Favorites
Recents
Applications
Desktop
Documents
Downloads
iCloud
iCloud Drive
Locations
Network
Tags
Red
Orange
Yellow
Name
Date Modified
Size
Kind
Postman-osx-7.22.1 (1).zip
Apr 16, 2020 at 1:28 PM
88.7 MB
Postman-osx-7.22.1.zip
Apr 16, 2020 at 1:23 PM
88.7 MB
raspbian-buster-ssh-updates.zip
Apr 13, 2020 at 4:24 PM
2.12 GB
balenaEtcher-1.5.80.dmg
Apr 13, 2020 at 4:18 PM
97.6 MB
Cancel
Open

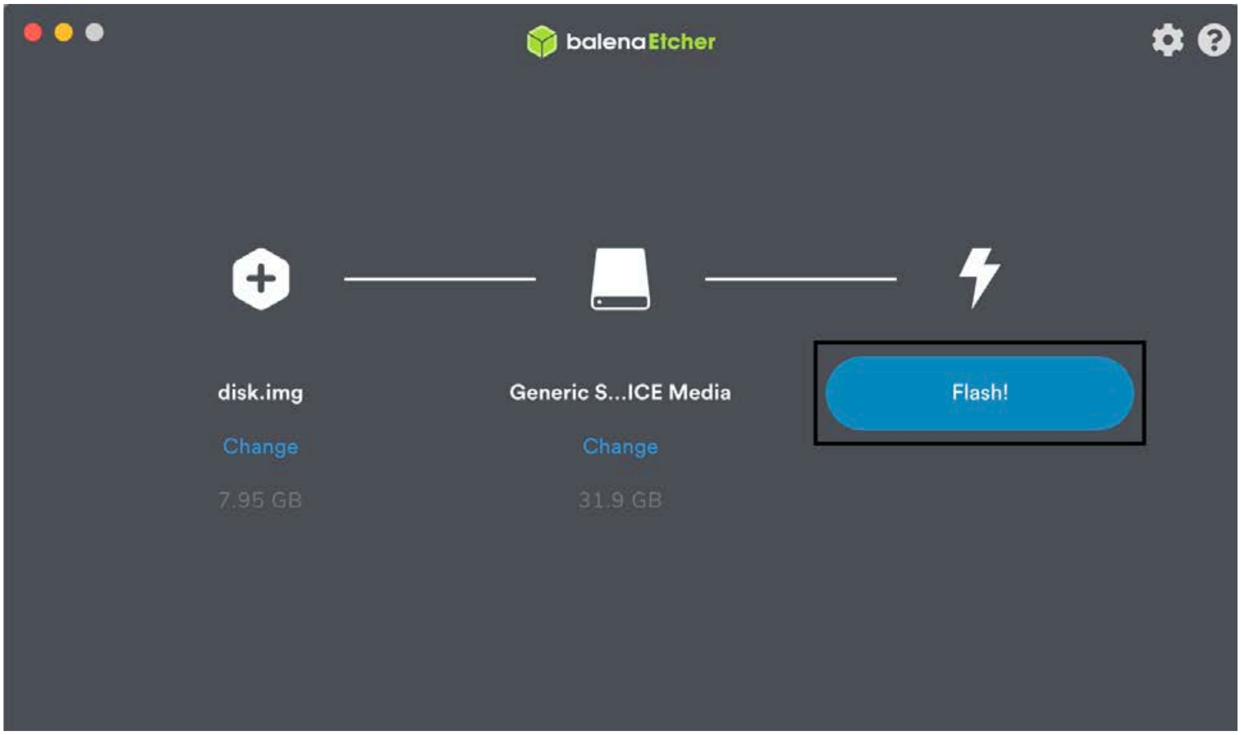
4. Click **Select target,** and select the SD card you connected to the computer (the software may automatically recognize the target):



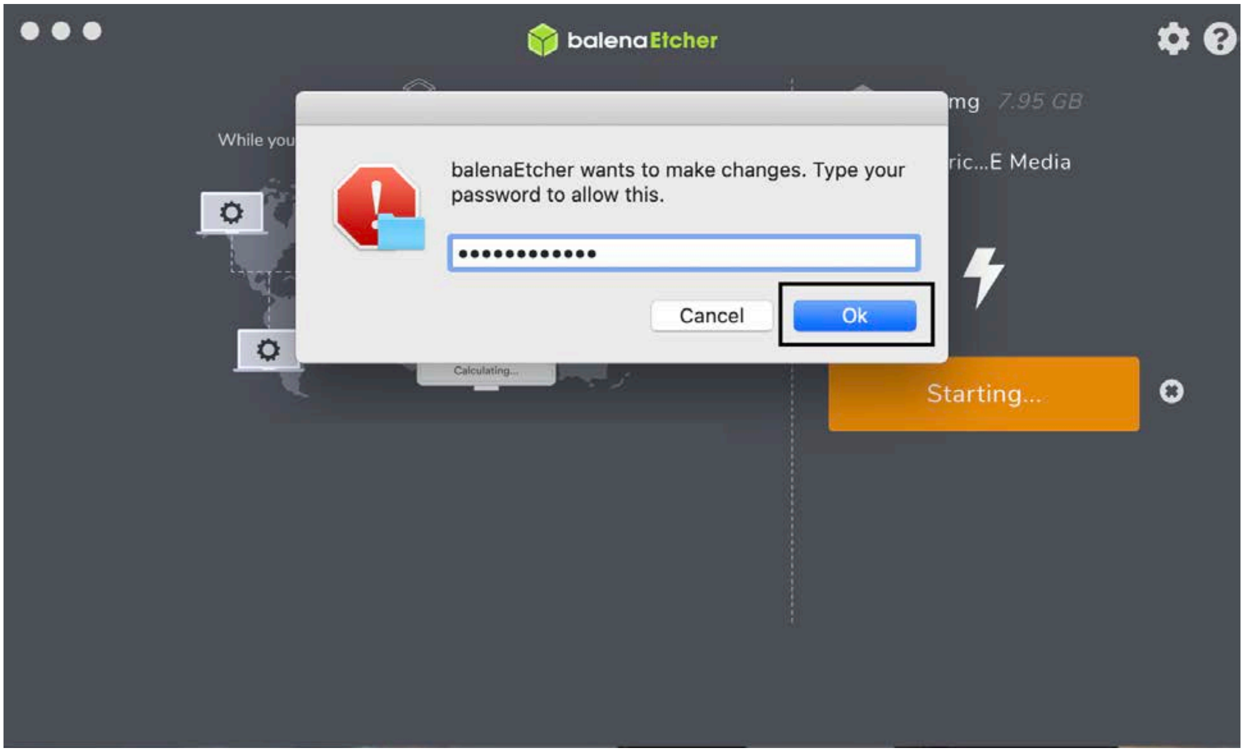
You should see something similar to the following:



2055 5. Click **Flash!** to start the flashing process:



2056 You may be prompted to enter your password, as seen below:



2057 When the flashing has completed, you should see output similar to the following:

4.1.5.3.2 Install Micronets Proto-Pi

1. Insert the SD card to the Raspberry Pi, and connect power using a micro–Universal Serial Bus (USB) cable.

2. Connect to the Raspberry Pi from a remote machine by using SSH:

   Note: You will need to figure out the Ethernet IP address of the Raspberry Pi, which can be done by looking at the DHCP assignments on the gateway to which you connected the Raspberry Pi.

   a. Enter the following command once you have identified the device's IP address:

   ```
   ssh pi@[ipaddress]
   ```

   b. You will be prompted to continue connecting as this is the first time connecting to the device:

```
[Bla          :~ bla          ta$ ssh pi@192.168.30.191
The authenticity of host '192.168.30.191 (192.168.30.191)' can't be established.
ECDSA key fingerprint is SHA256:
Are you sure you want to continue connecting (yes/no)? yes
```

c. Enter the password for the Raspberry Pi:

Note: The password is “micronets” if you are leveraging the CableLabs Raspberry Pi image:

```
[Bla          :~ bl          a$ ssh pi@192.168.30.191
The authenticity of host '192.168.30.191 (192.168.30.191)' can't be established.
ECDSA key fingerprint is SHA256:
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.30.191' (ECDSA) to the list of known hosts.
pi@192.168.30.191's password:
```

d. You will now have access to a terminal on the Raspberry Pi:

```
[Bla          -2:~ bla          a$ ssh pi@192.168.30.191
The authenticity of host '192.168.30.191 (192.168.30.191)' can't be established.
ECDSA key fingerprint is SHA256:
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.30.191' (ECDSA) to the list of known hosts.
[pi@192.168.30.191's password:
Linux raspberrypi 4.19.75-v7+ #1270 SMP Tue Sep 24 18:45:11 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Dec 23 20:06:19 2019
pi@raspberrypi:~ $
```

3. Ensure that you are in the home directory by entering the following command:

```
cd ~
```

4. Download the Micronets Proto-Pi software from GitHub by entering the following command:

```
git clone https://git@github.com/cablelabs/micronets-pi3.git
```

You should see output similar to the following:

```
pi@raspberrypi:~ $ git clone https://git@github.com/cablelabs/micronets-pi3.git
Cloning into 'micronets-pi3'...
remote: Enumerating objects: 459, done.
remote: Counting objects: 100% (459/459), done.
remote: Compressing objects: 100% (328/328), done.
remote: Total 459 (delta 247), reused 338 (delta 126), pack-reused 0
Receiving objects: 100% (459/459), 12.74 MiB | 8.51 MiB/s, done.
Resolving deltas: 100% (247/247), done.
```

5. Change into the micronets-pi3 directory by entering the following command:

```
cd micronets-pi3/
```

6. Check out the nccoe-build-3 branch by entering the following branch:

```
git checkout nccoe-build-3
```

You should see output similar to the following:

```
pi@raspberrypi:~/micronets-pi3 $ git checkout nccoe-build-3
Branch 'nccoe-build-3' set up to track remote branch 'nccoe-build-3' from 'origin'.
Switched to a new branch 'nccoe-build-3'
```

7. Change into the deploy directory by entering the following command:

```
cd deploy/
```

8. Install the Micronets Proto-Pi software by entering the following command:

```
./install
```

When prompted to accept disk space required, input **Y** as seen below:

```
Get:4 http://raspbian.raspberrypi.org/raspbian buster/main armhf Packages [13.0 MB]
Fetched 13.4 MB in 13s (1,015 kB/s)
Reading package lists... Done
*** Configuring sudoer privileges required by micronets application (user: pi) ***
*** Adding user pi to groups: netdev, gpio ***
*** Creating desktop autostart file ***
*** Install python (pip3) dependencies ***
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting pyscreenshot
  Downloading https://files.pythonhosted.org/packages/ef/f2/35066da41daceabb3d6f1d44d98457
f2b3ddca786181fc7cc9c45e8ef491/pyscreenshot-1.0-py2.py3-none-any.whl
Collecting entrypoint2 (from pyscreenshot)
  Downloading https://files.pythonhosted.org/packages/ca/7e/2c5f211ebbb37c7bd474f3b2d813bd
e5b5391f31c46e190b2b84d83ec9b7/entrypoint2-0.2-py2.py3-none-any.whl
Collecting EasyProcess (from pyscreenshot)
  Downloading https://files.pythonhosted.org/packages/32/8f/88d636f1da22a3c573259e44cfefb4
6a117d3f9432e2c98b1ab4a21372ad/EasyProcess-0.2.10-py2.py3-none-any.whl
Collecting decorator (from entrypoint2->pyscreenshot)
  Downloading https://files.pythonhosted.org/packages/ed/1b/72a1821152d07cf1d8b6fce298aeb0
6a7eb90f4d6d41acec9861e7cc6df0/decorator-4.4.2-py2.py3-none-any.whl
Collecting argparse (from entrypoint2->pyscreenshot)
  Downloading https://files.pythonhosted.org/packages/f2/94/3af39d34be01a24a6e65433d19e107
099374224905f1e0cc6bbe1fd22a2f/argparse-1.4.0-py2.py3-none-any.whl
Installing collected packages: decorator, argparse, entrypoint2, EasyProcess, pyscreenshot
Successfully installed EasyProcess-0.2.10 argparse-1.4.0 decorator-4.4.2 entrypoint2-0.2 p
yscreenshot-1.0
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting qrcode
  Downloading https://files.pythonhosted.org/packages/42/87/4a3a77e59ab7493d64da1f69bf1c2e
899a4cf81e51b2baa855e8cc8115be/qrcode-6.1-py2.py3-none-any.whl
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from qrcode) (1.12.0
)
Installing collected packages: qrcode
  The script qr is installed in '/home/pi/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use -
-no-warn-script-location.
Successfully installed qrcode-6.1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  point-rpi
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  python3-pil
Suggested packages:
  python-pil-doc python3-pil-dbg python3-pil.imagetk-dbg
The following NEW packages will be installed:
  python3-pil.imagetk
The following packages will be upgraded:
  python3-pil
1 upgraded, 1 newly installed, 0 to remove and 154 not upgraded.
Need to get 429 kB of archives.
After this operation, 93.2 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

2091
```
yscreenshot-1.0
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting qrcode
  Downloading https://files.pythonhosted.org/packages/42/87/4a3a77e59ab7493d64da1f69bf1c2e
899a4cf81e51b2baa855e8cc8115be/qrcode-6.1-py2.py3-none-any.whl
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from qrcode) (1.12.0
)
Installing collected packages: qrcode
  The script qr is installed in '/home/pi/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use -
-no-warn-script-location.
Successfully installed qrcode-6.1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  point-rpi
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  python3-pil
Suggested packages:
  python-pil-doc python3-pil-dbg python3-pil.imagetk-dbg
The following NEW packages will be installed:
  python3-pil.imagetk
The following packages will be upgraded:
  python3-pil
1 upgraded, 1 newly installed, 0 to remove and 154 not upgraded.
Need to get 429 kB of archives.
After this operation, 93.2 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://mirror.umd.edu/raspbian/raspbian buster/main armhf python3-pil.imagetk armhf
5.4.1-2+deb10u1 [65.0 kB]
Get:2 http://mirror.umd.edu/raspbian/raspbian buster/main armhf python3-pil armhf 5.4.1-2+
deb10u1 [364 kB]
Fetched 429 kB in 1s (471 kB/s)
Reading changelogs... Done
Selecting previously unselected package python3-pil.imagetk:armhf.
(Reading database ... 95711 files and directories currently installed.)
Preparing to unpack .../python3-pil.imagetk_5.4.1-2+deb10u1_armhf.deb ...
Unpacking python3-pil.imagetk:armhf (5.4.1-2+deb10u1) ...
Preparing to unpack .../python3-pil_5.4.1-2+deb10u1_armhf.deb ...
Unpacking python3-pil:armhf (5.4.1-2+deb10u1) over (5.4.1-2) ...
Setting up python3-pil:armhf (5.4.1-2+deb10u1) ...
Setting up python3-pil.imagetk:armhf (5.4.1-2+deb10u1) ...
*** Configuring splash screen service ***
Created symlink /etc/systemd/system/sysinit.target.wants/splashscreen.service → /etc/syste
md/system/splashscreen.service.
*** Configuring goodbye screen service ***
Created symlink /etc/systemd/system/multi-user.target.wants/goodbyescreen.service → /usr/l
ib/systemd/system-shutdown/goodbyescreen.service.
Created symlink /etc/systemd/system/goodbyescreen.service → /usr/lib/systemd/system-shutdo
wn/goodbyescreen.service.
*** Configure onboard wifi ***
Onboard wifi should be disabled if you are using an external USB wifi adapter.
Disable onboard wifi adapter? [y/N] Y
```

2092

```
[PITFT] Making sure console doesn't use PiTFT
Removing console fbcon map from /boot/cmdline.txt
Screen blanking time reset to 10 minutes
[PITFT] Adding FBCP support...
Installing cmake...
W: --force-yes is deprecated, use one of the options starting with --allow instead.
Downloading rpi-fbcp...
Uncompressing rpi-fbcp...
Building rpi-fbcp...
Installing rpi-fbcp...
Remove fbcp from /etc/rc.local, if it's there...
We have systemd, so install fbcp systemd unit...
Created symlink /etc/systemd/system/multi-user.target.wants/fbcp.service → /etc/systemd/sy
stem/fbcp.service.
Setting raspi-config to boot to desktop w/o login...
Configuring boot/config.txt for forced HDMI
Using x2 resolution
[PITFT] Updating X11 default calibration...
[PITFT] Success!

Settings take effect on next boot.

REBOOT NOW? [y/N] Exiting without reboot.
~/micronets-pi3/deploy
*** Build/Install wpa_supplicant ***
Stopping wpa_supplicant service
Selected interface 'wlan1'
OK
Removed /etc/systemd/system/dbus-fi.w1.wpa_supplicant1.service.
Removed /etc/systemd/system/multi-user.target.wants/wpa_supplicant.service.
*** Installing pre-requisites ***
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.6).
gcc is already the newest version (4:8.3.0-1+rpi2).
gcc set to manually installed.
make is already the newest version (4.2.1-1.2).
make set to manually installed.
pkg-config is already the newest version (0.29-6).
The following package was automatically installed and is no longer required:
  point-rpi
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libssl1.1
Suggested packages:
  libssl-doc
The following NEW packages will be installed:
  libnl-3-dev libnl-genl-3-dev libssl-dev
The following packages will be upgraded:
  libssl1.1
1 upgraded, 3 newly installed, 0 to remove and 151 not upgraded.
Need to get 2,970 kB of archives.
After this operation, 6,558 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

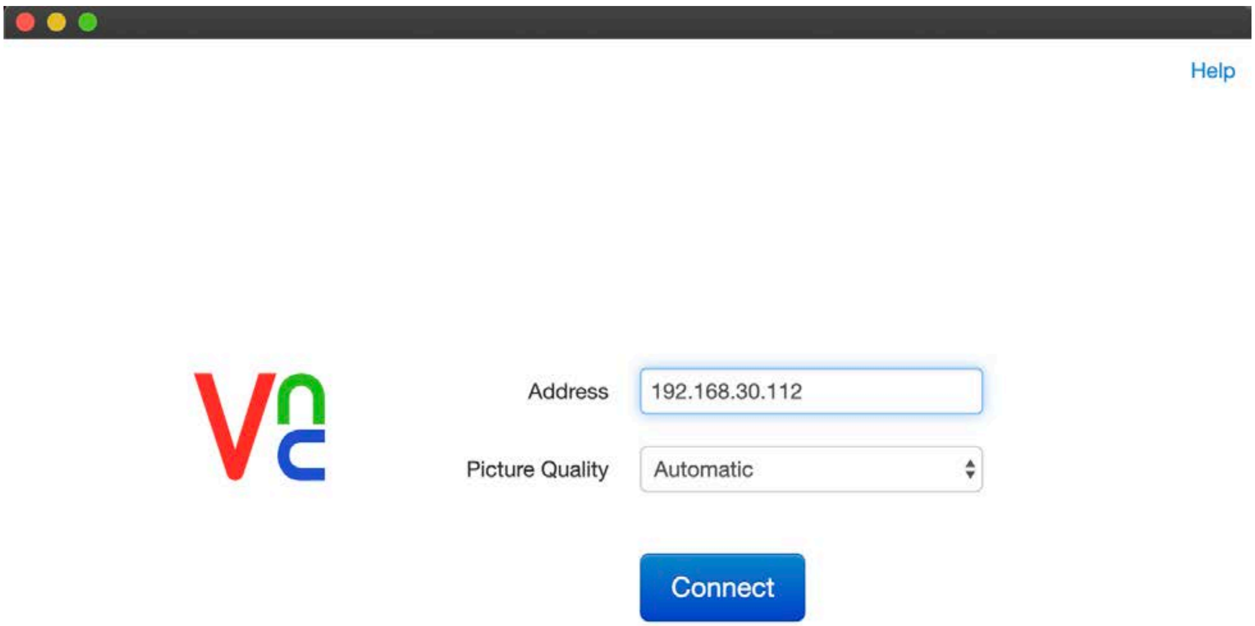
```
  CC  ../src/crypto/random.c
  CC  ../src/common/ctrl_iface_common.c
  CC  ctrl_iface.c
  CC  ctrl_iface_unix.c
  CC  ../src/utils/base64.c
  CC  sme.c
  CC  ../src/common/ieee802_11_common.c
  CC  ../src/common/hw_features_common.c
  CC  ../src/eap_common/eap_common.c
  CC  ../src/crypto/sha1-prf.c
  CC  ../src/crypto/sha1-tlsprf.c
  CC  ../src/common/gas_server.c
  CC  ../src/common/gas.c
  CC  gas_query.c
  CC  offchannel.c
  CC  ../src/utils/json.c
  CC  ../src/drivers/driver_common.c
  CC  wpa_supplicant.c
  CC  events.c
  CC  blacklist.c
  CC  wpas_glue.c
  CC  scan.c
  CC  main.c
  CC  ../src/drivers/driver_wext.c
  CC  ../src/drivers/driver_wired.c
  CC  ../src/drivers/driver_wired_common.c
  CC  ../src/drivers/driver_nl80211.c
  CC  ../src/drivers/driver_nl80211_capa.c
  CC  ../src/drivers/driver_nl80211_event.c
  CC  ../src/drivers/driver_nl80211_monitor.c
  CC  ../src/drivers/driver_nl80211_scan.c
  CC  ../src/drivers/netlink.c
  CC  ../src/drivers/linux_ioctl.c
  CC  ../src/drivers/rfkill.c
  CC  ../src/utils/radiotap.c
  CC  ../src/drivers/drivers.c
  CC  ../src/l2_packet/l2_packet_linux.c
  LD  wpa_supplicant
  CC  wpa_cli.c
  CC  ../src/common/wpa_ctrl.c
  CC  ../src/common/cli.c
  CC  ../src/utils/edit_simple.c
  LD  wpa_cli
  CC  wpa_passphrase.c
  LD  wpa_passphrase
  sed systemd/wpa_supplicant.service.in
  sed systemd/wpa_supplicant.service.arg.in
  sed systemd/wpa_supplicant-nl80211.service.arg.in
  sed systemd/wpa_supplicant-wired.service.arg.in
  sed dbus/fi.w1.wpa_supplicant1.service.in
*** Installing wpa_supplicant and wpa_cli ***
*** Initializing /etc/wpa_supplicant/wpa_supplicant.conf ***
Buster+
Touchscreen already configured
Reboot Now? [y/N] Y
```

4.1.5.3.3 Operation

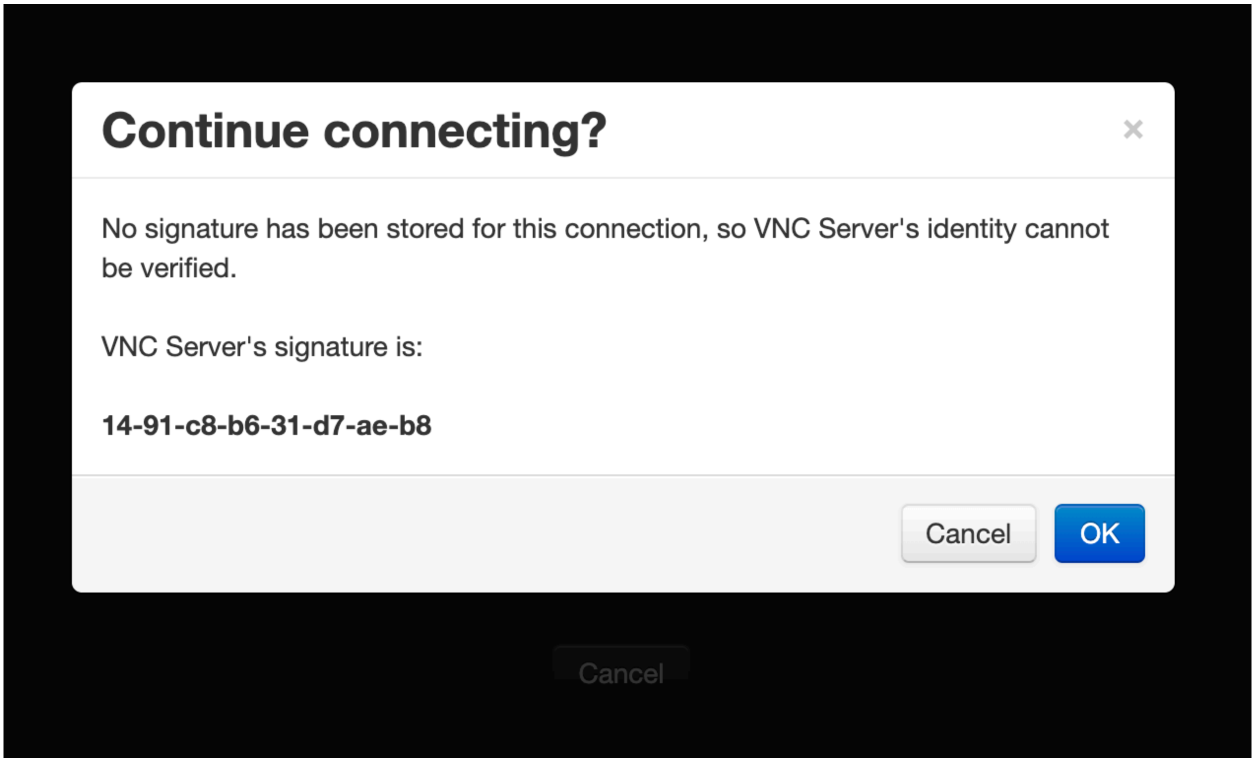
Four buttons are used for general operation in the Micronets Proto-Pi application. These buttons are on the right side of the application and will be described in the upcoming sections.

1. Accessing Raspberry Pi Using Virtual Network Computing (VNC)Viewer:

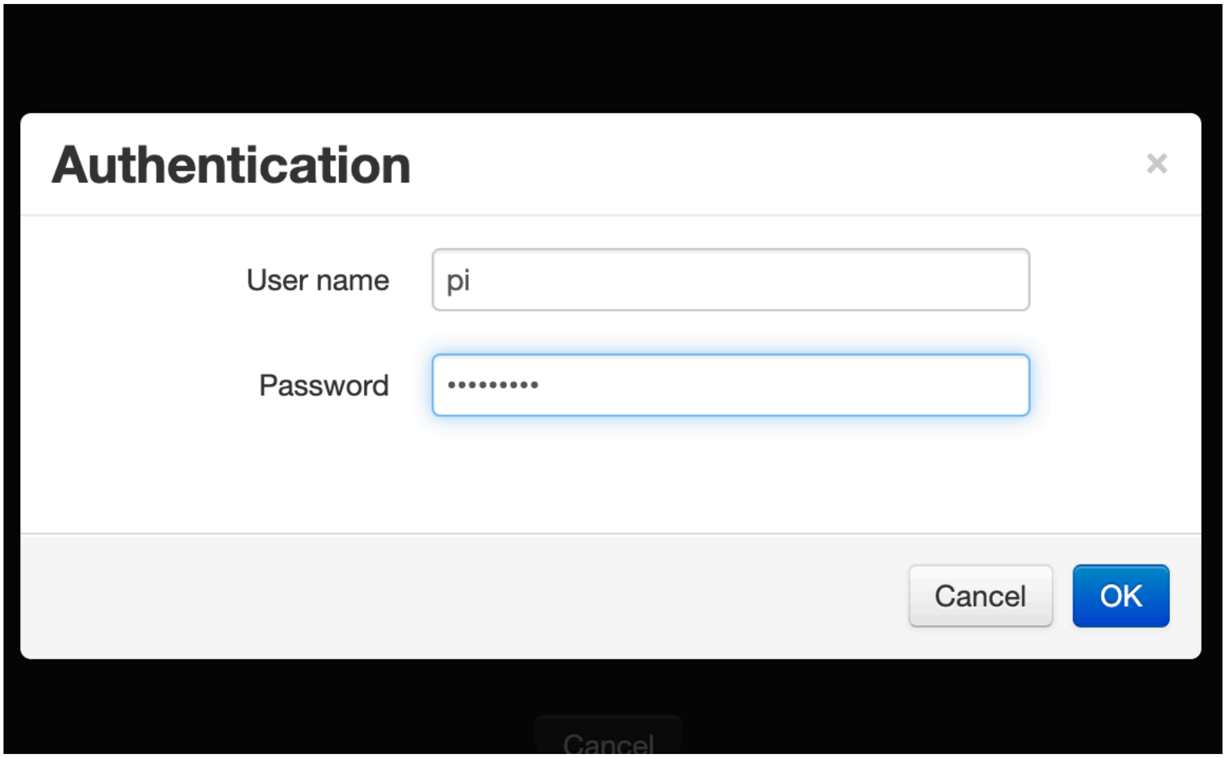
    a. Access the Raspberry Pi using the VNC Viewer, enter the IP address of the Raspberry Pi, and click **Connect:**



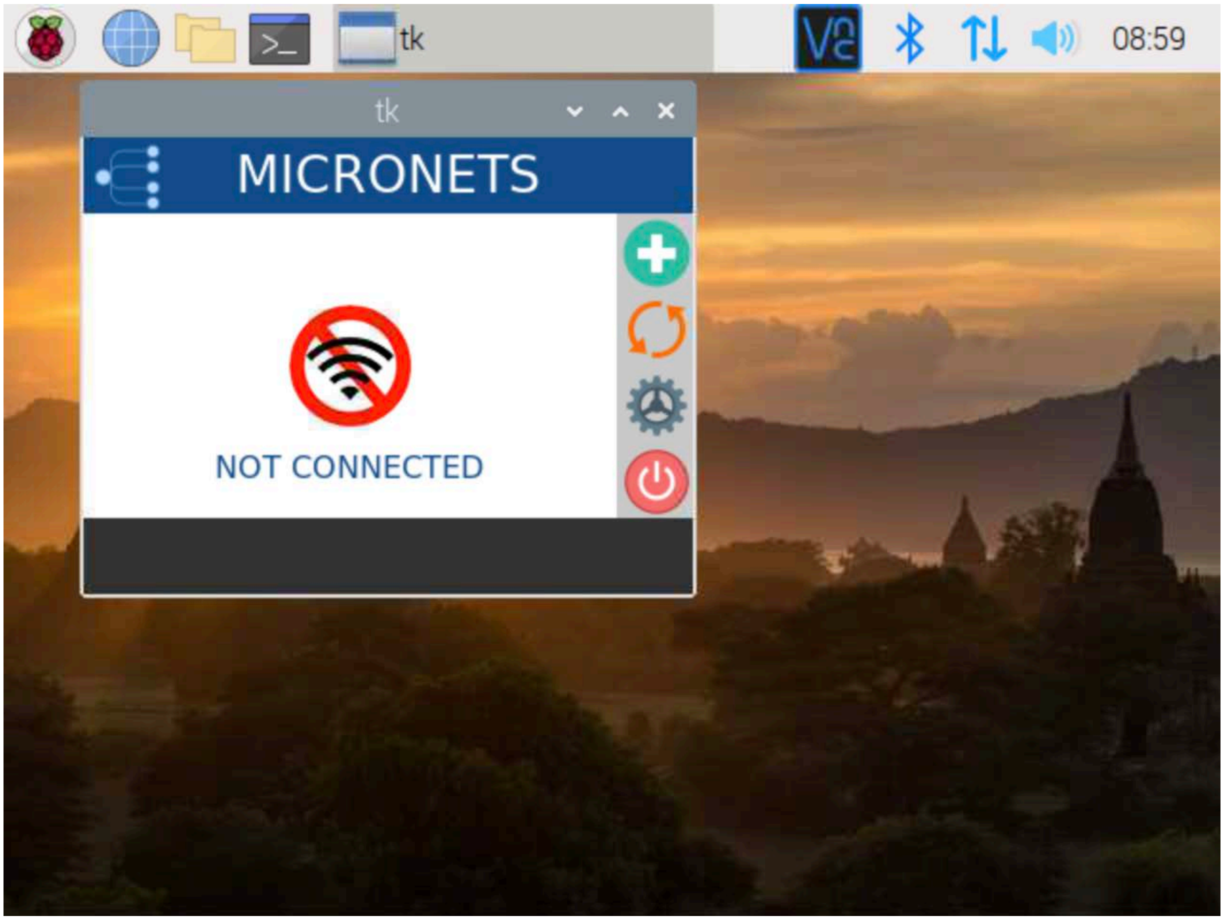
    You will be prompted to accept and store the signature for this device as it is the first time connecting to it. Click **OK:**



2102 Once accepted, proceed to log in with the username and password, as seen below:



b. You should see the Micronets Proto-Pi application on the screen as seen below:



2104 2. The onboard button described in the following steps allows the user to initiate the onboard op-
2105 eration:

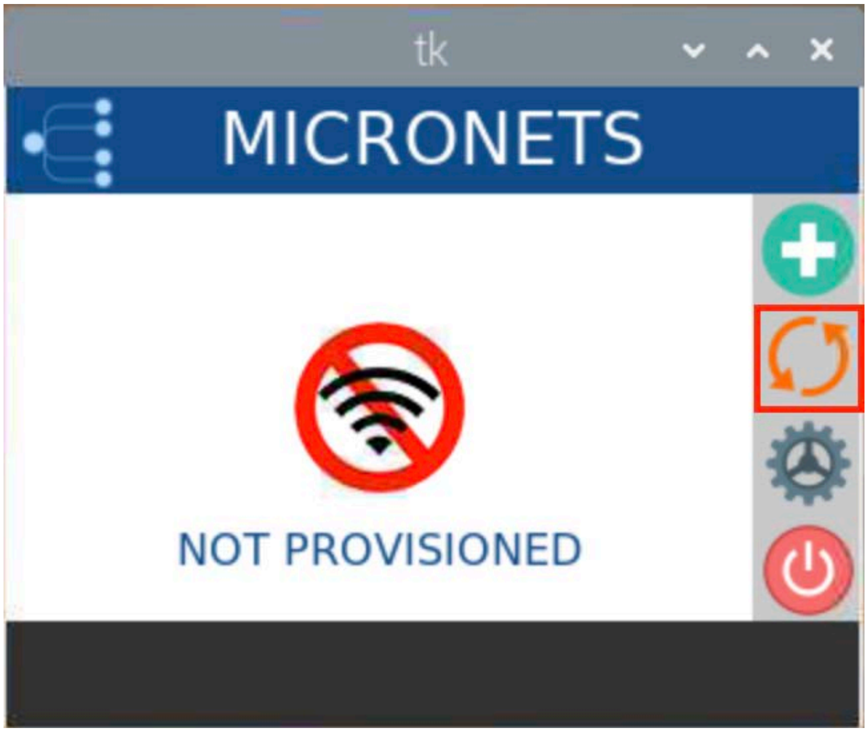
2106 a. Click the green button to initiate the onboard process:

2107 

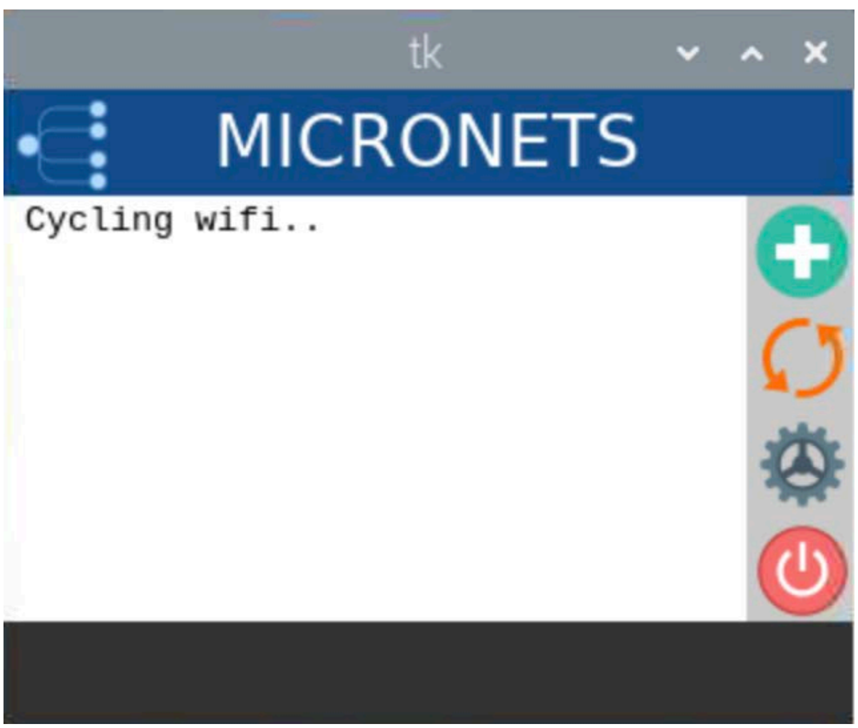
2108 A QR code will appear as seen below. The mobile application will be used to scan this QR
2109 code for onboarding:

3. The cycle button described in the following steps turns the Wi-Fi off/on to reconnect to the configured service set identifier (SSID).

    a. Click the orange cycle button:

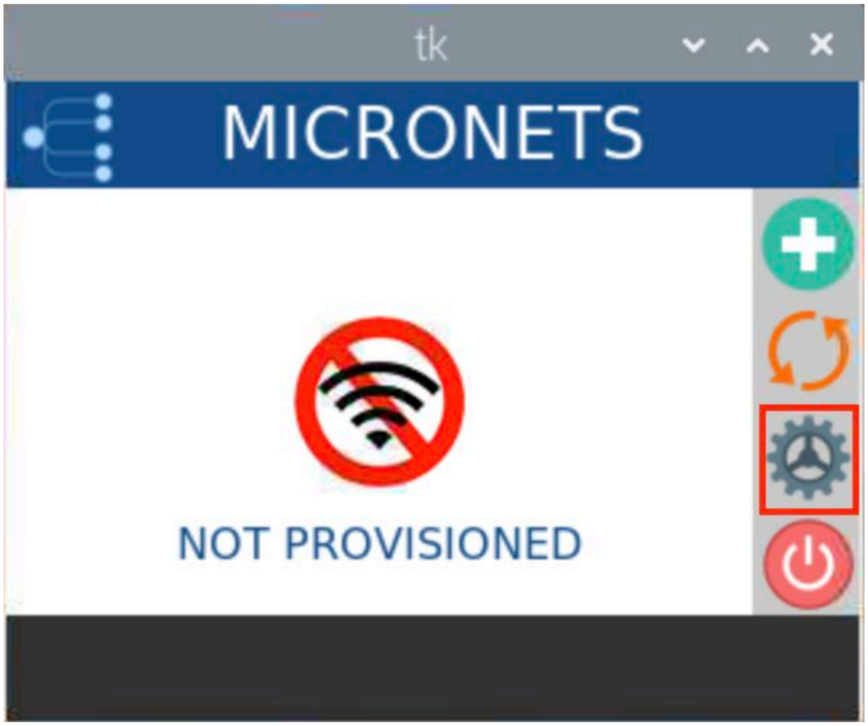


You should see output similar to the following:

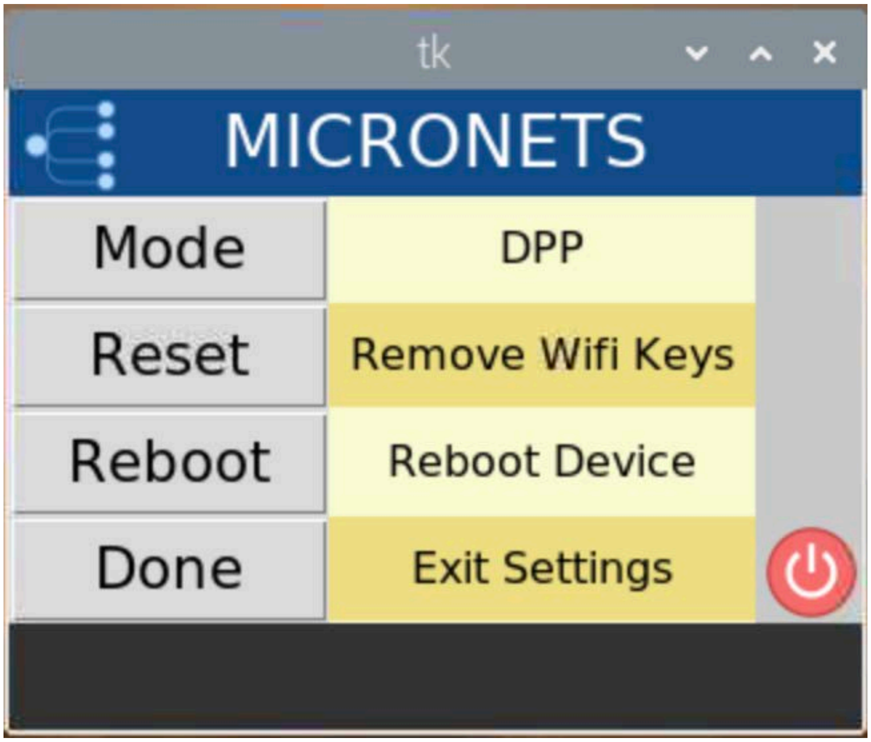


4. The settings button described in the following steps will open the settings menu, which has four different operations/buttons:

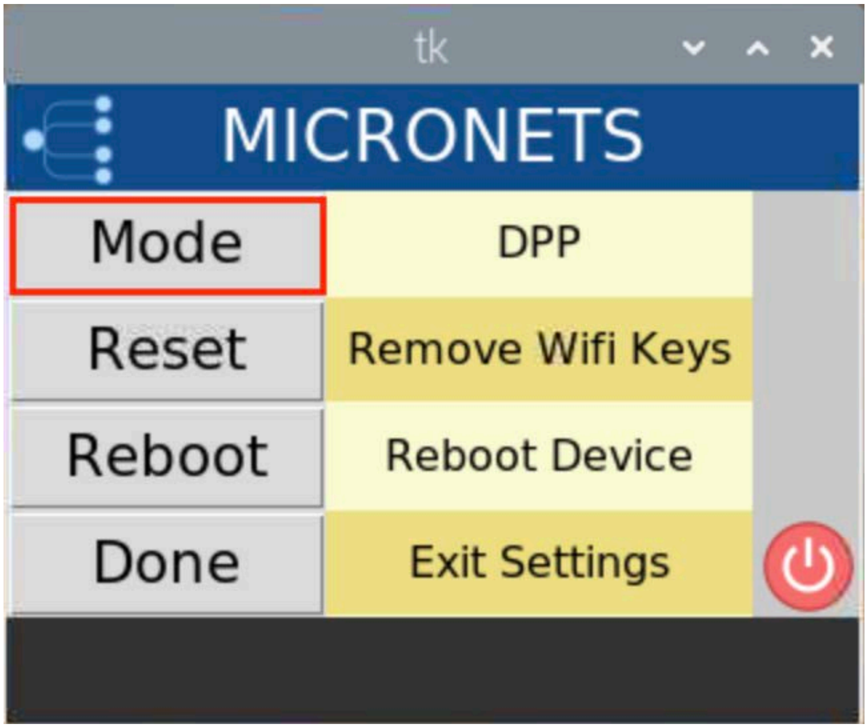
    a. Click the gear button:



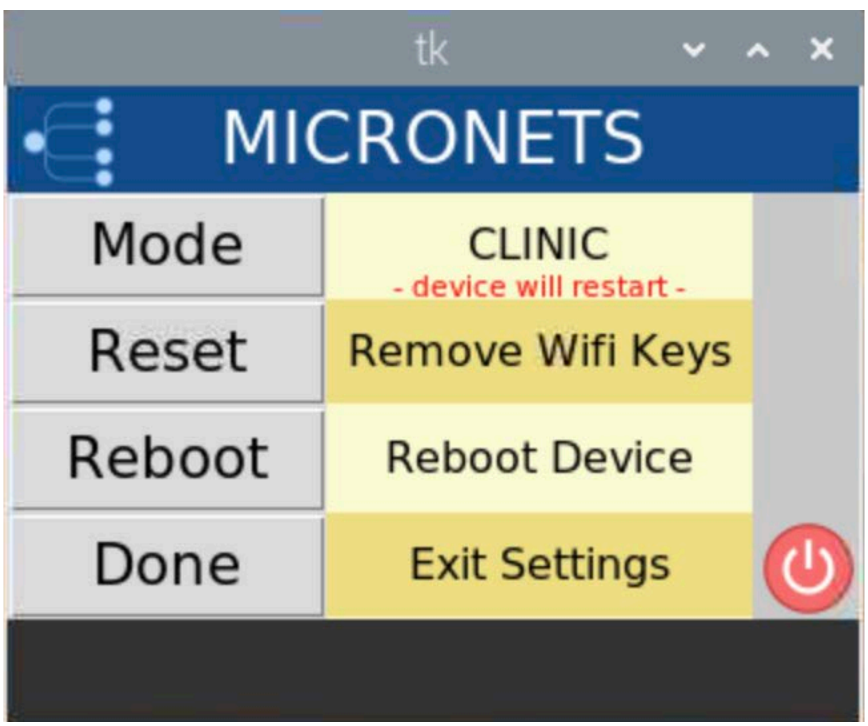
The following menu will appear:



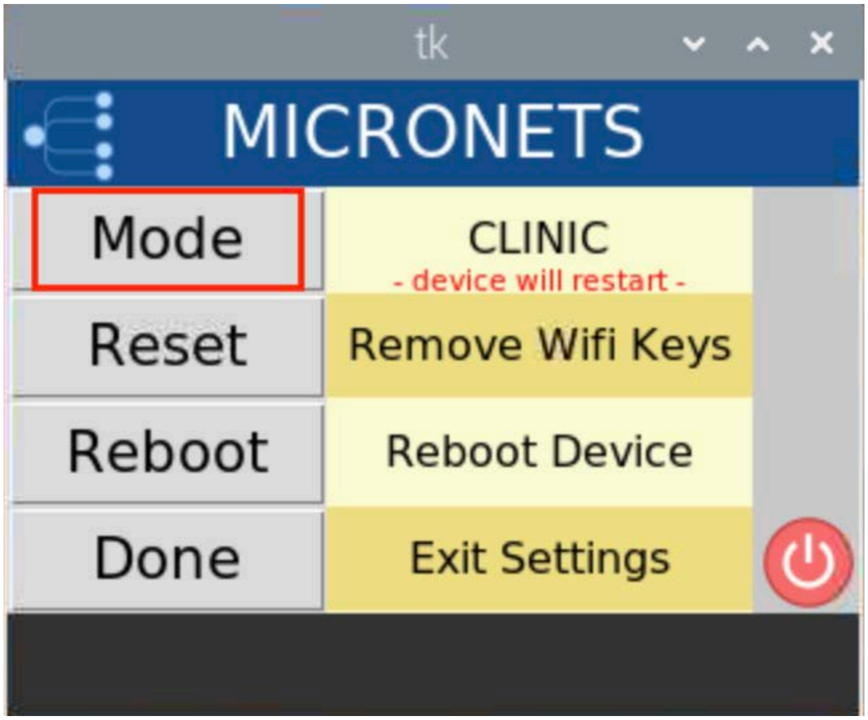
b. Click the **Mode** button to change the onboarding mode from DPP to clinic, and vice versa:



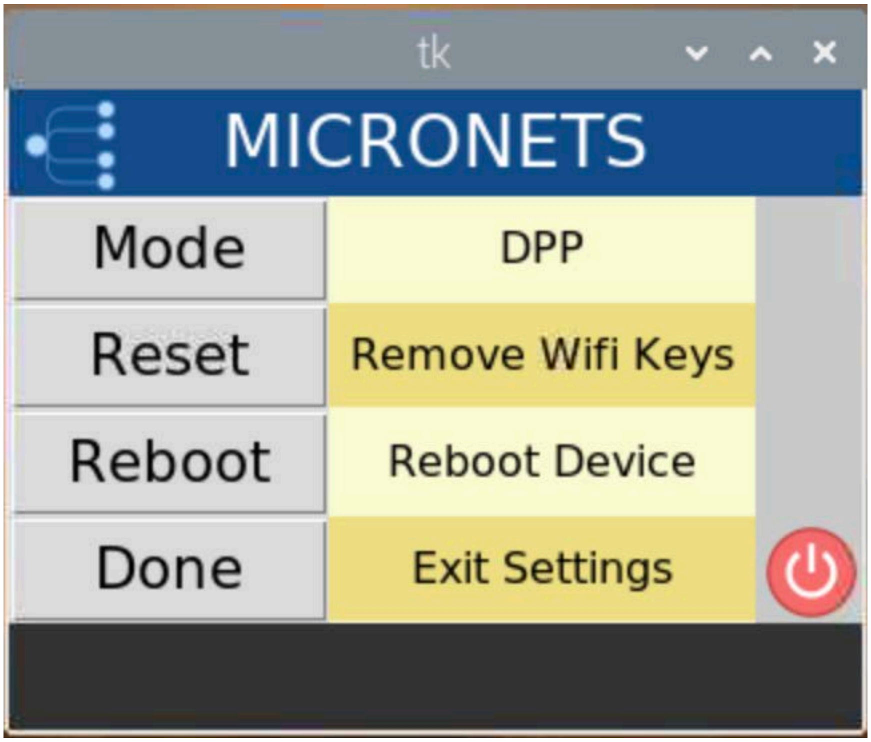
The following screen displays:



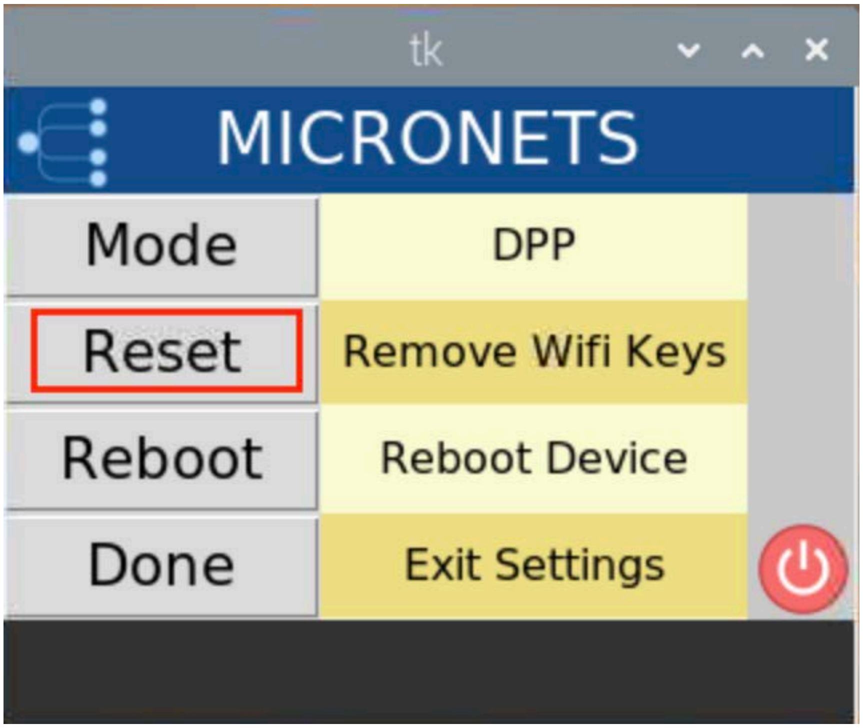
c. Click the **Mode** button again to return to DPP mode:



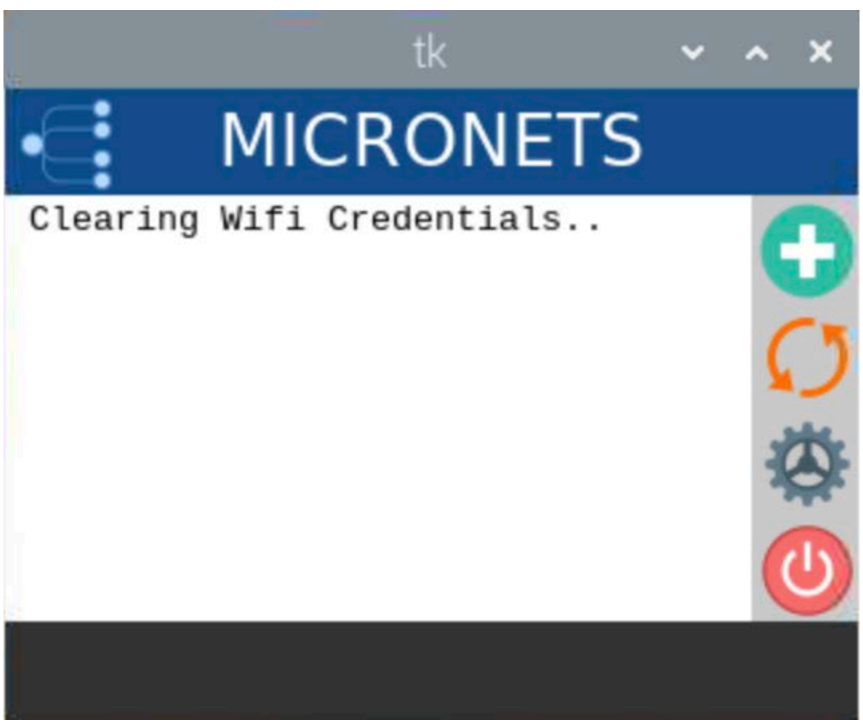
You will see the following change to your screen:



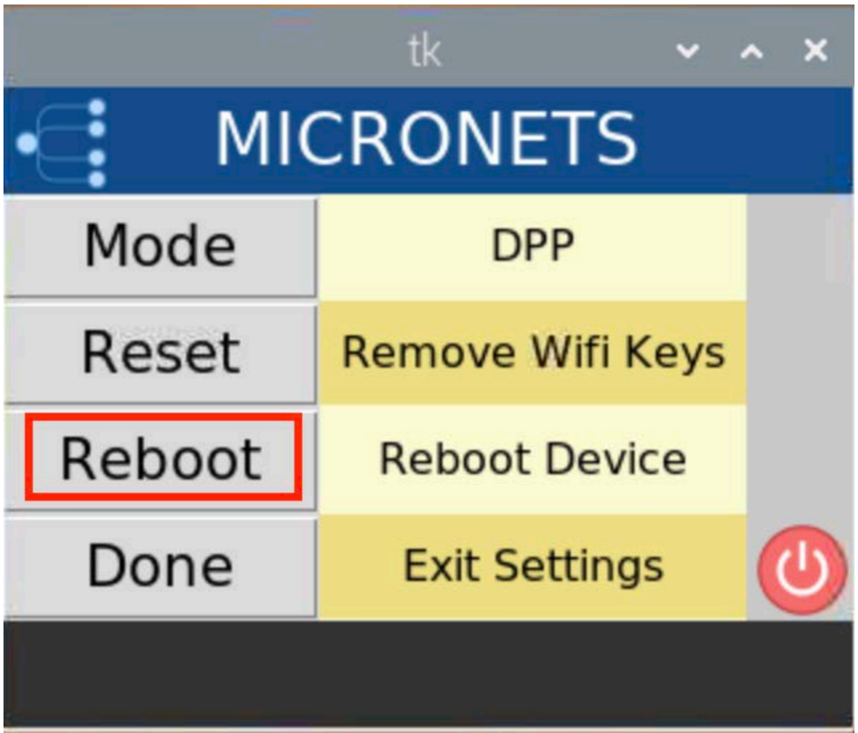
d. Click the **Reset** button to clear Wi-Fi credentials (Note: If the device is in clinic mode, it will restore the credentials for the clinic Wi-Fi):



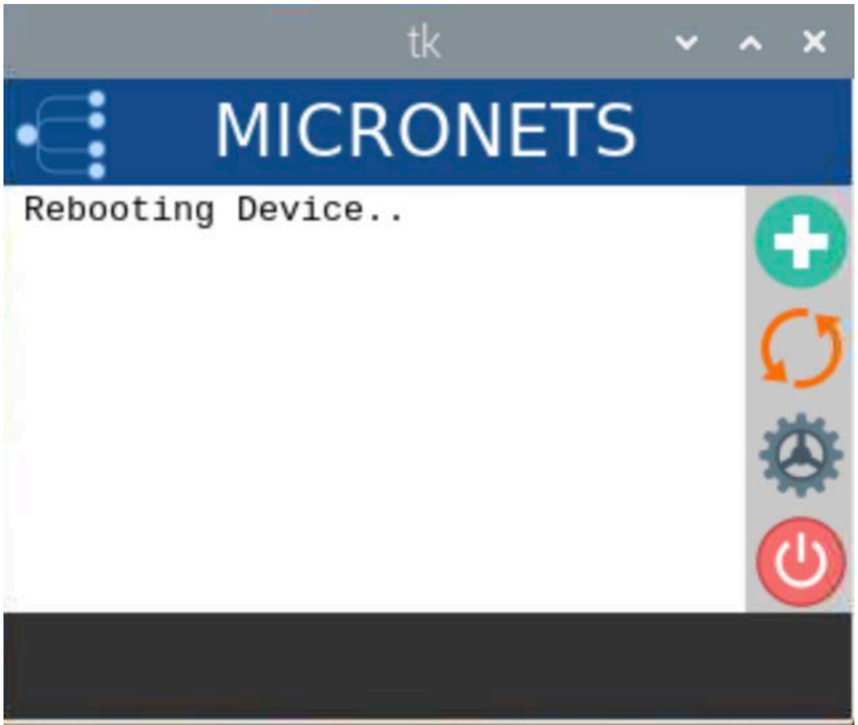
You should see output similar to the following:



e. Click the **Reboot** button to reboot the Pi:

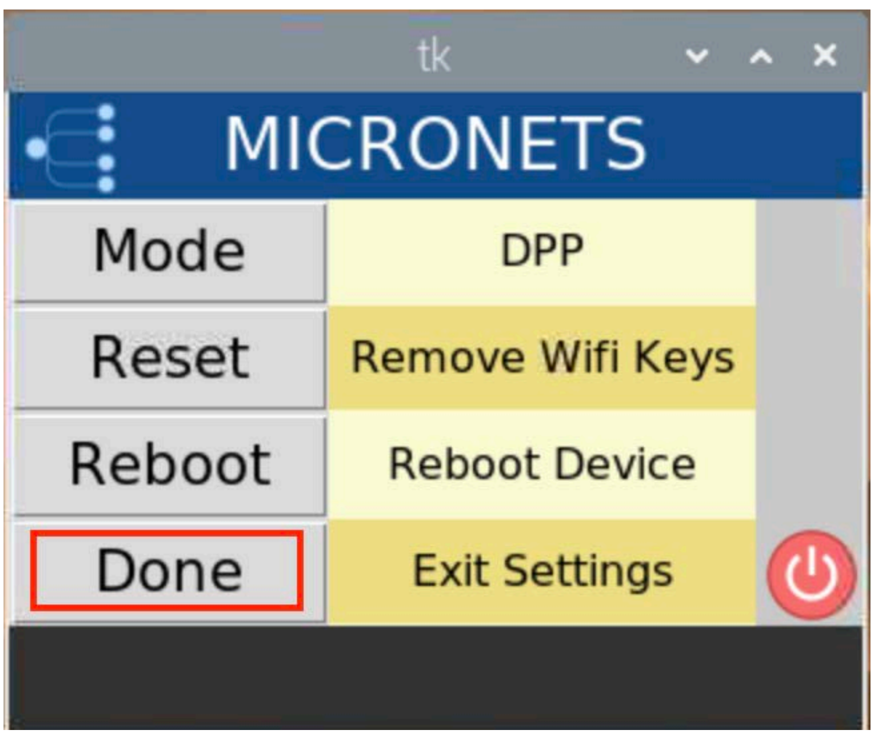


You should see output similar to the following:



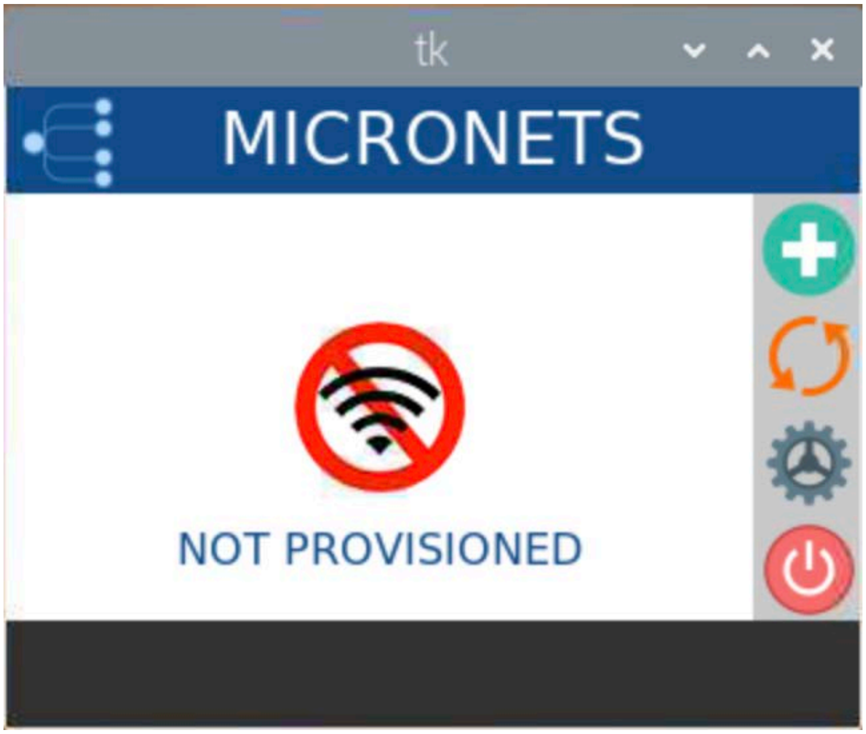
2140

2141 f. Click the **Done** button to exit the settings screen:



2142

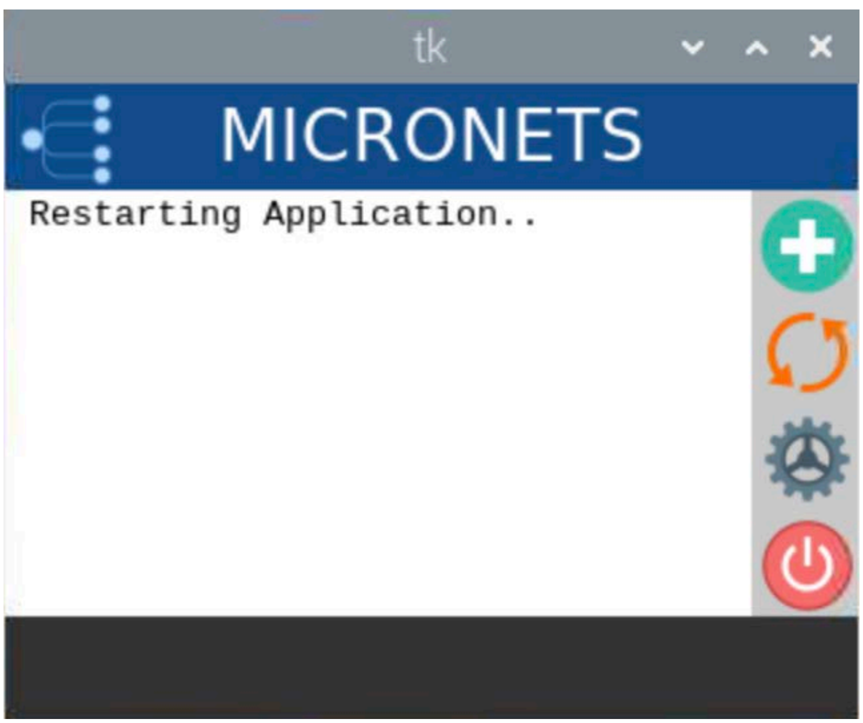
You should see output similar to the following:



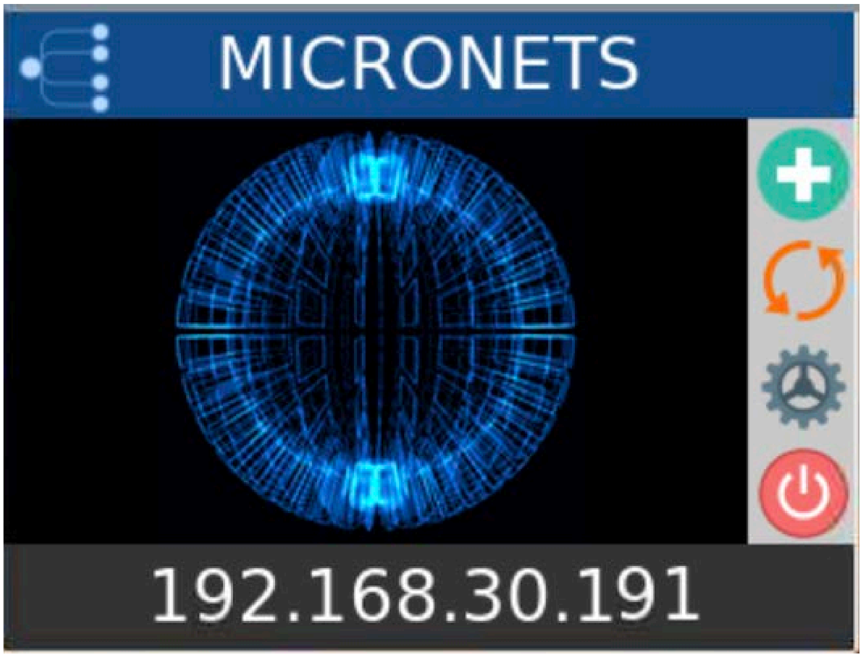
5. The power button described in the following steps appears on the main screen of the Micronets Proto-Pi application and is used to restart the application as well as shut down the Pi entirely:

    a. Tap the power button to restart the application:

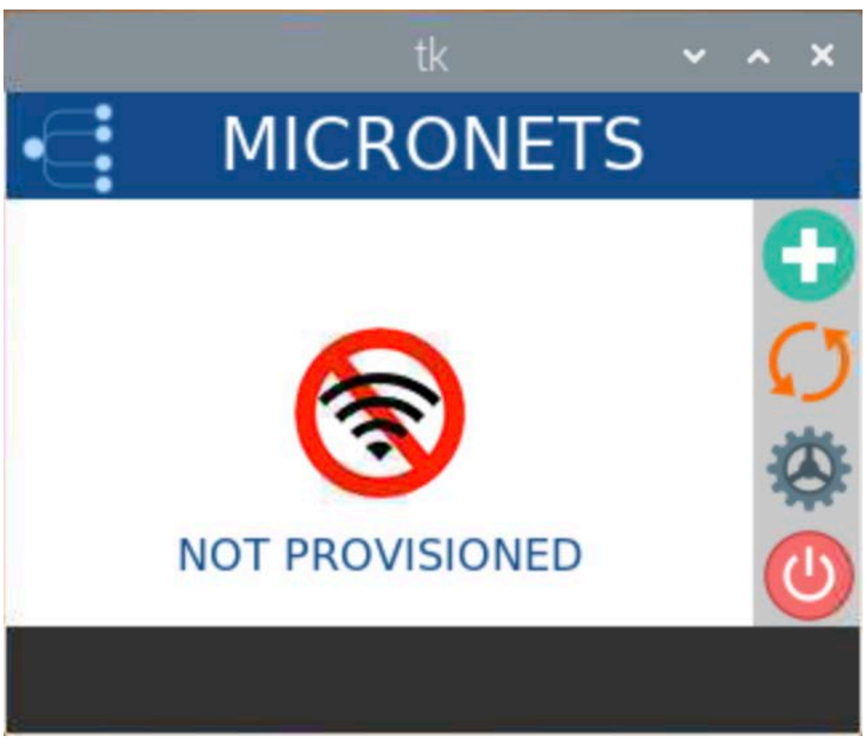
You should see output similar to the following:



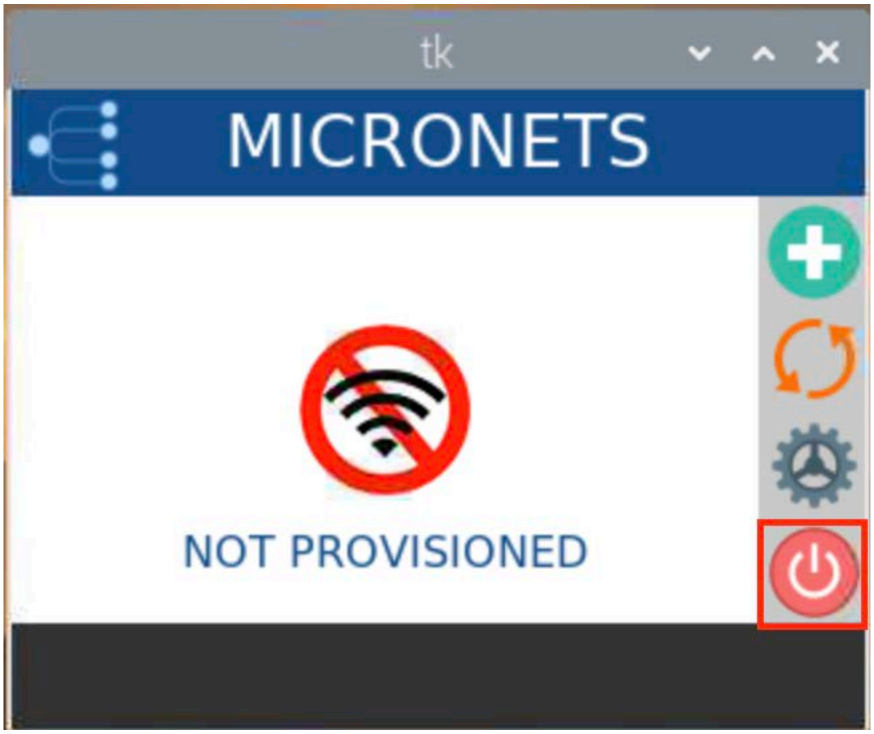
Next, the following screen should appear:



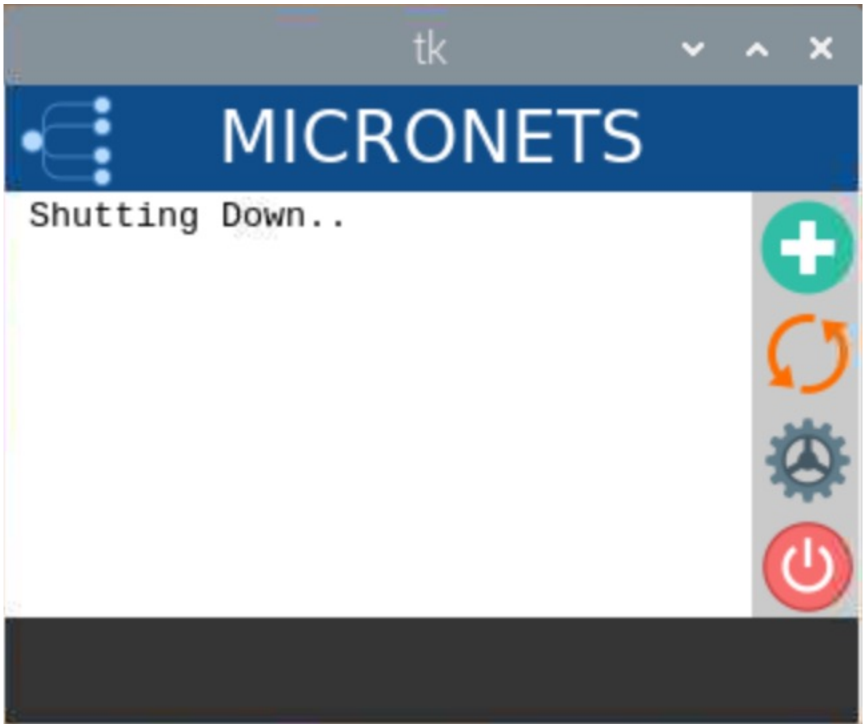
Finally, the main screen appears as seen below:



b. Hold the power button to shut down the Pi:



You should see output similar to the following:



4.1.6 Update Server

Build 3 leverages the preexisting update server that is described in Build 1's Update Server section. To implement a server that will act as an update server, see the documentation in Build 1's Update Server section. The update server will attempt to access and be accessed by the IoT device, which, in this case, is one of the development kits we built in the lab.

4.1.7 Unapproved Server

Build 3 leverages the preexisting unapproved server that is described in Build 1's Unapproved Server section. To implement a server that will act as an unapproved server, see the documentation in Build 1's Unapproved Server section. The unapproved server will attempt to access and be accessed by an IoT device, which, in this case, is one of the MUD-capable devices on the implementation network.

4.1.8 CableLabs MUD Registry

This section describes the CableLabs MUD registry, which, for this implementation, is a cloud-provided service. This implementation leveraged the nccoe-build-3 branch of CableLabs MUD registry Git release. This service can be hosted by the implementer or another party. This documentation describes setting up your own MUD registry.

4.1.8.1 CableLabs MUD Registry Overview

The Micronets MUD registry provides the capability to look up the MUD URL that is associated with a particular device. This registration and MUD URL association can be done manually or by the device using self-registration.

4.1.8.2 Configuration Overview

The following subsections document the software and network configurations for the MUD registry. Please note that the MUD manager, Micronets Manager, Websocket Proxy, MUD registry, and MSO portal are all implemented on the same server, nccoe-server1.micronets.net. Many of these configurations have already been covered in previous sections of this document but are repeated here for consistency.

4.1.8.2.1 Network Configuration

This server was hosted outside the lab environment on a Linode cloud-hosted Linux server. Its IP address was statically assigned.

4.1.8.2.2 Software Configuration

For this build, the server ran on an Ubuntu 18.04 LTS operating system. The MUD registry runs in its own docker container and is configured to use SSL/TLS encryption.

The following software is required to install, configure, and operate the MUD registry:

- an Ubuntu 18.04 LTS server reachable by the server hosting the Micronets Manager instances and any Micronets gateways
- docker (v18.06 or higher)
- curl
- NGINX

4.1.8.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the MUD registry:

- 4 GB of RAM
- 50 GB of free disk space

4.1.8.3 Setup

4.1.8.3.1 Install and Configure MUD Registry

1. Log in to docker by using the following command:

   ```
   docker login
   ```

   You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/micronets-dev/.docker/con
fig.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

2. Retrieve the nccoe-build-3 tagged image by entering the following command:

```
docker pull community.cablelabs.com:4567/micronets-docker/micronets-mud-regis-
try:nccoe-build-3
```

3. Execute the following command to run the image that was just retrieved:

   The command will follow the syntax below. Replace **<MUDFILESERVER_URL>** with your MUD file server URL:

   docker run -d -p 127.0.0.1:3082:3082 --env mud_base_uri=https://**<MUDFILESERVER_URL>** -v /etc/micronets/micronets-mud-registry.d/:/etc/micronets/config --name=micronets-mud-registry community.cablelabs.com:4567/micronets-docker/micronets-mud-registry:nccoe-build-3

```
docker run -d -p 127.0.0.1:3082:3082 --env mud_base_uri=https://nccoe-
server2.micronets.net/micronets-mud -v /etc/micronets/micronets-mud-regis-
try.d/:/etc/micronets/config --name=micronets-mud-registry community.cable-
labs.com:4567/micronets-docker/micronets-mud-registry:nccoe-build-3
```

4. Configure your own vendor code for your implementation by completing the following steps:

   a. Create and modify the ***mud-registry.conf*** file by executing the following command. (Note: The configuration file must be named “mud-registry.conf” and must reside in a host folder that is passed to the docker instance in the docker run command executed in the previous step.)

   ```
   sudo vim /etc/micronets/micronets-mud-registry.d/mud-registry.conf
   ```

   b. Replace **<VENDOR-CODE>** with your choice of vendor name, <**MUDREGISTRY_URL>** with the MUD registry URL, and **<MUDFILESERVER_URL>** with the MUD file server URL:

   {

   "vendors" : {

```
        "<VENDOR-CODE> ": "https:// <MUDREGISTRY_URL> /registry/devices",
        "ABCD": "https://abcd-domain.com:3082/vendors"
    },
    "mud_base_uri": "https:// <MUDFILESERVER_URL> /micronets-mud",
    "device_db_file": "/etc/micronets/config/device-registration.nedb"
}
```

For this implementation, we added the following:

```
{
    "vendors" : {
        "TEST": "https://nccoe-server1.micronets.net/registry/devices",
        "ABCD": "https://abcd-domain.com:3082/vendors"
    },
    "mud_base_uri": "https://nccoe-server2.micronets.net/micronets-mud",
    "device_db_file": "/etc/micronets/config/device-registration.nedb"
}
```

```
{
    "vendors" : {
        "TEST": "https://nccoe-server1.micronets.net/registry/devices",
        "ABCD": "https://abcd-domain.com:3082/vendors"
    },
    "mud_base_uri": "https://nccoe-server2.micronets.net/micronets-mud",
    "device_db_file": "/etc/micronets/config/device-registration.nedb"
}
```

c. Modify the sites-available file for the NGINX server to route appropriate traffic to the docker container by executing the following commands:

   i. Open the sites-available file for the NGINX server by entering the following command:

```
sudo vim /etc/nginx/sites-available/nccoe-server1.micronets.net
```

ii. Map the location for the /registry/devices so it is routed to vendors/ in the docker instance running on port 3082 and for the /mud/ to be passed to the global registry by adding the following to the server block:

```
location /registry/devices {
        proxy_pass    http://localhost:3082/vendors/;
}
location /mud/{
        proxy_pass    http://localhost:3082/registry/;
}
```

```
server {
        listen 443 ssl;
        listen [::]:443 ssl;
        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;
        server_name nccoe-server1.micronets.net;

        location / {
                try_files $uri $uri/ =404;
        }

        location /micronets/mud-manager/ {
        proxy_pass      http://localhost:8888/;
        }

        location /registry/devices {
                proxy_pass      http://localhost:3082/vendors/;
        }
        location /mud/{
                proxy_pass      http://localhost:3082/registry/;
        }

        ssl_certificate /home/micronets-dev/Projects/micronets/cert/nccoe-server1_micronets_n
et.crt;
        ssl_certificate_key /home/micronets-dev/Projects/micronets/cert/nccoe-server1_microne
ts_net.key;
}
~
~
```

4.1.9 CableLabs Micronets Manager for SDN Control

This section describes the CableLabs Micronets Manager, which, for this implementation, is a cloud-provided service. This implementation leveraged the nccoe-build-3 branch of CableLabs Micronets Manager Git release. This service can be hosted by the implementer or another party. This documentation describes setting up your own Micronets Manager.

4.1.9.1 CableLabs Micronets Manager Overview

The Micronets Manager provides micro-services to the implementation. It receives onboarding requests, bootstrapping information, and more for a particular subscriber and is a core component for handing off requests among different components in the architecture.

4.1.9.2 Configuration Overview

The following subsections document the software and network configurations for the Micronets Manager. Please note that these instructions have the MUD manager, Micronets Manager, Websocket Proxy, MUD registry, and MSO portal all deployed onto the same server, nccoe-server1.micronets.net. Many of these configurations are already covered in previous sections of this document but are repeated here for consistency.

4.1.9.2.1 Network Configuration

This server was hosted outside the lab environment on a Linode cloud-hosted Linux server. Its IP address was statically assigned.

4.1.9.2.2 Software Configuration

For this build, the server ran on an Ubuntu 18.04 LTS operating system. The Micronets Manager runs in its own docker container and is configured to use SSL/TLS encryption.

The following software is required to install, configure, and operate the Micronets Manager:

- an Ubuntu 18.04 LTS server reachable by any Micronets gateways
- docker (v18.06 or higher)
- docker-compose (v1.23.1 or higher)
- OpenSSL (1.0.2g or higher)
- curl
- NGINX (1.14.0 or higher)

4.1.9.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the Micronets Manager:

- 4 GB of RAM
- 50 GB of free disk space

4.1.9.3 Setup

4.1.9.3.1 Install Dependencies

1. Install docker, docker-compose, openssl, curl, and NGINX by entering the following command:

```
sudo apt-get install docker docker-compose openssl curl nginx
```

4.1.9.3.2 Install and Configure the Micronets Manager

1. Ensure the version of docker-compose is correct and upgrade if needed:
    a. Check the current version by entering the following command:

       ```
       docker-compose –version
       ```

       You should see the version output as seen below:

       ```
       micronets-dev@nccoe-server1:~/Projects/micronets$ docker-compose --version
       docker-compose version 1.24.1, build 4667896b
       ```

    b. If the version is earlier than v1.23.1, run the following command to install a new version in /usr/local/bin directory:
        i. Download the docker-compose utility:

           ```
           curl -s -L -O https://github.com/docker/compose/releases/download/1.24.1/docker-compose-Linux-`uname -m`
           ```

        ii. Install the docker-compose utility to the appropriate directory:

           ```
           sudo install -v -o root -m 755 docker-compose-Linux-`uname -m` /usr/local/bin/docker-compose
           ```

           You should see output similar to the following:

           ```
           micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 docker-compose-Linux-`uname -m` /usr/local/bin/docker-compose
           [sudo] password for micronets-dev:
           removed '/usr/local/bin/docker-compose'
           'docker-compose-Linux-x86_64' -> '/usr/local/bin/docker-compose'
           ```

2. Download the Micronets Manager management script, and install it by entering the following commands:
    a. Download the Micronets Manager management script:

       ```
       curl -s -O https://raw.githubusercontent.com/cablelabs/micronets-manager/nccoe-build-3/scripts/mm-container
       ```

    b. Download the docker-compose utility:

       ```
       curl -s -O https://raw.githubusercontent.com/cablelabs/micronets-manager/nccoe-build-3/scripts/docker-compose.yml
       ```

c. Install the management script to the appropriate location:

```
sudo install -v -o root -m 755 -D -t /etc/micronets/micronets-manager.d
mm-container
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 -D
-t /etc/micronets/micronets-manager.d mm-container
[sudo] password for micronets-dev:
removed '/etc/micronets/micronets-manager.d/mm-container'
'mm-container' -> '/etc/micronets/micronets-manager.d/mm-container'
```

d. Install the docker-compose utility to the appropriate location:

```
sudo install -v -o root -m 644 -D -t /etc/micronets/micronets-manager.d
docker-compose.yml
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 644 -D
-t /etc/micronets/micronets-manager.d docker-compose.yml
removed '/etc/micronets/micronets-manager.d/docker-compose.yml'
'docker-compose.yml' -> '/etc/micronets/micronets-manager.d/docker-compose.yml'
```

3. Copy the Micronets Manager server cert/key and the Websocket Proxy root CA cert created in earlier steps for use by the Micronets Manager docker container(s):

a. Install the certificates and keys by entering the following command:

```
sudo install -v -o root -m 600 -D -t /etc/micronets/micronets-man-
ager.d/lib micronets-manager.{cert,key}.pem micronets-ws-root.cert.pem
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 600 -D
-t /etc/micronets/micronets-manager.d/lib micronets-manager.{cert,key}.pem micronets
-ws-root.cert.pem
removed '/etc/micronets/micronets-manager.d/lib/micronets-manager.cert.pem'
'micronets-manager.cert.pem' -> '/etc/micronets/micronets-manager.d/lib/micronets-ma
nager.cert.pem'
removed '/etc/micronets/micronets-manager.d/lib/micronets-manager.key.pem'
'micronets-manager.key.pem' -> '/etc/micronets/micronets-manager.d/lib/micronets-man
ager.key.pem'
removed '/etc/micronets/micronets-manager.d/lib/micronets-ws-root.cert.pem'
'micronets-ws-root.cert.pem' -> '/etc/micronets/micronets-manager.d/lib/micronets-ws
-root.cert.pem'
```

b. Create a placeholder *micronets-ws-proxy.pkeycert.pem* file. This file is not used, but the Micronets Manager currently checks for it:

```
sudo touch /etc/micronets/micronets-manager.d/lib/micronets-ws-proxy.pkeycert.pem
```

4. Copy the shared secret value generated during the MSO portal installation:

```
sudo install -v -o root -g docker -m 660 -D -t /etc/micronets/micronets-manager.d/lib mso-auth-secret
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -g docker
-m 660 -D -t /etc/micronets/micronets-manager.d/lib mso-auth-secret
removed '/etc/micronets/micronets-manager.d/lib/mso-auth-secret'
'mso-auth-secret' -> '/etc/micronets/micronets-manager.d/lib/mso-auth-secret'
```

5. Execute the following command to download the Micronets Manager docker image (Note: If you cannot connect to the docker service, use sudo usermod -aG docker to add the user account to the docker group):

```
/etc/micronets/micronets-manager.d/mm-container pull
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/micronets-manager.d
/mm-container pull
nccoe-build-3: Pulling from micronets-docker/micronets-manager-api
Digest: sha256:dcaf5c0c0a504844733ead8992666f30b213aa594367ef079245a9d3b7e35cad
Status: Image is up to date for community.cablelabs.com:4567/micronets-docker/micron
ets-manager-api:nccoe-build-3
community.cablelabs.com:4567/micronets-docker/micronets-manager-api:nccoe-build-3
```

6. Complete the following steps to configure NGINX for the Micronets Manager:

   d. The Micronets Manager management script creates NGINX forward entries that provide a unique URI for each Micronets Manager docker image. To create the infrastructure for these entries, run:

   ```
   sudo /etc/micronets/micronets-manager.d/mm-container setup-web-proxy
   ```

   You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo /etc/micronets/micronets-mana
ger.d/mm-container setup-web-proxy
Setting up directory /etc/nginx/micronets-subscriber-forwards for writing nginx conf
 files (using group 'docker')
changed ownership of '/etc/nginx/micronets-subscriber-forwards/sub-test.conf' from r
oot:root to :docker
ownership of '/etc/nginx/micronets-subscriber-forwards' retained as root:docker
mode of '/etc/nginx/micronets-subscriber-forwards' retained as 0775 (rwxrwxr-x)
mode of '/etc/nginx/micronets-subscriber-forwards/sub-test.conf' changed from 0644 (
rw-r--r--) to 0664 (rw-rw-r--)
------------------------------------------------------------------------------------
-------
NOTE: Add the following line to and/all nginx 'server' blocks (e.g. files in '/etc/n
ginx/sites-available/')

  include /etc/nginx/micronets-subscriber-forwards/*.conf;
------------------------------------------------------------------------------------
-------
```

7. This sets up the folder to dynamically create forwarding entries for Micronets Manager instances as they are created/removed. But the site files in /etc/nginx/sites-available/ need the following added to the server blocks to enable forwarding subscriber operations to the correct docker container.

    a. Open the NGINX sites-available file created in:

    ```
    sudo vim /etc/nginx/sites-available/nccoe-server1.micronets.net
    ```

    b. Add the following entry to the file:

    ```
    include /etc/nginx/micronets-subscriber-forwards/*.conf;
    ```

    For example:

    ```
    server {
        server_name nccoe-server1.micronets.net;
     {…]
        include /etc/nginx/micronets-subscriber-forwards/*.conf;
    }
    ```

```
server {
        listen 443 ssl;
        listen [::]:443 ssl;
        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;
        server_name nccoe-server1.micronets.net;

        location / {
                try_files $uri $uri/ =404;
        }

        location /micronets/mud-manager/ {
        proxy_pass      http://localhost:8888/;
        }

        location /registry/devices {
                proxy_pass      http://localhost:3082/vendors/;
        }
        location /mud/{
                proxy_pass      http://localhost:3082/registry/;
        }

        ssl_certificate /home/micronets-dev/Projects/micronets/cert/nccoe-server1_micronets_n
et.crt;
        ssl_certificate_key /home/micronets-dev/Projects/micronets/cert/nccoe-server1_microne
ts_net.key;

        include /etc/nginx/micronets-subscriber-forwards/*.conf;
}
~
~
```

8. Complete the following steps to configure the Micronets Manager to communicate with other Micronets services on the server:

    a. Open the ***docker-compose.yml*** file by entering the following command:

    ```
    sudo vim /etc/micronets/micronets-manager.d/docker-compose.yml
    ```

    b. Modify the following environmental variables in the *docker-compose.yml* file. Replace **<ServerURL>** with your server URL:

    ```
    MM_API_PUBLIC_BASE_URL: https://<ServerURL>/sub/${MM_SUBSCRIBER_ID}/api
    MM_APP_PUBLIC_BASE_URL: https:// <ServerURL>/sub/${MM_SUBSCRIBER_ID}/app
    MM_IDENTITY_SERVER_BASE_URL: https://<ServerURL>:8888/
    MM_MSO_PORTAL_BASE_URL: https:// <ServerURL>/micronets/mso-portal
    MM_MUD_MANAGER_BASE_URL: https:// <ServerURL>/micronets/mud-manager
    MM_MUD_REGISTRY_BASE_URL: https:// <ServerURL>/micronets/mud/v1
    MM_GATEWAY_WEBSOCKET_BASE_URL: wss://<ServerURL>:5050/micronets/v1/ws-
    proxy/gw
    ```

```
      com.cablelabs.micronets.resource-type: mm-mongo
      com.cablelabs.micronets.subscriber-id: ${MM_SUBSCRIBER_ID}
  api:
    image: "${MM_API_SOURCE_IMAGE}"
    depends_on:
      - mongodb
    mem_limit: 200m
    restart: unless-stopped
    volumes:
      - ${MM_CERTS_DIR}:/usr/src/micronets-manager/certs:ro
    networks:
      - mm-priv-network
    command: ["node", "--inspect=0.0.0.0:9229", "api/"]
    environment:
      NODE_ENV: production
      MM_API_LISTEN_HOST: 0.0.0.0
      MM_API_LISTEN_PORT: 3030
      MM_MONGO_DB_URL: mongodb://mongodb/micronets
      MM_SUBSCRIBER_ID: ${MM_SUBSCRIBER_ID}
      MM_API_PUBLIC_BASE_URL: https://nccoe-server1.micronets.net/sub/${MM_SUBSCRIBE
R_ID}/api
      MM_APP_PUBLIC_BASE_URL: https://nccoe-server1.micronets.net/sub/${MM_SUBSCRIBE
R_ID}/app
      MM_IDENTITY_SERVER_BASE_URL: http://nccoe-server1.micronets.net:8888/
      MM_MSO_PORTAL_BASE_URL: https://nccoe-server1.micronets.net
      MM_MSO_PORTAL_AUTH_SECRET: ${MM_MSO_SECRET}
      MM_MUD_MANAGER_BASE_URL: http://nccoe-server1.micronets.net:8888
      MM_MUD_REGISTRY_BASE_URL: https://nccoe-server1.micronets.net/mud/v1
      MM_GATEWAY_WEBSOCKET_BASE_URL: wss://nccoe-server1.micronets.net:5050/micronet
s/v1/ws-proxy/gw
    labels:
      com.cablelabs.micronets.resource-type: mm-api
      com.cablelabs.micronets.subscriber-id: ${MM_SUBSCRIBER_ID}
```

4.1.10 Micronets Websocket Proxy

This section describes the CableLabs Micronets Websocket Proxy, which, for this implementation, is a cloud-provided service. This implementation leverages the nccoe-build-3 branch of CableLabs Micronets Websocket Proxy Git release. This service can be hosted by the implementer or another party. This documentation describes setting up your own Micronets Manager.

4.1.10.1 Micronets Websocket Proxy Overview

The Micronets Websocket Proxy is a service for establishing a Websocket connection between a subscriber’s gateway and Micronets Manager. This connection is leveraged to issue representational state transfer (REST) commands to the gateway and to receive event notifications from the gateway.

4.1.10.2 Configuration Overview

The following subsections document the software and network configurations for the Websocket Proxy. Please note that the MUD manager, Micronets Manager, Websocket Proxy, MUD registry, and MSO portal are all implemented on the same server, nccoe-server1.micronets.net. Many of these configurations are already covered in previous sections of this document but are repeated here for consistency.

4.1.10.2.1 Network Configuration

This server was hosted outside the lab environment on a Linode cloud-hosted Linux server. Its IP address was statically assigned.

4.1.10.2.2 Software Configuration

For this build, the server ran on an Ubuntu 18.04 LTS operating system. The Websocket Proxy runs in its own docker container and is configured to use SSL/TLS encryption.

The following software is required to install, configure, and operate the Websocket Proxy:

- an Ubuntu 18.04 LTS server reachable by the Micronets Manager and any Micronets gateways
- docker (v18.06 or higher)
- docker-compose (v1.23.1 or higher)
- curl
- Python 3.6+
- Python virtualenv package

4.1.10.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the Websocket Proxy:

- 4 GB of RAM
- 50 GB of free disk space

4.1.10.3 Setup

1. Change to the working directory by entering the following command:

   ```
   cd Projects/micronets/
   ```

   If you have not already created this directory, execute the following command:

   ```
   mkdir Projects/micronets/
   ```

   Next, change directories by entering the following command:

   ```
   cd Projects/micronets/
   ```

2. Download and install the cert generation scripts by executing the following commands:

   a. Download the script to generate the root certificates:

   ```
   curl -s -O https://raw.githubusercontent.com/cablelabs/micronets-ws-proxy/nccoe-build-3/bin/gen-root-cert
   ```

   b. Download the script to generate leaf certificates:

   ```
   curl -s -O https://raw.githubusercontent.com/cablelabs/micronets-ws-proxy/nccoe-build-3/bin/gen-leaf-cert
   ```

   c. Install both scripts by executing the following command:

   ```
   sudo install -v -o root -m 755 -D -t /etc/micronets/micronets-ws-proxy.d/ gen-*-cert
   ```

   You should see output similar to the following:

   ```
   [micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 -D -]
   t /etc/micronets/micronets-ws-proxy.d/ gen-*-cert
   [[sudo] password for micronets-dev:                                                  ]
   'gen-leaf-cert' -> '/etc/micronets/micronets-ws-proxy.d/gen-leaf-cert'
   'gen-root-cert' -> '/etc/micronets/micronets-ws-proxy.d/gen-root-cert'
   ```

3. Create the root certificate for the Websocket Proxy:

   ```
   /etc/micronets/micronets-ws-proxy.d/gen-root-cert --cert-basename micronets-ws-root \
       --subject-org-name "Micronets Websocket Root Cert" \
       --expiration-in-days 3650
   ```

   You should see output similar to the following:

```
Creating EC parameter file micronets-ws-root.ecparams.pem for EC prime256v1
Creating private key file (micronets-ws-root.key.pem) from micronets-ws-root.ecparams
.pem
Creating certificate signing request file (micronets-ws-root.csr.pem) using key file
micronets-ws-root.key.pem
Can't load /home/micronets-dev/.rnd into RNG
139696849768896:error:2406F079:random number generator:RAND_load_file:Cannot open fil
e:../crypto/rand/randfile.c:88:Filename=/home/micronets-dev/.rnd
Creating extension option file (micronets-ws-root.cert_ext.txt)
Creating self-signed root CA certificate (micronets-ws-root.cert.pem)
Signature ok
subject=O = Micronets Websocket Root Cert
Getting Private key
Successfully generated root certificate "micronets-ws-root.cert.pem"/"micronets-ws-ro
ot.cert.der"
```

4. Create the Websocket Proxy's server certificate and private key by entering the following command (Note: This certificate and key host the Websocket Proxy server):

```
/etc/micronets/micronets-ws-proxy.d/gen-leaf-cert --cert-basename micronets-ws-
proxy \
  --subject-org-name "Micronets Websocket Proxy Cert" \
  --expiration-in-days 3650 \
  --ca-certfile micronets-ws-root.cert.pem \
  --ca-keyfile micronets-ws-root.key.pem
```

You should see output similar to the following:

```
Creating EC parameter file micronets-ws-proxy.ecparams.pem for EC prime256v1
Creating private key file (micronets-ws-proxy.key.pem) from micronets-ws-proxy.ecpara
ms.pem
Creating certificate signing request file (micronets-ws-proxy.csr.pem) using key file
 micronets-ws-root.key.pem
Can't load /home/micronets-dev/.rnd into RNG
139824120451520:error:2406F079:random number generator:RAND_load_file:Cannot open fil
e:../crypto/rand/randfile.c:88:Filename=/home/micronets-dev/.rnd
Creating extension option file (micronets-ws-proxy.cert_ext.txt)
Signing leaf certificate (micronets-ws-proxy.cert.pem) with micronets-ws-root.key.pem
Signature ok
subject=O = Micronets Websocket Proxy Cert
Getting CA Private Key
Successfully generated leaf certificate "micronets-ws-proxy.cert.pem"/"micronets-ws-p
roxy.cert.der"
```

5. Combine the private key and certificate into one file by entering the following command:

```
cat micronets-ws-proxy.cert.pem micronets-ws-proxy.key.pem \
```

```
> micronets-ws-proxy.pkeycert.pem
```

6. Generate the client certificate and key to be used by the Micronets Manager to connect to the Websocket Proxy (Note: These files will enable the Micronets Manager to connect to the proxy):

```
/etc/micronets/micronets-ws-proxy.d/gen-leaf-cert --cert-basename micronets-manager \
    --subject-org-name "Micronets Manager Websocket Client Cert" \
    --expiration-in-days 3650 \
    --ca-certfile micronets-ws-root.cert.pem \
    --ca-keyfile micronets-ws-root.key.pem
```

You should see output similar to the following:

```
Creating EC parameter file micronets-manager.ecparams.pem for EC prime256v1
Creating private key file (micronets-manager.key.pem) from micronets-manager.ecparams
.pem
Creating certificate signing request file (micronets-manager.csr.pem) using key file
micronets-ws-root.key.pem
Can't load /home/micronets-dev/.rnd into RNG
140018969551296:error:2406F079:random number generator:RAND_load_file:Cannot open fil
e:../crypto/rand/randfile.c:88:Filename=/home/micronets-dev/.rnd
Creating extension option file (micronets-manager.cert_ext.txt)
Signing leaf certificate (micronets-manager.cert.pem) with micronets-ws-root.key.pem
Signature ok
subject=O = Micronets Manager Websocket Client Cert
Getting CA Private Key
Successfully generated leaf certificate "micronets-manager.cert.pem"/"micronets-manag
er.cert.der"
```

7. Combine the private key and certificate into one file by entering the following command:

```
cat micronets-manager.cert.pem micronets-manager.key.pem \
    > micronets-manager.pkeycert.pem
```

8. Generate the certificate and key to be used by the Micronets Gateway to connect to the Websocket Proxy (Note: These files will enable the Micronets Gateway to connect to the proxy):

```
/etc/micronets/micronets-ws-proxy.d/gen-leaf-cert --cert-basename micronets-gw-service \
    --subject-org-name "Micronets Gateway Service Websocket Client Cert" \
    --expiration-in-days 3650 \
```

```
    --ca-certfile micronets-ws-root.cert.pem \
    --ca-keyfile micronets-ws-root.key.pem
```

You should see output similar to the following:

```
Creating EC parameter file micronets-gw-service.ecparams.pem for EC prime256v1
Creating private key file (micronets-gw-service.key.pem) from micronets-gw-service.ec
params.pem
Creating certificate signing request file (micronets-gw-service.csr.pem) using key fi
le micronets-ws-root.key.pem
Can't load /home/micronets-dev/.rnd into RNG
140269637321152:error:2406F079:random number generator:RAND_load_file:Cannot open fil
e:../crypto/rand/randfile.c:88:Filename=/home/micronets-dev/.rnd
Creating extension option file (micronets-gw-service.cert_ext.txt)
Signing leaf certificate (micronets-gw-service.cert.pem) with micronets-ws-root.key.p
em
Signature ok
subject=O = Micronets Gateway Service Websocket Client Cert
Getting CA Private Key
Successfully generated leaf certificate "micronets-gw-service.cert.pem"/"micronets-gw
-service.cert.der"
```

9. Combine the private key and certificate into one file by entering the following command:

```
cat micronets-gw-service.cert.pem micronets-gw-service.key.pem \
    > micronets-gw-service.pkeycert.pem
```

10. Download and install the management script by entering the following commands:

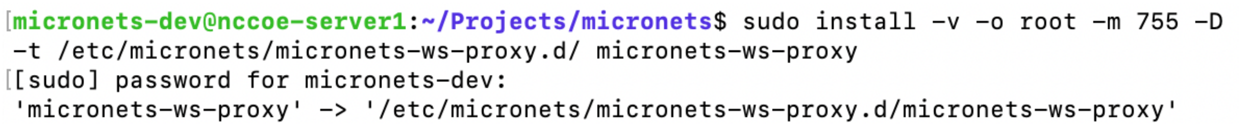
    a. Download the micronets-ws-proxy script:

    ```
    curl -s -O https://raw.githubusercontent.com/cablelabs/micronets-ws-proxy/nccoe-build-3/bin/micronets-ws-proxy
    ```

    b. Install the script to the appropriate directory:

    ```
    sudo install -v -o root -m 755 -D -t /etc/micronets/micronets-ws-proxy.d/ micronets-ws-proxy
    ```

    You should see output similar to the following:

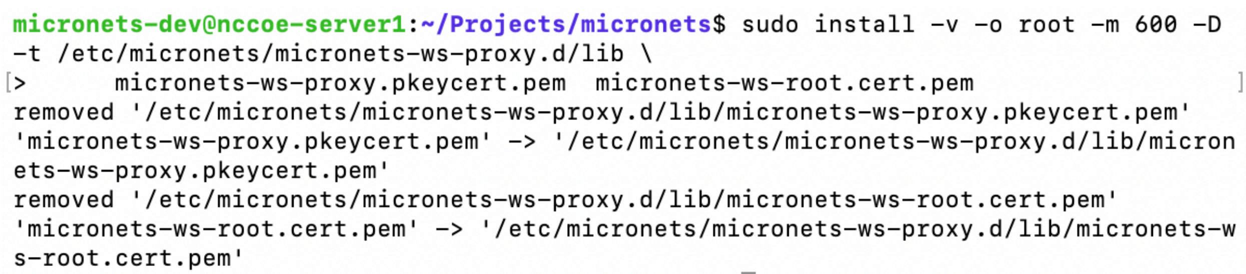


```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 -D
-t /etc/micronets/micronets-ws-proxy.d/ micronets-ws-proxy
[sudo] password for micronets-dev:
'micronets-ws-proxy' -> '/etc/micronets/micronets-ws-proxy.d/micronets-ws-proxy'
```

11. Copy the Websocket Proxy server cert and key for use by the Websocket Proxy docker container:

```
sudo install -v -o root -m 600 -D -t /etc/micronets/micronets-ws-proxy.d/lib \
  micronets-ws-proxy.pkeycert.pem  micronets-ws-root.cert.pem
```

You should see output similar to the following:

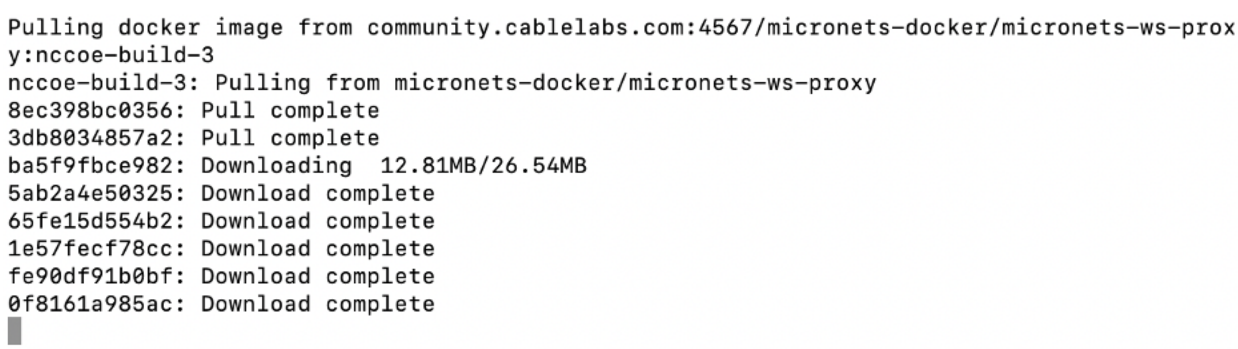


```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 600 -D
-t /etc/micronets/micronets-ws-proxy.d/lib \
>      micronets-ws-proxy.pkeycert.pem  micronets-ws-root.cert.pem
removed '/etc/micronets/micronets-ws-proxy.d/lib/micronets-ws-proxy.pkeycert.pem'
'micronets-ws-proxy.pkeycert.pem' -> '/etc/micronets/micronets-ws-proxy.d/lib/micron
ets-ws-proxy.pkeycert.pem'
removed '/etc/micronets/micronets-ws-proxy.d/lib/micronets-ws-root.cert.pem'
'micronets-ws-root.cert.pem' -> '/etc/micronets/micronets-ws-proxy.d/lib/micronets-w
s-root.cert.pem'
```

12. Download the Micronets Websocket Proxy docker image (Note: If you cannot connect to the docker service, use sudo usermod -aG docker to add the user account to the docker group):

```
/etc/micronets/micronets-ws-proxy.d/micronets-ws-proxy docker-pull
```

You should see output similar to the following:



```
Pulling docker image from community.cablelabs.com:4567/micronets-docker/micronets-ws-prox
y:nccoe-build-3
nccoe-build-3: Pulling from micronets-docker/micronets-ws-proxy
8ec398bc0356: Pull complete
3db8034857a2: Pull complete
ba5f9fbce982: Downloading  12.81MB/26.54MB
5ab2a4e50325: Download complete
65fe15d554b2: Download complete
1e57fecf78cc: Download complete
fe90df91b0bf: Download complete
0f8161a985ac: Download complete
```

13. Start the Websocket Proxy:

```
/etc/micronets/micronets-ws-proxy.d/micronets-ws-proxy docker-run
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/micronets-ws-proxy.
d/micronets-ws-proxy docker-run
Starting container "micronets-ws-proxy-service" from community.cablelabs.com:4567/mi
cronets-docker/micronets-ws-proxy:nccoe-build-3 (on 0.0.0.0:5050)
1ca41776f2be42b488a87b2bf07a80ef4e82d9320d8f1106fe060b5cfb0ef7e1
```

14. Verify that the Websocket Proxy is running:

```
/etc/micronets/micronets-ws-proxy.d/micronets-ws-proxy docker-logs
```

You should see output similar to the following:

```
2020-04-24T17:34:07.535588025Z 2020-04-24 17:34:07,535 micronets-ws-proxy: INFO Serv
er cert/key: /app/lib/micronets-ws-proxy.pkeycert.pem
2020-04-24T17:34:07.536263687Z 2020-04-24 17:34:07,536 micronets-ws-proxy: INFO CA p
ath: None
2020-04-24T17:34:07.536462663Z 2020-04-24 17:34:07,536 micronets-ws-proxy: INFO Addi
tional CA certs: /app/lib/micronets-ws-root.cert.pem
2020-04-24T17:34:07.537057042Z 2020-04-24 17:34:07,536 micronets-ws-proxy: INFO URL
Path Prefix: /micronets/v1/ws-proxy/
2020-04-24T17:34:07.537249748Z 2020-04-24 17:34:07,537 micronets-ws-proxy: INFO Repo
rt Interval: 0
2020-04-24T17:34:07.544754798Z 2020-04-24 17:34:07,543 micronets-ws-proxy: INFO Load
ing proxy certificate/key from /app/lib/micronets-ws-proxy.pkeycert.pem
2020-04-24T17:34:07.546560336Z 2020-04-24 17:34:07,546 micronets-ws-proxy: INFO Star
ting micronets websocket proxy on 0.0.0.0 port 5050...
2020-04-24T17:34:07.546863216Z 2020-04-24 17:34:07,546 micronets-ws-proxy: INFO Clie
nts may connect to wss://0.0.0.0:5050/micronets/v1/ws-proxy/*
```

15. Verify the Websocket Proxy credentials by executing the following steps:

    a. Download the Websocket test client script:

```
curl -O https://raw.githubusercontent.com/cablelabs/micronets-ws-proxy/nccoe-build-3/bin/websocket-test-client.py
```

    b. Download the requirements text file:

```
curl -O https://raw.githubusercontent.com/cablelabs/micronets-ws-proxy/nccoe-build-3/requirements.txt
```

c. Clear out the nonroot installation of virtualenv, and set the Python interpreter to use Python 3.6 for the script installation:

```
virtualenv --clear -p $(which python3.6) $PWD/virtualenv
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ virtualenv --clear -p $(which pyth
on3.6) $PWD/virtualenv
Running virtualenv with interpreter /usr/bin/python3.6
Deleting tree /home/micronets-dev/Projects/micronets/virtualenv/lib/python3.6
Not deleting /home/micronets-dev/Projects/micronets/virtualenv/bin
Using base prefix '/usr'
New python executable in /home/micronets-dev/Projects/micronets/virtualenv/bin/pytho
n3.6
Not overwriting existing python script /home/micronets-dev/Projects/micronets/virtua
lenv/bin/python (you must use /home/micronets-dev/Projects/micronets/virtualenv/bin/
python3.6)
Installing setuptools, pkg_resources, pip, wheel...done.
```

d. Install virtualenv and pass the requirements text file:

```
./virtualenv/bin/pip install -r requirements.txt
```

You should see output similar to the following:

```
Installing collected packages: pip, pipdeptree, blinker, aiofiles, MarkupSafe, Jinja
2, multidict, itsdangerous, sortedcontainers, click, h11, hpack, hyperframe, h2, wsp
roto, typing-extensions, Hypercorn, Quart, setuptools, websockets, wheel
  Attempting uninstall: pip
    Found existing installation: pip 20.1
    Uninstalling pip-20.1:
      Successfully uninstalled pip-20.1
  Attempting uninstall: setuptools
    Found existing installation: setuptools 46.1.3
    Uninstalling setuptools-46.1.3:
      Successfully uninstalled setuptools-46.1.3
  Attempting uninstall: wheel
    Found existing installation: wheel 0.34.2
    Uninstalling wheel-0.34.2:
      Successfully uninstalled wheel-0.34.2
Successfully installed Hypercorn-0.1.0 Jinja2-2.10.1 MarkupSafe-1.1.1 Quart-0.6.1 ai
ofiles-0.3.2 blinker-1.4 click-6.7 h11-0.7.0 h2-3.0.1 hpack-3.0.0 hyperframe-5.1.0 i
tsdangerous-0.24 multidict-4.3.1 pip-19.0.3 pipdeptree-0.13.2 setuptools-41.0.0 sort
edcontainers-2.0.4 typing-extensions-3.6.5 websockets-5.0.1 wheel-0.33.1 wsproto-0.1
1.0
```

e. Run the Websocket test client script:

```
./virtualenv/bin/python websocket-test-client.py \
    --client-cert micronets-manager.pkeycert.pem \
    --ca-cert micronets-ws-root.cert.pem  \
```

```
wss://localhost:5050/micronets/v1/ws-proxy/test/mm
```

You should see output similar to the following:

```
Startup...
Loading test client certificate from micronets-manager.pkeycert.pem
Loading CA certificate from micronets-ws-root.cert.pem
ws-test-client: Opening websocket to wss://localhost:5050/micronets/v1/ws-proxy/test
/mm...
ws-test-client: Connected to wss://localhost:5050/micronets/v1/ws-proxy/test/mm.
ws-test-client: Sending HELLO message...
ws-test-client: > sending hello message:  {"message": {"messageId": 0, "messageType"
: "CONN:HELLO", "requiresResponse": false, "peerClass": "micronets-ws-test-client",
"peerId": "12345678"}}
ws-test-client: Waiting for HELLO message...
```

f. Verify communication from the test client to the Websocket Proxy by checking the logs:

```
/etc/micronets/micronets-ws-proxy.d/micronets-ws-proxy docker-logs
```

You should see output similar to the following:

```
2020-05-05T17:52:43.366375745Z 2020-05-05 17:52:43,366 micronets-ws-proxy: INFO ws_c
onnected: client 139799007351752: Received HELLO message:
2020-05-05T17:52:43.367278293Z 2020-05-05 17:52:43,367 micronets-ws-proxy: INFO {
2020-05-05T17:52:43.367291343Z   "message": {
2020-05-05T17:52:43.367295073Z     "messageId": 0,
2020-05-05T17:52:43.367298363Z     "messageType": "CONN:HELLO",
2020-05-05T17:52:43.367301603Z     "peerClass": "micronets-ws-test-client",
2020-05-05T17:52:43.367304803Z     "peerId": "12345678",
2020-05-05T17:52:43.367307973Z     "requiresResponse": false
2020-05-05T17:52:43.367310943Z   }
2020-05-05T17:52:43.367313733Z }
2020-05-05T17:52:43.367543683Z 2020-05-05 17:52:43,367 micronets-ws-proxy: INFO ws_c
onnected: client 139799007351752 is the first connected to /micronets/v1/ws-proxy/te
st/mm
2020-05-05T17:52:43.367758972Z 2020-05-05 17:52:43,367 micronets-ws-proxy: INFO mess
age: {'message': {'messageId': 0, 'messageType': 'CONN:HELLO', 'requiresResponse': F
alse, 'peerClass': 'micronets-ws-test-client', 'peerId': '12345678'}}
2020-05-05T17:52:43.368011242Z 2020-05-05 17:52:43,367 micronets-ws-proxy: INFO
2020-05-05T17:52:43.368021152Z -----------------------------------------------------
----------------------------------
2020-05-05T17:52:43.368024452Z WEBSOCKET MEETUP TABLE REPORT FOR 0.0.0.0:5050//micro
nets/v1/ws-proxy/
2020-05-05T17:52:43.368027442Z
2020-05-05T17:52:43.368030162Z   MEETUP ID: test/mm
2020-05-05T17:52:43.368032862Z     Client 1: Client 139799007351752 (peer: 12345678)
 @ ('172.17.0.1', 56004))
2020-05-05T17:52:43.368035672Z     Client 2: Not connected
2020-05-05T17:52:43.368038352Z -----------------------------------------------------
-----------------------------------
2020-05-05T17:52:43.368041102Z
2020-05-05T17:52:43.368244001Z 2020-05-05 17:52:43,368 micronets-ws-proxy: INFO ws_c
lient 139799007351752: wait_for_peer: waiting for peer on test/mm...
```

16. Save the *micronets-manager.pkeycert.pem, micronets-gw-service.pkeycert.pem,* and *micronets-ws-root.cert.pem* files for configuring the Micronets Manager and Micronets Gateway components.

4.1.11 Micronets iPhone Application for Device Onboarding

This section describes the CableLabs Micronets iPhone application, which is a mobile application used for onboarding DPP-capable devices. This implementation leverages the latest CableLabs Micronets iPhone application [Git release](). This documentation describes setting up your own Micronets iPhone application.

4.1.11.1 Micronets iPhone Application Overview

The Micronets iPhone application is responsible for sending onboarding requests and related elements to the MSO portal when the user initiates the onboarding process on the Micronets Proto-Pi device and scans the QR code. If building with an Android phone, follow the documentation provided here: https://github.com/cablelabs/micronets-mobile/blob/nccoe-build-3/README.md#android

4.1.11.2 Configuration Overview

The following subsections document the software and network configurations for the Micronets iPhone application.

4.1.11.2.1 Network Configuration

The mobile phone on which the Micronets application is being installed should have internet access via either the cellular network or Wi-Fi.

4.1.11.2.2 Software Configuration

The following software is required to install, configure, and operate the Micronets iPhone application:

- macOS (minimum version 10.13; High Sierra)
- Apple iOS Developer license
- Node (minimum version 8)
- Cordova (version 8.0.0; problems with version 9)
- Xcode (minimum version 9.2)
- ImageMagick
- Brew

4.1.11.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the Micronets iPhone application:

- Apple computing system (laptop or desktop)
- Apple iPhone (any model compatible with iOS 10.3 and above)

4.1.11.3 Setup

4.1.11.3.1 Install Dependencies

1. Install Node by entering the following command in the terminal:

```
brew install node
```

2. Install ImageMagick by entering the following command in the terminal:

```
brew install imagemagick
```

3. Install Cordova version 8.0.0 by entering the following command:

```
sudo npm install -g cordova@8.0.0
```

4. Install ios-deploy, which Cordova uses to cable-load the application, by entering the following command:

```
sudo npm install -g --unsafe-perm=true ios-deploy
```

Note: The `unsafe-perm` flag is required on macOS versions El Capitan and higher.

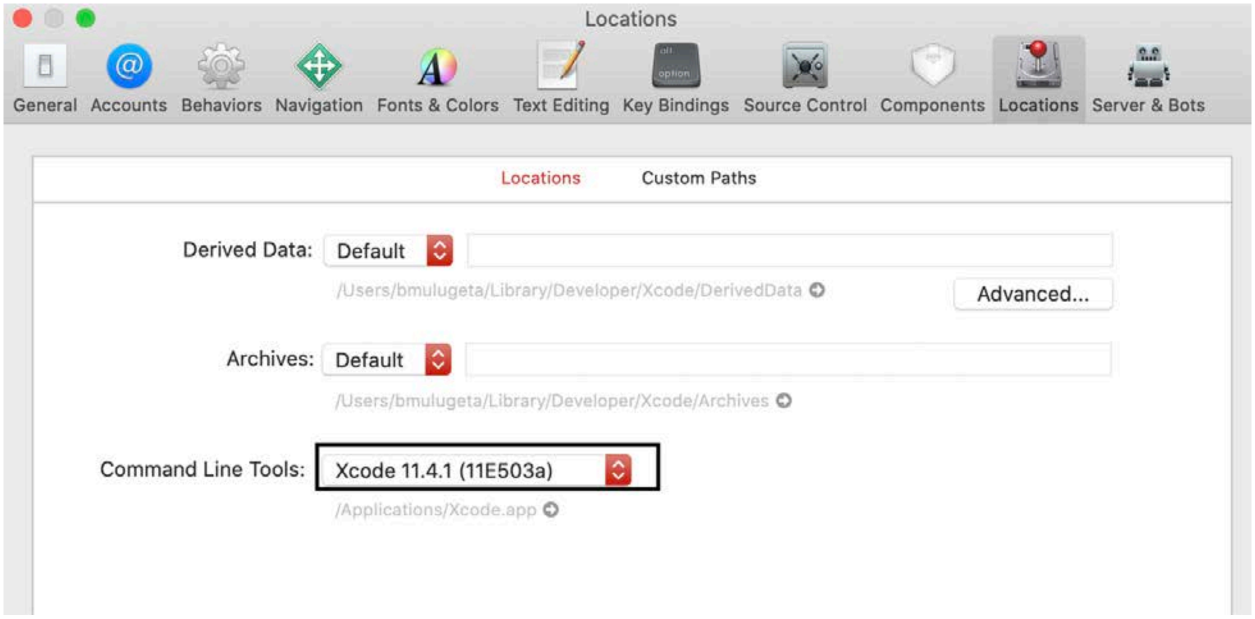
If you run into an EACCES: permission denied error, attempt the following fixes:

```
sudo chown -R $USER:$GROUP ~/.npm

sudo chown -R $USER:$GROUP ~/.config
```

5. Open Xcode, and add Xcode to your command-line tools:

Preferences > Location > Command Line Tools

Select your Xcode version as seen in screenshot below:



4.1.11.3.2 Build Micronets iPhone Application

1. Check out the repo that contains the Micronets mobile application build by entering the following command:

   `git clone https://www.github.com/cablelabs/micronets-mobile.git`

```
MM252521-PC:cablelabs bmulugeta$ git clone https://www.github.com/cablelabs/micronets-mobile.
git
Cloning into 'micronets-mobile'...
warning: redirecting to https://github.com/cablelabs/micronets-mobile.git/
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 213 (delta 3), reused 6 (delta 2), pack-reused 206
Receiving objects: 100% (213/213), 12.48 MiB | 502.00 KiB/s, done.
Resolving deltas: 100% (86/86), done.
```

2. Enter the Micronets mobile directory by entering the following command:

   `cd micronets-mobile`

3. Add the target platform by entering the following command:

   `cordova platform add ios`

```
MM252521-PC:micronets-mobile bmulugeta$ cordova platform add ios
Using cordova-fetch for cordova-ios@^4.5.5
Adding ios project...
Creating Cordova project for the iOS platform:
        Path: platforms/ios
        Package: com.cablelabs.micronets.mobile
        Name: Micronets
iOS project created with cordova-ios@4.5.5
Discovered plugin "cordova-plugin-app-preferences" in config.xml. Adding it to the project
Installing "cordova-plugin-app-preferences" for ios
Platform android not found: skipping
Adding cordova-plugin-app-preferences to package.json
Saved plugin info for "cordova-plugin-app-preferences" to config.xml
Discovered plugin "cordova-plugin-statusbar" in config.xml. Adding it to the project
Installing "cordova-plugin-statusbar" for ios
Adding cordova-plugin-statusbar to package.json
Saved plugin info for "cordova-plugin-statusbar" to config.xml
Discovered plugin "cordova-plugin-whitelist" in config.xml. Adding it to the project
Installing "cordova-plugin-whitelist" for ios
Adding cordova-plugin-whitelist to package.json
Saved plugin info for "cordova-plugin-whitelist" to config.xml
Discovered plugin "phonegap-plugin-barcodescanner" in config.xml. Adding it to the project
Installing "phonegap-plugin-barcodescanner" for ios
Adding phonegap-plugin-barcodescanner to package.json
Saved plugin info for "phonegap-plugin-barcodescanner" to config.xml
Discovered plugin "cordova-plugin-cache-clear" in config.xml. Adding it to the project
Installing "cordova-plugin-cache-clear" for ios
Adding cordova-plugin-cache-clear to package.json
Saved plugin info for "cordova-plugin-cache-clear" to config.xml
ios settings bundle was successfully generated
--save flag or autosave detected
Saving ios@~4.5.5 into config.xml file ...
```

4. Generate iOS icon set by entering the following command:

```
npx app-icon generate
```

You should see the following output:

```
[MM252521-PC:micronets-mobile bmulugeta$ npx app-icon generate
npx: installed 25 in 2.133s
Found iOS iconset: platforms/ios/Micronets/Images.xcassets/AppIcon.appiconset...
    ✓ Generated icon ipad-29x29-1x.png
    ✓ Generated icon iphone-57x57-1x.png
    ✓ Generated icon iphone-40x40-3x.png
    ✓ Generated icon iphone-40x40-2x.png
    ✓ Generated icon iphone-29x29-3x.png
    ✓ Generated icon iphone-29x29-2x.png
    ✓ Generated icon iphone-29x29-1x.png
    ✓ Generated icon ipad-20x20-2x.png
    ✓ Generated icon iphone-20x20-3x.png
    ✓ Generated icon iphone-20x20-2x.png
    ✓ Generated icon ipad-20x20-1x.png
    ✓ Generated icon ipad-40x40-2x.png
    ✓ Generated icon iphone-57x57-2x.png
    ✓ Generated icon ipad-40x40-1x.png
    ✓ Generated icon iphone-60x60-2x.png
    ✓ Generated icon ipad-29x29-2x.png
    ✓ Generated icon ipad-50x50-1x.png
    ✓ Generated icon iphone-60x60-3x.png
    ✓ Generated icon ipad-72x72-1x.png
    ✓ Generated icon ipad-50x50-2x.png
    ✓ Generated icon ipad-76x76-1x.png
    ✓ Generated icon ipad-83.5x83.5-2x.png
    ✓ Generated icon ipad-76x76-2x.png
    ✓ Generated icon ipad-72x72-2x.png
    ✓ Generated icon ios-marketing-1024x1024-1x.png
    ✓ Updated Contents.json
```

5. Plug your iPhone into your computer, unlock your phone, and open to home screen. (You will need to allow developer use of the phone. You will be prompted.)

6. Run the following command to build the mobile application:

```
cordova run ios --device --buildFlag='-UseModernBuildSystem=0'
```

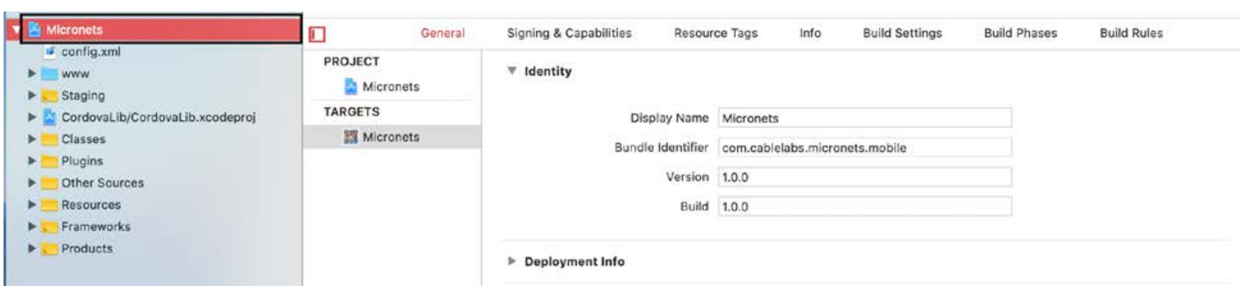
You should see output similar to the following:

```
=== BUILD TARGET Micronets OF PROJECT Micronets WITH CONFIGURATION Debug ===

Check dependencies
Code Signing Error: Signing for "Micronets" requires a development team. Select a development
 team in the Signing & Capabilities editor.
Code Signing Error: Code signing is required for product type 'Application' in SDK 'iOS 13.4'

** ARCHIVE FAILED **


The following build commands failed:
        Check dependencies
(1 failure)
(node:50941) UnhandledPromiseRejectionWarning: Error code 65 for command: xcodebuild with arg
s: -xcconfig,/Users/bmulugeta/Desktop/cablelabs/micronets-mobile/platforms/ios/cordova/build-
debug.xcconfig,-workspace,Micronets.xcworkspace,-scheme,Micronets,-configuration,Debug,-desti
nation,generic/platform=iOS,-archivePath,Micronets.xcarchive,archive,CONFIGURATION_BUILD_DIR=
/Users/bmulugeta/Desktop/cablelabs/micronets-mobile/platforms/ios/build/device,SHARED_PRECOMP
S_DIR=/Users/bmulugeta/Desktop/cablelabs/micronets-mobile/platforms/ios/build/sharedpch,-UseM
odernBuildSystem=0
(node:50941) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error origin
ated either by throwing inside of an async function without a catch block, or by rejecting a
promise which was not handled with .catch(). To terminate the node process on unhandled promi
se rejection, use the CLI flag `--unhandled-rejections=strict` (see https://nodejs.org/api/cl
i.html#cli_unhandled_rejections_mode). (rejection id: 1)
(node:50941) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In th
e future, promise rejections that are not handled will terminate the Node.js process with a n
on-zero exit code.
```

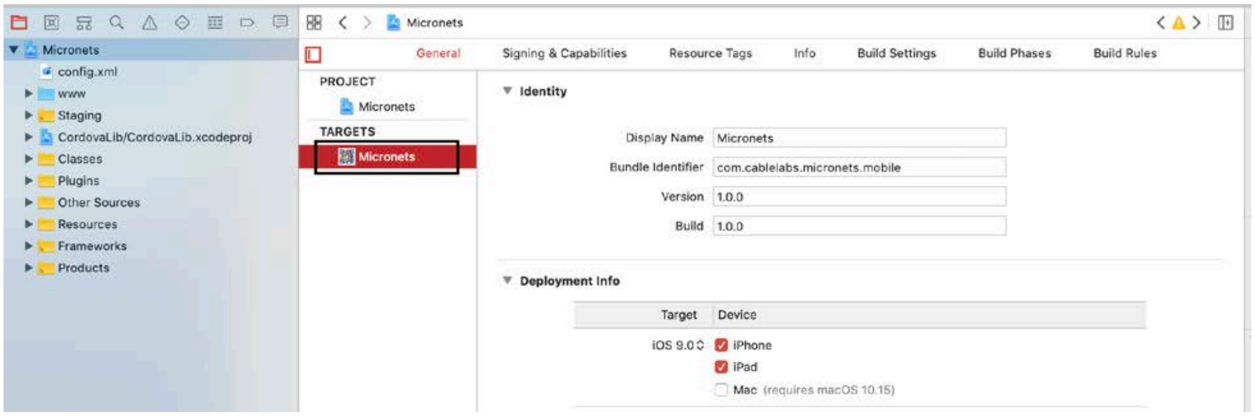
Note: This initial attempt to build is expected to fail. It is necessary to open the project in Xcode and change some settings.

7. Open the project file *platforms/ios/Micronets.xcodeproj* in Xcode.

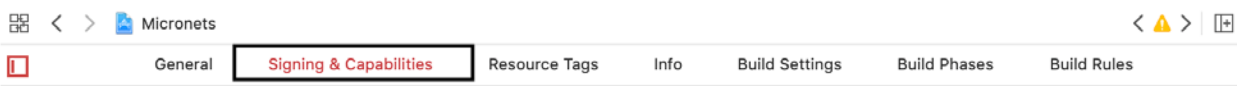
8. Click the Micronets icon in the navigator pane on the left. The properties pane should now be visible on the right:



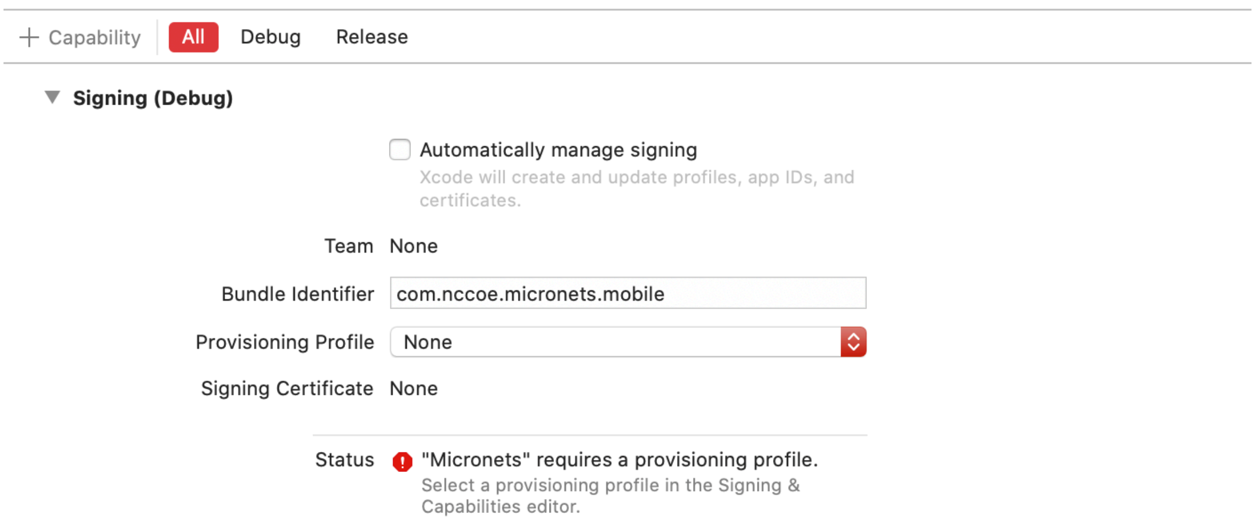
9. Select **Micronets** under **TARGETS:**



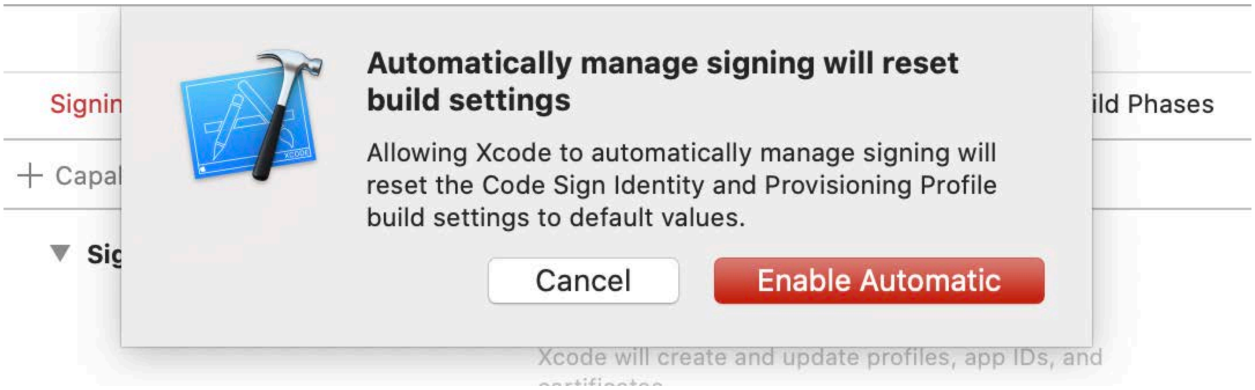
10. Select the **Signing & Capabilities** tab in the heading:



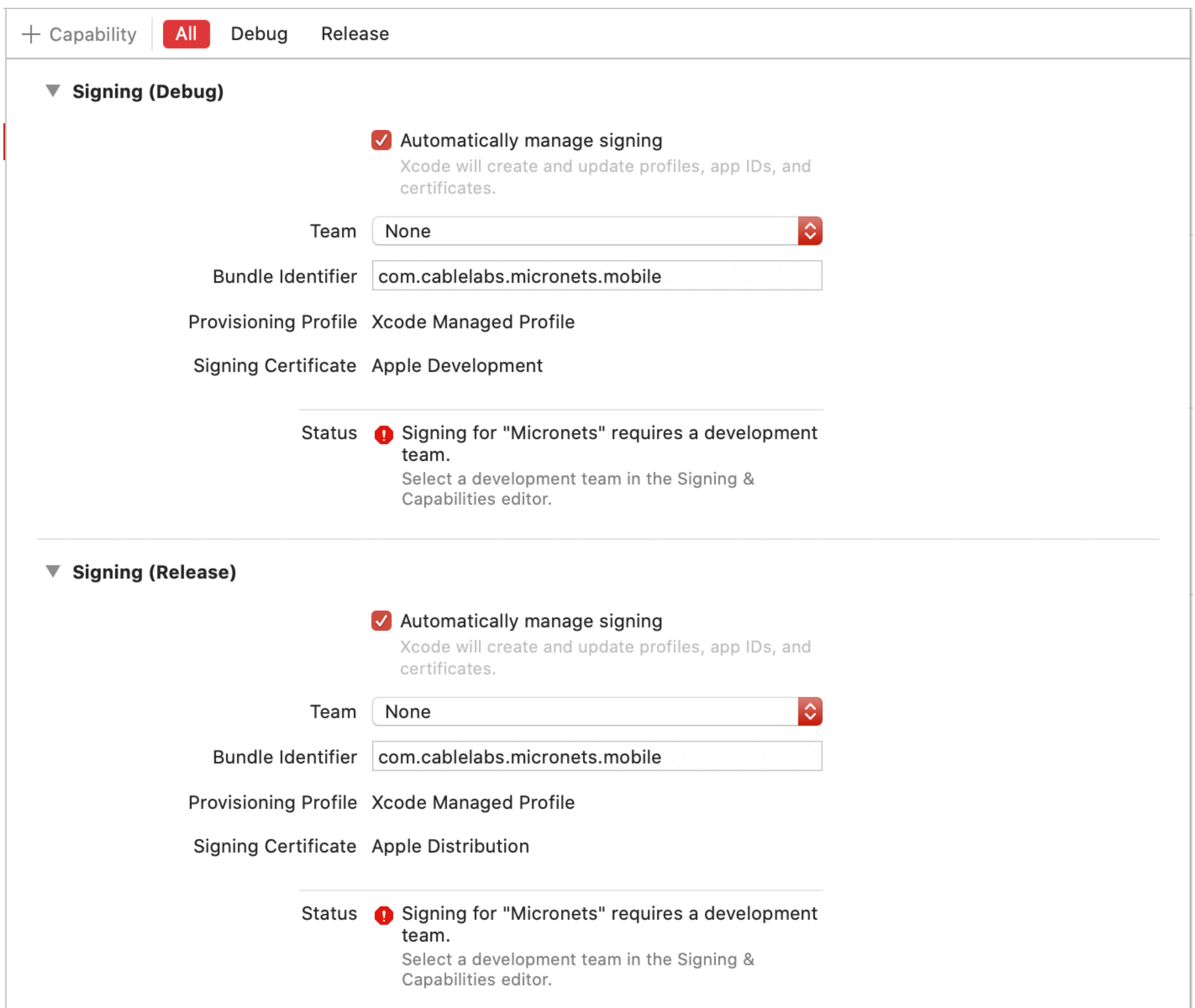
11. Ensure **Automatically manage signing** is checked:



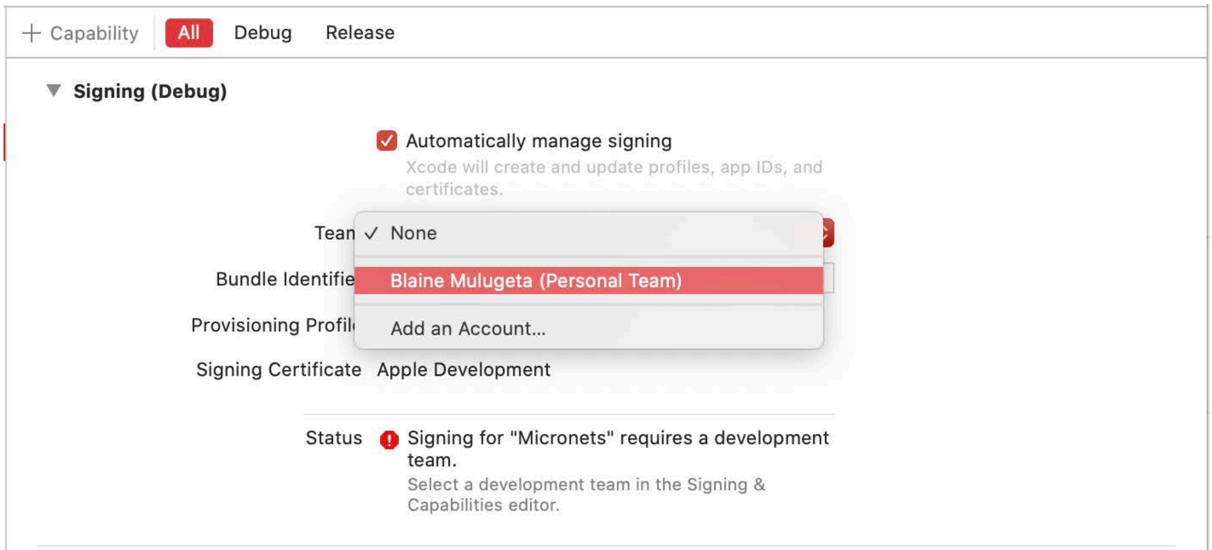
You will see the following notification. Select **Enable Automatic:**



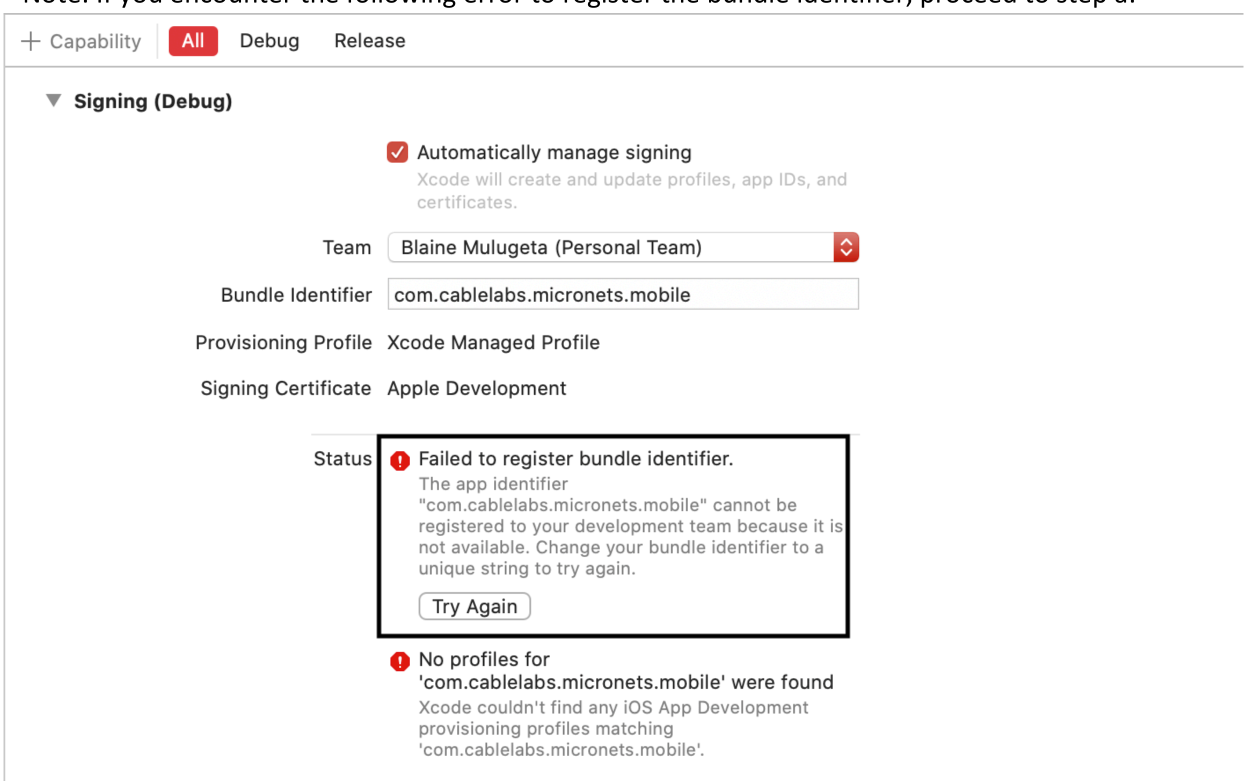
The **Automatically manage signing** setting should now be selected as seen below:



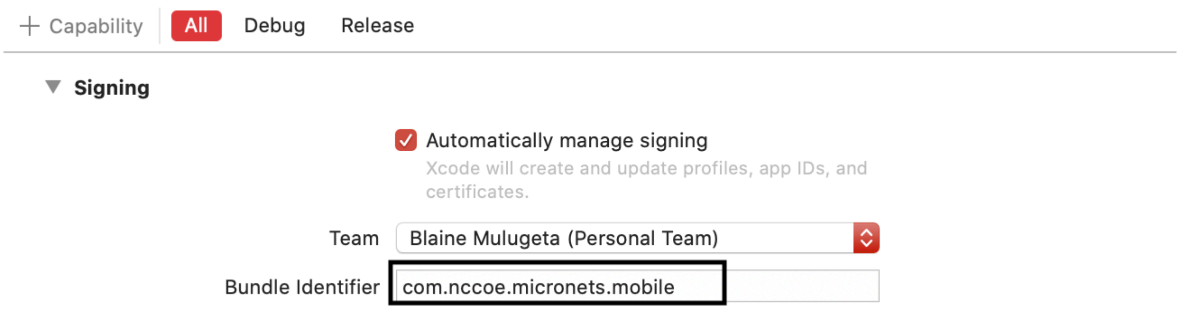
12. Ensure that your team is selected under the **Team** drop-down:



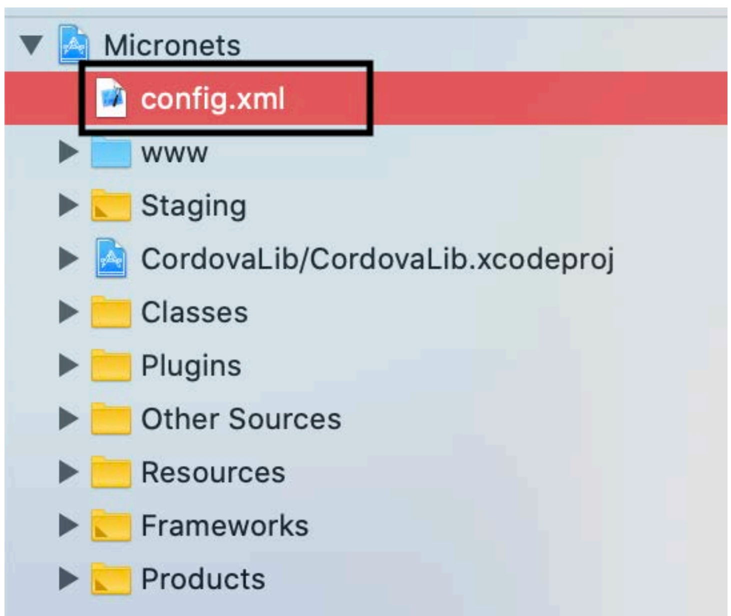
Note: If you encounter the following error to register the bundle identifier, proceed to step a:



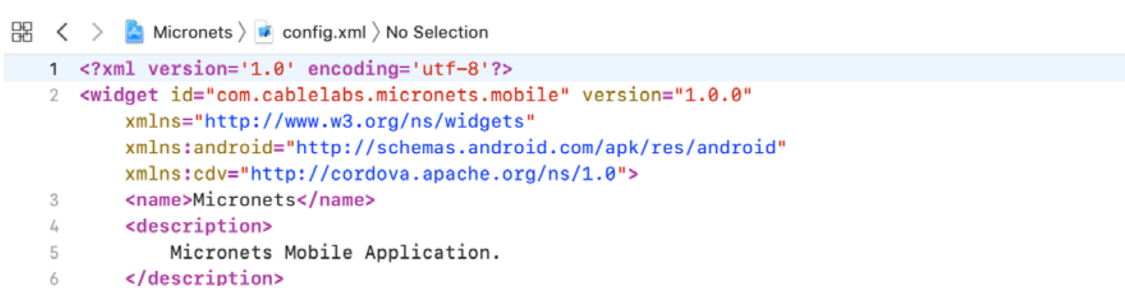
a. Change the **Bundle Identifier** to your own unique identifier:



b. Navigate to the ***config.xml*** file by selecting as shown below:



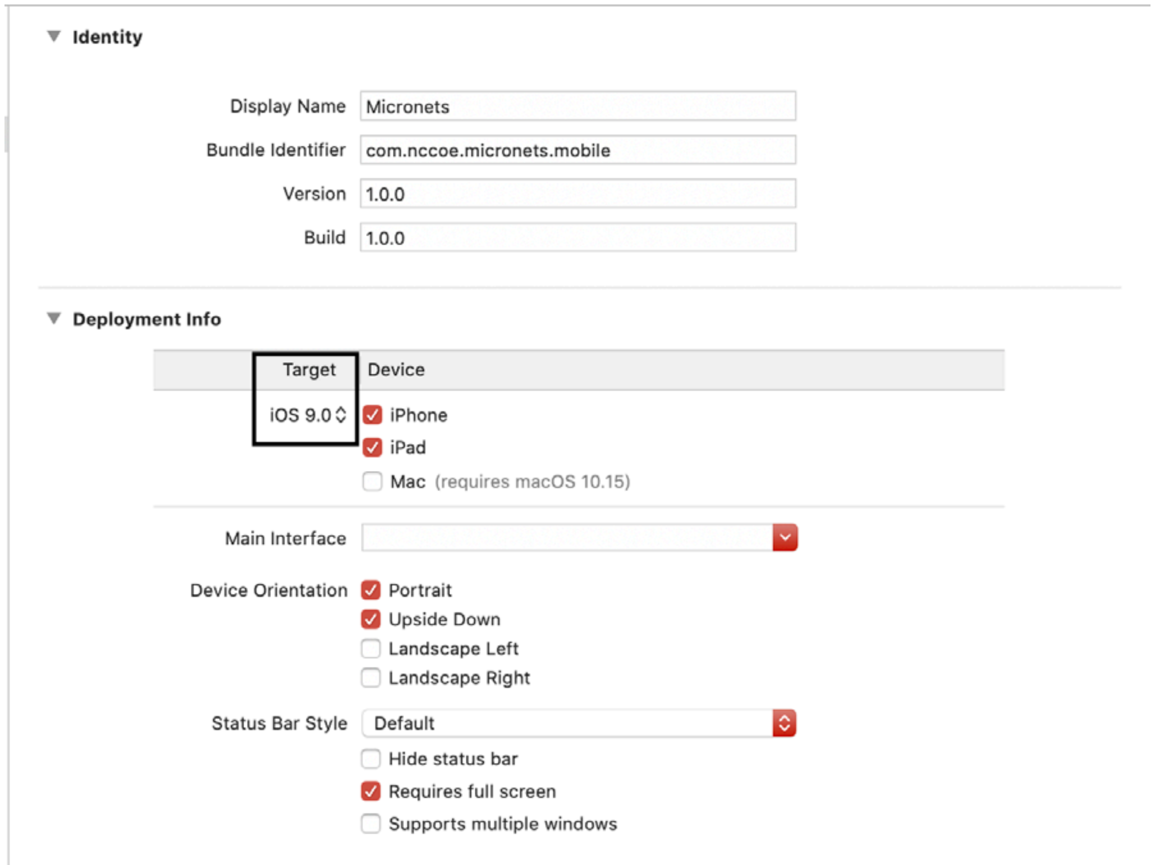
c. Modify the widget id from **com.cablelabs.micronets.mobile** to the build identifier created in step a as seen below:

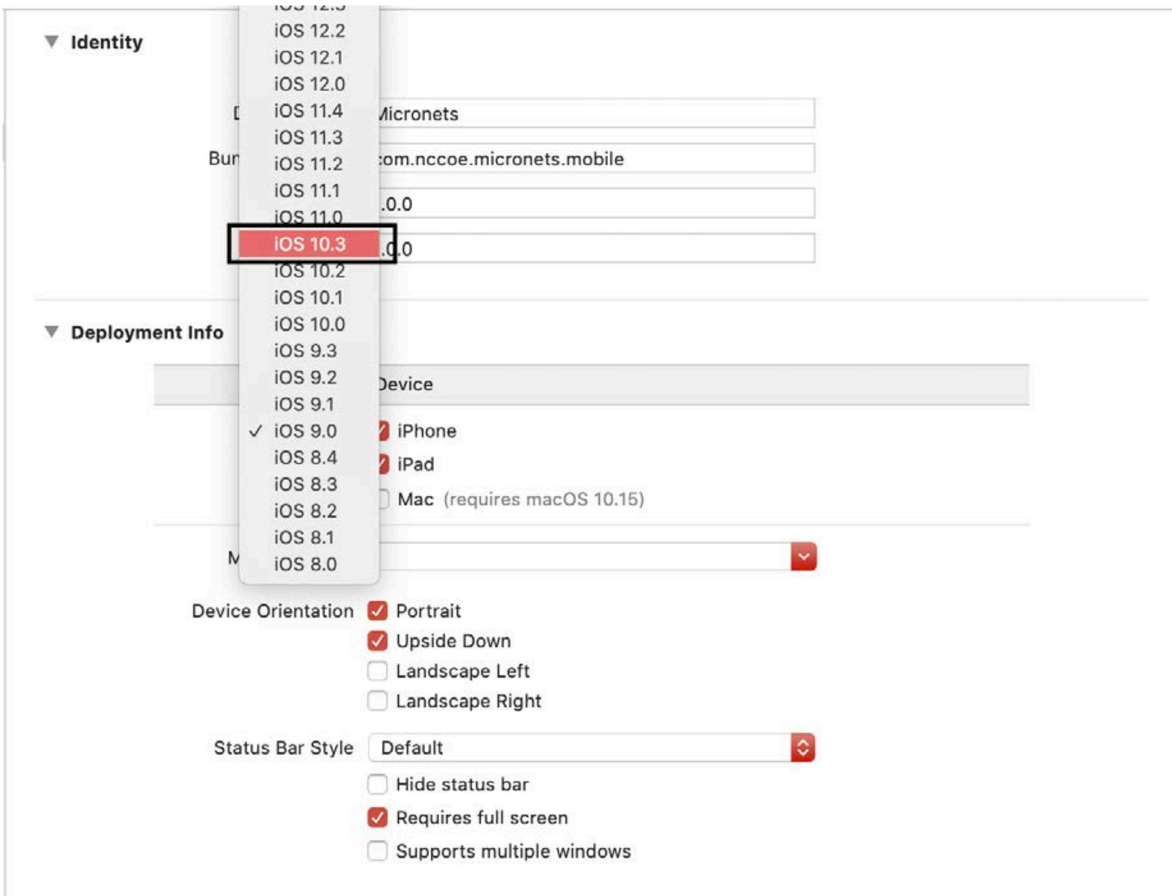


13. Select the **General** tab in the heading:

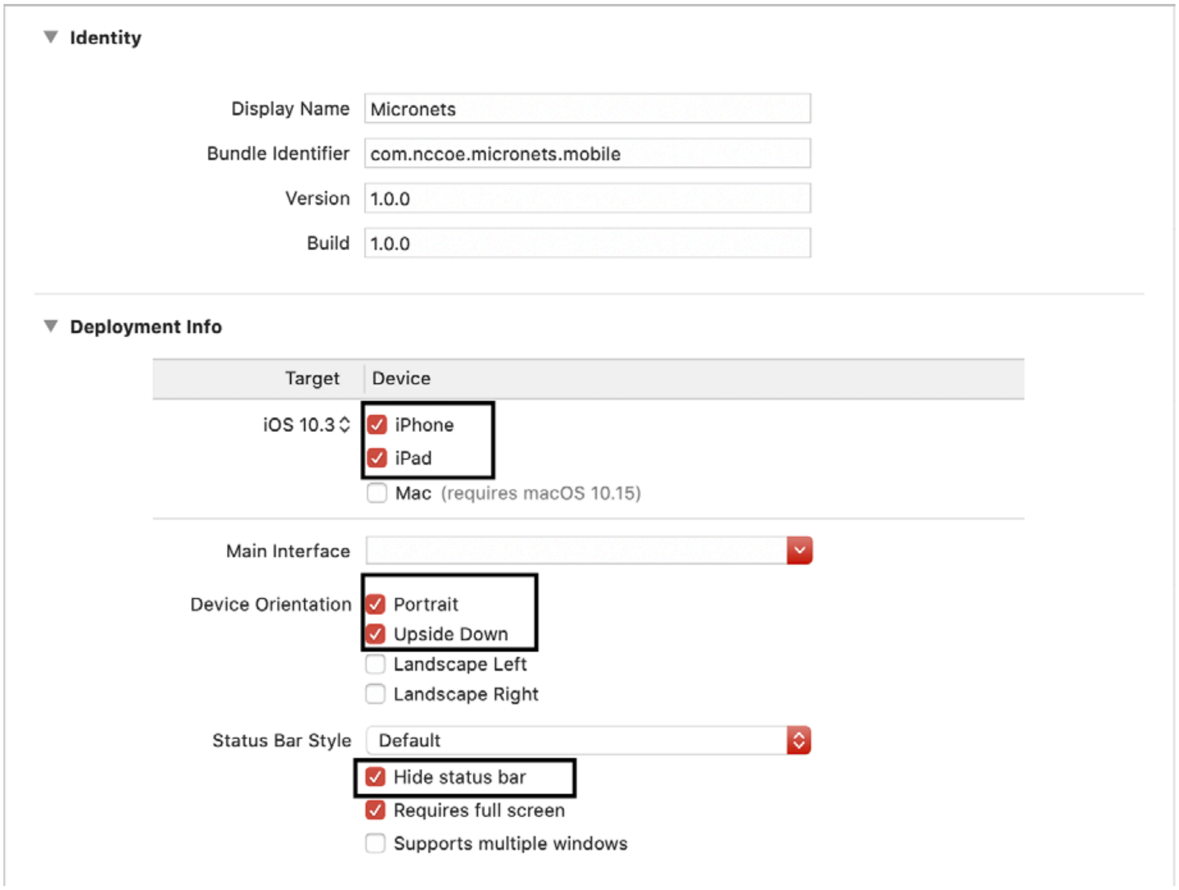
14. Under **Deployment Info,** make the following modifications:

    a. Select the deployment **Target** (suggested 10.3)


Identity
Display Name
Micronets
Bundle Identifier
com.nccoe.micronets.mobile
Version
1.0.0
Build
1.0.0
Deployment Info
Target
Device
iOS 9.0
iPhone
iPad
Mac (requires macOS 10.15)
Main Interface
Device Orientation
Portrait
Upside Down
Landscape Left
Landscape Right
Status Bar Style
Default
Hide status bar
Requires full screen
Supports multiple windows



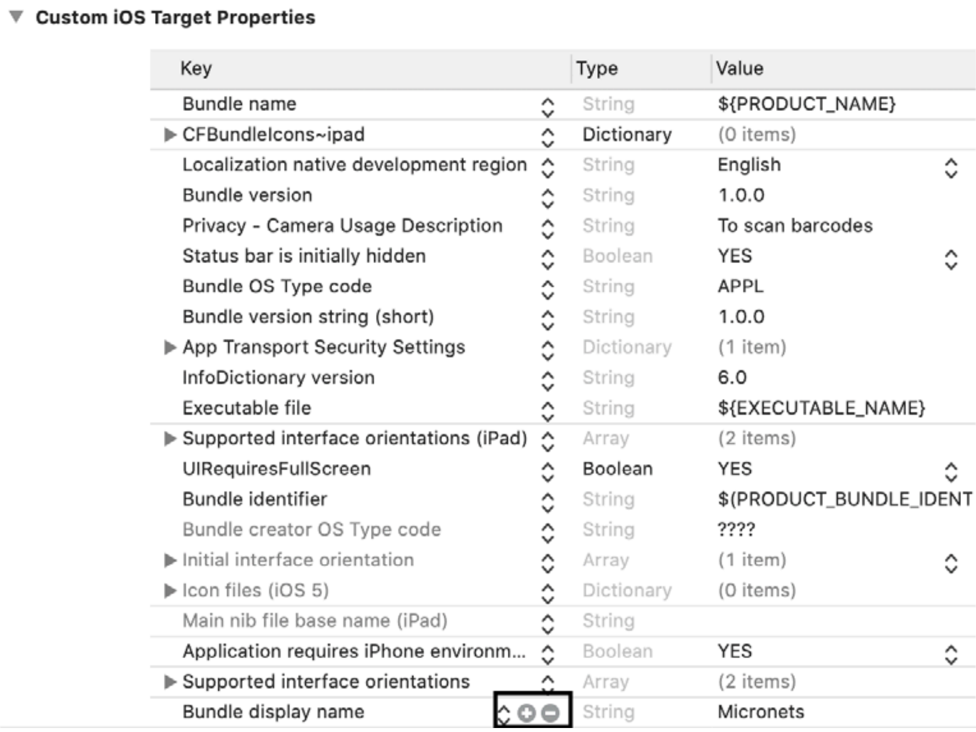
b. Select Device type **iPhone and iPad,** Device Orientation **Portrait and Upside Down,** Status Bar style **Hide status bar:**



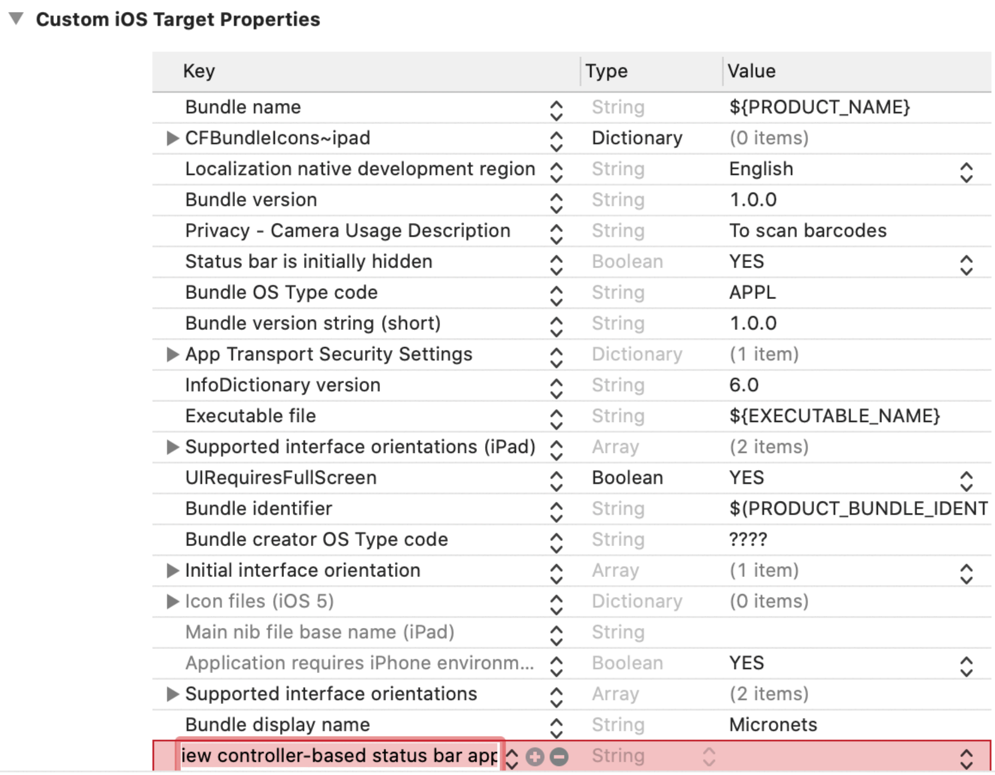
15. Select the **Info** tab, and make the following modifications:

   a. On last entry in **Custom iOS Target Properties,** hover over the down arrow.

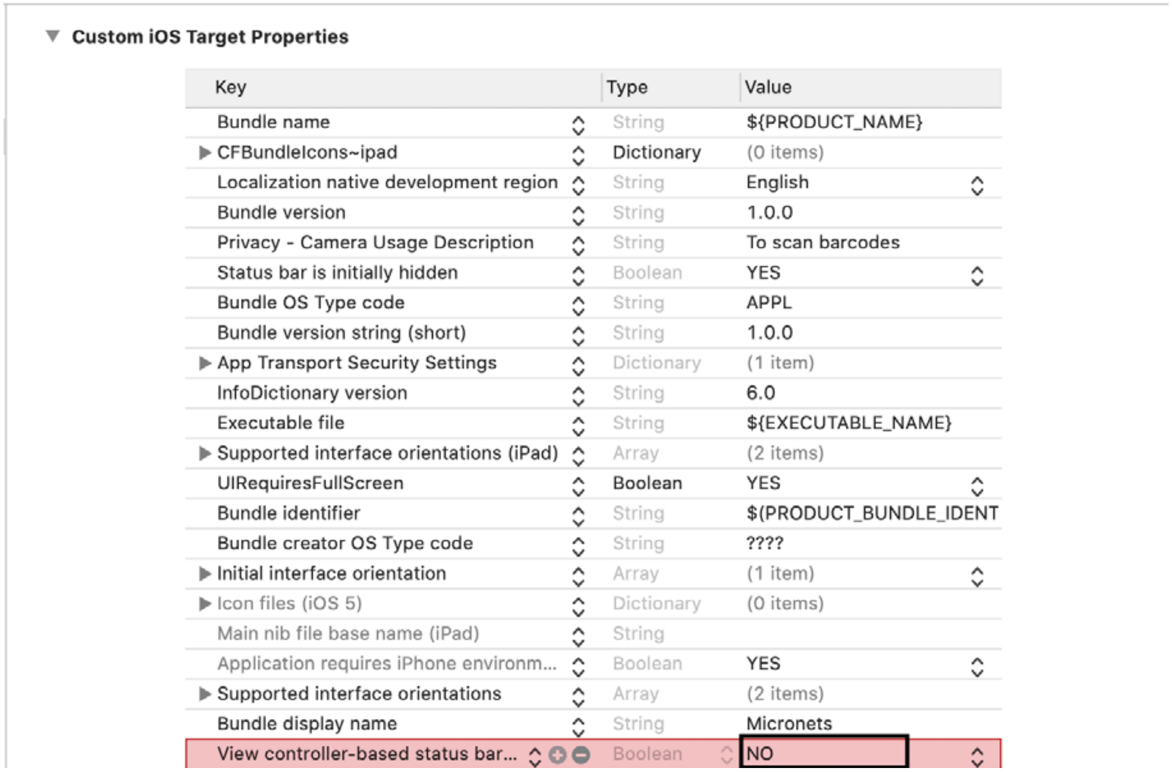
b. A plus sign appears. Click it to create a new property.



c. In the combo box drop-down, start typing **View controller**, and choose the auto-fill suggestion **View controller-based status bar appearance:**



d. Click **enter** to add this entry. Ensure this entry is set to **NO.**



▼ Custom iOS Target Properties

Key	Type	Value
Bundle name	String	${PRODUCT_NAME}
▶ CFBundleIcons~ipad	Dictionary	(0 items)
Localization native development region	String	English
Bundle version	String	1.0.0
Privacy - Camera Usage Description	String	To scan barcodes
Status bar is initially hidden	Boolean	YES
Bundle OS Type code	String	APPL
Bundle version string (short)	String	1.0.0
▶ App Transport Security Settings	Dictionary	(1 item)
InfoDictionary version	String	6.0
Executable file	String	${EXECUTABLE_NAME}
▶ Supported interface orientations (iPad)	Array	(2 items)
UIRequiresFullScreen	Boolean	YES
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENT
Bundle creator OS Type code	String	????
▶ Initial interface orientation	Array	(1 item)
▶ Icon files (iOS 5)	Dictionary	(0 items)
Main nib file base name (iPad)	String	
Application requires iPhone environm...	Boolean	YES
▶ Supported interface orientations	Array	(2 items)
Bundle display name	String	Micronets
View controller-based status bar...	Boolean	NO

16. Return to the terminal, and run the following command (ensure the iPhone is unlocked first):

```
cordova run ios --device --buildFlag='-UseModernBuildSystem=0'
```

Note: You may see an **UnhandledPromiseRejectionWarning** as seen below, but the application should still have been loaded onto your iPhone:

```
41D-9D0B-1E70E44AFCA0" "/Users/bmulugeta/Desktop/cablelabs/micronets-mobile/platforms/ios/bui
ld/device" /Developer "/Users/bmulugeta/Library/Developer/Xcode/iOS DeviceSupport/13.4.1 (17E
262) arm64e/Symbols/Developer"
(lldb)     command script import "/tmp/01B4BD9E-D31A-4A01-8033-04E6F2F78381/fruitstrap_000080
20_001E0D8126B9002E.py"
(lldb)     command script add -f fruitstrap_00008020_001E0D8126B9002E.connect_command connect
(lldb)     command script add -s asynchronous -f fruitstrap_00008020_001E0D8126B9002E.run_com
mand run
(lldb)     command script add -s asynchronous -f fruitstrap_00008020_001E0D8126B9002E.autoexi
t_command autoexit
(lldb)     command script add -s asynchronous -f fruitstrap_00008020_001E0D8126B9002E.safequi
t_command safequit
(lldb)     connect
(lldb)     run
error: process launch failed: The operation couldn't be completed. Unable to launch com.nccoe
.micronets.mobile because it has an invalid code signature, inadequate entitlements or its pr
ofile has not been explicitly trusted by the user.
(lldb)     safequit

Application has not been launched

(node:52444) UnhandledPromiseRejectionWarning: Error code 1 for command: ios-deploy with args
: --justlaunch,--no-wifi,-d,-b,/Users/bmulugeta/Desktop/cablelabs/micronets-mobile/platforms/
ios/build/device/Micronets.app
(node:52444) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error origin
ated either by throwing inside of an async function without a catch block, or by rejecting a
promise which was not handled with .catch(). To terminate the node process on unhandled promi
se rejection, use the CLI flag `--unhandled-rejections=strict` (see https://nodejs.org/api/cl
i.html#cli_unhandled_rejections_mode). (rejection id: 1)
(node:52444) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In th
e future, promise rejections that are not handled will terminate the Node.js process with a n
on-zero exit code.
```

4.1.12 MSO Portal Bootstrapping Interface to the Onboarding Manager

This section describes the CableLabs Micronets MSO portal, which, for this implementation, is a cloud-provided service. This implementation leverages the nccoe-build-3 branch of CableLabs Micronets MSO portal Git release. This service can be hosted by the implementer or another party. This documentation describes setting up your own MSO portal.

4.1.12.1 MSO Portal Overview

The MSO portal is the interface between the Micronets iPhone application and the Micronets Manager. It is responsible for passing onboarding requests and respective onboarding information to the Micronets Manager to complete the request.

4.1.12.2 Configuration Overview

The following subsections document the software and network configurations for the MSO portal. Please note that the MUD manager, Micronets Manager, Websocket Proxy, MUD registry, and MSO portal are all implemented on the same server, nccoe-server1.micronets.net. Many of these configurations are already covered in previous sections of this document but are repeated here for consistency.

4.1.12.2.1 Network Configuration

This server was hosted outside the lab environment on a Linode cloud-hosted Linux server. Its IP address was statically assigned.

4.1.12.2.2 Software Configuration

The following software is required to install, configure, and operate the MSO portal:

- docker (v18.06 or higher)
- docker-compose (v1.23.1 or higher)
- OpenSSL (1.0.2g or higher)
- NGINX and requisite certificates if https is to be supported

4.1.12.2.3 Hardware Configuration

The following hardware is required to install, configure, and operate the MSO portal:

- 4 GB of RAM
- 50 GB of free disk space

4.1.12.3 Setup

4.1.12.3.1 Install Dependencies

1. Install docker, docker-compose, openssl, and NGINX by entering the following command:

   ```
   sudo apt-get install docker docker-compose openssl nginx
   ```

4.1.12.3.2 Install and Configure MSO Portal

1. Install a newer version of docker-compose, if necessary. (Ubuntu 18.04 comes with an older version.)

   a. Check the current version by entering the following command:

      ```
      docker-compose --version
      ```

      The result should be similar to the following:

      ```
      micronets-dev@nccoe-server1:~/Projects/micronets$ docker-compose --version
      docker-compose version 1.24.1, build 4667896b
      ```

b. If the version is earlier than v1.23.1, run the following commands to install a new version in /usr/local/bin:

i. Download the docker compose utility:

```
curl -L -O
https://github.com/docker/compose/releases/download/1.24.1/docker-
compose-Linux-`uname -m’
```

ii. Install the docker compose utility into the appropriate directory:

```
sudo install -v -o root -m 755 docker-compose-Linux-`uname -m`
/usr/local/bin/docker-compose
```

The result should be similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 doc
ker-compose-Linux-`uname -m` /usr/local/bin/docker-compose
[sudo] password for micronets-dev:
removed '/usr/local/bin/docker-compose'
'docker-compose-Linux-x86_64' -> '/usr/local/bin/docker-compose'
```

2. Download and install the MSO portal management script by entering the following commands:

a. Download the MSO portal management script by executing the following command:

```
curl -O https://raw.githubusercontent.com/cablelabs/micronets-mso-
portal/nccoe-build-3/scripts/mso-portal
```

b. Download the *docker-compose.yml* file by executing the following command:

```
curl -O https://raw.githubusercontent.com/cablelabs/micronets-mso-
portal/nccoe-build-3/scripts/docker-compose.yml
```

c. Install the MSO portal management script to the appropriate directory by executing the following command:

```
sudo install -v -o root -m 755 -D -t /etc/micronets/mso-portal.d mso-
portal
```

The result should be similar to the following:

```
[micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 755 -D -t /etc/m]
icronets/mso-portal.d mso-portal
removed '/etc/micronets/mso-portal.d/mso-portal'
'mso-portal' -> '/etc/micronets/mso-portal.d/mso-portal'
```

d. Install the *docker-compose.yml* management script to the appropriate directory by executing the following command:

```
sudo install -v -o root -m 644 -D -t /etc/micronets/mso-portal.d docker-
compose.yml
```

The result should be similar to the following:

```
[micronets-dev@nccoe-server1:~/Projects/micronets$ sudo install -v -o root -m 644 -D ]
-t /etc/micronets/mso-portal.d docker-compose.yml
removed '/etc/micronets/mso-portal.d/docker-compose.yml'
'docker-compose.yml' -> '/etc/micronets/mso-portal.d/docker-compose.yml'
```

Note: The MSO portal management script contains default values that can be modified directly in your copy of the management script or overridden via command-line parameters.
Run /etc/micronets/mso-portal.d --help to see the options.

3. Download the MSO portal docker image by executing the following command (Note: If you cannot connect to the docker service, you can use sudo usermod -aG docker <username> to add the user account to the docker group):

```
/etc/micronets/mso-portal.d/mso-portal docker-pull
```

The result should be similar to the following:

```
[micronets-dev@nccoe-server1:~/Projects/micronets$ /etc/micronets/mso-portal.d/mso-po]
rtal docker-pull
Pulling docker image from community.cablelabs.com:4567/micronets-docker/micronets-ms
o-portal:nccoe-build-3
nccoe-build-3: Pulling from micronets-docker/micronets-mso-portal
48839397421a: Already exists
cbb6511d79bf: Already exists
587ebf5326af: Already exists
2bb87fce75b3: Already exists
df077bfbdbf4: Already exists
93207cfecda5: Already exists
f1a2689c2afd: Pull complete
27d9a703ba0a: Pull complete
5fabee586821: Pull complete
Digest: sha256:d7628a7815482718240a60c01390ad8dd1d795d87021246ebff3afbc93b66506
Status: Downloaded newer image for community.cablelabs.com:4567/micronets-docker/mic
ronets-mso-portal:nccoe-build-3
community.cablelabs.com:4567/micronets-docker/micronets-mso-portal:nccoe-build-3
```

4. Generate a shared secret for enabling communication between the Micronets Manager instances and the MSO portal:

```
sudo /etc/micronets/mso-portal.d/mso-portal create-mso-secret
```

The result should be similar to the following:

```
[micronets-dev@nccoe-server1:~/Projects/micronets$ sudo /etc/micronets/mso-portal.d/m]
so-portal create-mso-secret
'/tmp/tmp.M9Mtj9mGH6' -> '/etc/micronets/mso-portal.d/lib/mso-auth-secret'
Saved a 512-hex-digit shared secret to /etc/micronets/mso-portal.d/lib/mso-auth-secr
et
```

Note: This value will need to be copied to the Micronets Manager host server to allow Micronets Manager instances to access the MSO portal APIs.

5. Configure MSO portal URLs:

   a. Open the *mso-portal* file by entering the following command:

   ```
   sudo vim /etc/micronets/mso-portal.d/mso-portal
   ```

   b. Modify the parameters of the MSO portal management script to reflect the public end points of the MSO portal service. For example:

      i. The **DEF_MSO_API_BASE_URL** path variable can be set to:

      ```
      DEF_MSO_API_BASE_URL="https://nccoe-
      server1.micronets.net/micronets/mso-portal/"
      ```

      ii. The **DEF_WS_PROXY_BASE_URL** path variable can be set to:

```
DEF_WS_PROXY_BASE_URL="wss:// nccoe-server1.micronets.net:5050/micronets/v1/ws-proxy/gw"
```

```
#!/bin/bash

set -e

# dump_vars=1


# set -x

script_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"

DEF_IMAGE_LOCATION="community.cablelabs.com:4567/micronets-docker/micronets-mso-port
al"
DEF_IMAGE_TAG="nccoe-build-3"
DEF_DOCKER_PROJECT_NAME="micronets-mso-portal"
DEF_MSO_API_BASE_URL="https://nccoe-server1.micronets.net/micronets/mso-portal/"
DEF_WS_PROXY_BASE_URL="wss://nccoe-server1.micronets.net:5050/micronets/v1/ws-proxy/
gw"
DEF_BIND_PORT=3210
DEF_BIND_ADDRESS=127.0.0.1
DEF_DOCKER_COMPOSE_FILE="${script_dir}/docker-compose.yml"
DEF_MSO_AUTH_SECRET_FILE="/etc/micronets/mso-portal.d/lib/mso-auth-secret"

DOCKER_CMD="docker"
DOCKER_COMPOSE_CMD="docker-compose"
OPENSSL_CMD="openssl"

function bailout()
{
    local shortname="${0##*/}"
    local message="$1"
    echo "$shortname: error: ${message}" >&2
    exit 1;
}

function bailout_with_usage()
                                                                  1,11          Top
```

6. Start the MSO portal docker image by executing the following command:

   ```
   sudo /etc/micronets/mso-portal.d/mso-portal docker-run
   ```

   The result should be similar to the following:

```
micronets-dev@nccoe-server1:~/Projects/micronets$ sudo /etc/micronets/mso-portal.d/m
so-portal docker-run
[sudo] password for micronets-dev:
Starting container "micronets-mso-portal_api" from community.cablelabs.com:4567/micr
onets-docker/micronets-mso-portal:nccoe-build-3 (on 127.0.0.1:3210)
Performing docker-compose up operation...
Creating micronets-mso-portal_mongodb ... done
Creating micronets-mso-portal_api     ... done
```

7. Verify that the MSO portal started successfully by executing the following command:

   ```
   /etc/micronets/mso-portal.d/mso-portal docker-logs
   ```

   You should see output like the following at the end of the log:

   Feathers application started on "http://0.0.0.0:3210"

   Feathers webSocketBaseUrl "wss://<ServerURL>:5050/micronets/v1/ws-proxy/gw"

   Feathers publicApiBaseUrl "https://< ServerURL>/micronets/mso-portal/"

```
2020-05-05T19:10:17.844177983Z 2020-05-05 19:10:17 info [index.js]: Feathers applica
tion started on "http://0.0.0.0:3210"
2020-05-05T19:10:17.844472002Z 2020-05-05 19:10:17 info [index.js]: Feathers  webSoc
ketBaseUrl "wss://nccoe-server1.micronets.net:5050/micronets/v1/ws-proxy/gw"
2020-05-05T19:10:17.844657671Z 2020-05-05 19:10:17 info [index.js]: Feathers  public
ApiBaseUrl "https://nccoe-server1.micronets.net/micronets/mso-portal/"
2020-05-05T19:10:17.895522093Z (node:40) DeprecationWarning: collection.ensureIndex
is deprecated. Use createIndexes instead.
```

8. To securely expose the MSO API, configure your NGINX server block to allow the https proxy to redirect to localhost port 3210:

   a. Open the NGINX sites-available file for the server:

   ```
   sudo vim /etc/nginx/sites-available/nccoe-server1.micronets.net
   ```

   b. Add the following location to the server block:

   ```
   server {
          […]
       location /micronets/mso-portal/ {
          proxy_pass   http://127.0.0.1:3210/;
       }
   ```

```
        […]
}
```

```
server {
        listen 443 ssl;
        listen [::]:443 ssl;
        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;
        server_name nccoe-server1.micronets.net;

        location / {
                try_files $uri $uri/ =404;
        }

        location /micronets/mud-manager/ {
        proxy_pass      http://localhost:8888/;
        }

        location /registry/devices {
                proxy_pass      http://localhost:3082/vendors/;
        }
        location /mud/{
                proxy_pass      http://localhost:3082/registry/;
        }

        location /micronets/mso-portal/ {
                proxy_pass      http://127.0.0.1:3210/;
        }

        ssl_certificate /home/micronets-dev/Projects/micronets/cert/nccoe-server1_micronets_n
et.crt;
        ssl_certificate_key /home/micronets-dev/Projects/micronets/cert/nccoe-server1_microne
ts_net.key;

        include /etc/nginx/micronets-subscriber-forwards/*.conf;
}
~
```

4.2 Product Integration and Operation

This section details integration and operation of the Micronets components that were previously installed in the product installation section. Please ensure that the components from that section are installed as described before proceeding to the following sections.

4.2.1 Adding an MSO Subscriber

This section describes adding an MSO portal subscriber. This subscriber account will allow a valid connection and association among the Micronets mobile application, Micronets Gateway, and Micronets services.

4.2.1.1 Prerequisites

To successfully complete this section, complete the product installation section.

4.2.1.2 Instructions

1. Add a subscriber and associated user account and password to the MSO portal by entering the following command (Note: Be sure to use the server URL that reflects the location of your MSO portal):

```
curl -s -X POST https://nccoe-server1.micronets.net/micronets/mso-
portal/portal/v1/subscriber \
    -H "Content-Type: application/json" \
    -d '{
            "id" : "subscriber-001",
            "ssid" : "micronets-gw",
            "name" : "Subscriber 001",
            "gatewayId":"micronets-gw",
            "username":"micronets",
            "password":"micronets"
    }' \
| json_pp
```

You should see output similar to the following:

```
{
   "gatewayId" : "micronets-gw",
   "ssid" : "micronets-gw",
   "name" : "Subscriber 001",
   "id" : "subscriber-001",
   "registry" : ""
}
```

2. Start the Micronets Manager for the subscriber by executing the following command:

```
sudo /etc/micronets/micronets-manager.d/mm-container start subscriber-001
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ /etc/micronets/micronets-manager.d/mm-container start
 subscriber-001
Creating resources for subscriber subscriber-001...
Creating network "sub-subscriber-001_mm-priv-network" with the default driver
Creating volume "sub-subscriber-001_mongodb" with default driver
Creating sub-subscriber-001_mongodb_1 ... done
Creating sub-subscriber-001_api_1     ... done
Issuing nginx reload (running 'sudo nginx -s reload')
[sudo] password for micronets-dev:
```

3. Check the logs to confirm that the Micronets Manager for the new subscriber started successfully by executing the following command:

   ```
   /etc/micronets/micronets-manager.d/mm-container logs subscriber-001
   ```

   You should see output similar to the following:

```
2020-07-07T21:20:48.592313707Z 2020-07-07 21:20:48 ESC[34mdebugESC[39m [index.js]:
2020-07-07T21:20:48.592323377Z  Creating default micronet for result : {"_id":"5ee7bd72f7947d
002807d730","registry":"https://nccoe-server1.micronets.net/sub/subscriber-001/api","id":"sub
scriber-001","ssid":"micronets-gw","name":"Subscriber 001","gatewayId":"default-gw-subscriber
-001","createdAt":"2020-06-15T18:26:58.417Z","updatedAt":"2020-07-07T21:20:48.506Z","__v":0}
2020-07-07T21:20:48.592711656Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]:
2020-07-07T21:20:48.592722976Z  Hook Type: before  Path: mm/v1/subscriber  Method: create
2020-07-07T21:20:48.594055268Z 2020-07-07 21:20:48 ESC[34mdebugESC[39m [index.js]:
2020-07-07T21:20:48.594068138Z   Event Type "userCreate"                  Event data : {"type"
:"userCreate","id":"subscriber-001","name":"Subscriber 001","ssid":"micronets-gw","gatewayId"
:"default-gw-subscriber-001","micronets":[]}
2020-07-07T21:20:48.600624802Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]:
2020-07-07T21:20:48.600680273Z  Hook Type: after  Path: mm/v1/subscriber  Method: create
2020-07-07T21:20:48.600895833Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]: Hook.result
.data : undefined
2020-07-07T21:20:48.601240864Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]:
2020-07-07T21:20:48.601251324Z  Hook Type: before  Path: mm/v1/subscriber  Method: find
2020-07-07T21:20:48.604472856Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]:
2020-07-07T21:20:48.604517736Z  Hook Type: after  Path: mm/v1/subscriber  Method: find
2020-07-07T21:20:48.604743595Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]: Hook.result
.data : [{"_id":"5f04e7308a84ec1a8feab599","id":"subscriber-001","name":"Subscriber 001","ssi
d":"micronets-gw","gatewayId":"default-gw-subscriber-001","micronets":[],"createdAt":"2020-07
-07T21:20:48.597Z","updatedAt":"2020-07-07T21:20:48.597Z","__v":0}]
2020-07-07T21:20:48.604975416Z 2020-07-07 21:20:48 ESC[34mdebugESC[39m [index.js]:
2020-07-07T21:20:48.604985136Z  Default micronet for subscriber  : {"total":1,"limit":500,"sk
ip":0,"data":[{"_id":"5f04e7308a84ec1a8feab599","id":"subscriber-001","name":"Subscriber 001"
,"ssid":"micronets-gw","gatewayId":"default-gw-subscriber-001","micronets":[],"createdAt":"20
20-07-07T21:20:48.597Z","updatedAt":"2020-07-07T21:20:48.597Z","__v":0}]}
2020-07-07T21:20:48.605430046Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]:
2020-07-07T21:20:48.605439986Z  Hook Type: after  Path: mm/v1/micronets/registry  Method: cre
ate
2020-07-07T21:20:48.605631716Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]: Hook.result
.data : undefined
2020-07-07T21:20:48.605848037Z 2020-07-07 21:20:48 ESC[32minfoESC[39m [index.js]:
2020-07-07T21:20:48.605857217Z  Connecting to : "wss://nccoe-server1.micronets.net:5050/micro
nets/v1/ws-proxy/gw/subscriber-001" from mano configuration
2020-07-07T21:20:48.652161564Z Web socket connection on wss://nccoe-server1.micronets.net:505
0/micronets/v1/ws-proxy/gw/subscriber-001
```

4. Verify that the Micronets Manager for the subscriber has registered with the MSO portal by executing the following command:

```
curl -s https://my-server.org/micronets/mso-
portal/portal/v1/subscriber/subscriber-001 | json_pp
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl -s https://nccoe-server1.micronets.net/micronets
/mso-portal/portal/v1/subscriber/subscriber-001 | json_pp
{
   "name" : "Subscriber 001",
   "gatewayId" : "micronets-gw",
   "ssid" : "micronets-gw",
   "registry" : "",
   "id" : "subscriber-001"
}
```

4.2.2 Associating the Micronets Gateway with a Subscriber

This section describes associating an MSO portal subscriber with the Micronets Gateway. For additional instructions not detailed in this documentation, please follow the link to the CableLabs documentation: https://github.com/cablelabs/micronets-gw/releases/tag/1.0.62-u18.04 (for Micronets Gateway configuration) and https://github.com/cablelabs/micronets/blob/nccoe-build-3/docs/operation/gateway-4subscriber.md (for operations documentation).

4.2.2.1 Prerequisites

To successfully complete this section, complete the product installation section and complete Section 4.2.1. Ensure that all steps have been successfully completed before proceeding to the instructions.

4.2.2.2 Instructions

1. Create the /etc/network/interfaces file on the Micronets Gateway:

   a. Open a terminal on the Micronets Gateway. If this is the first installation of the Micronets Gateway, copy the sample interfaces file to your /etc/network/interfaces file by entering the following command:

   ```
   sudo cp /opt/micronets-gw/doc/interfaces.sample /etc/network/interfaces
   ```

   Modify the /etc/network/interfaces file:

   Retrieve the desired interface names on the gateway by running the following command in a terminal on the gateway:

   ```
   ifconfig
   ```

   Configure your wireless and wired interface by renaming the corresponding portion of the file to reference the respective interface name as seen in the config below:

   ```
   #
   # A wired interface managed by the Micronets gateway
   ```

```
#
allow-brmn001 enp1s0
iface enp1s0 manual
  ovs_type OVSPort
  ovs_bridge brmn001
  ovs_port_req 4
  ovs_port_initial_state blocked
#
# A wireless interface managed by the Micronets gateway
#
allow-brmn001 wlp2s0
iface wlp2s0inet manual
  ovs_type OVSPort
  ovs_bridge brmn001
  ovs_port_req 3
  ovs_port_initial_state blocked
```

Confirm that the bridge entry contains an ovs_ports line referring to the micronet interfaces **(enp1s0** and **wlp2s0)** as seen in the config below:

```
auto brmn001
allow-ovs brmn001
iface brmn001 inet manual
 ovs_type OVSBridge
  ...
 # the ovs_ports should list all wired and wireless interfaces under Micronets management
  ovs_ports diagout1 enp1s0 wlp2s0
  ...
```

Confirm that the entry in the interfaces file for the wired interface is set up correctly for the network to supply the uplink (the uplink interface is enp1s0) and get its address via DHCP so the configuration is similar to the following:

```
#
# The uplink port
```

```
#
auto eth enp1s0
iface eth0inet dhcp
```

Confirm that the bridge entry contains an **ovs_bridge_uplink_port** line referring to the uplink interface as seen in the config below:

```
auto brmn001
allow-ovs brmn001
iface brmn001 inet manual
  ovs_type OVSBridge
  ...
  # This is the port that's connected to the Internet
  ovs_bridge_uplink_port enp1s0
  ...
```

Reboot the gateway to apply the changes to the /etc/network/interfaces file by executing the following command:

```
sudo reboot
```

2. Create a gateway configuration file for the Micronets Gateway to register for the subscriber:
   a. Copy and save the MAC addresses and corresponding interface names output by executing the following command:

   ```
   ifconfig
   ```

   b. Navigate to the /etc/network/interfaces file on the gateway, and copy the subnets configurations, which will be used for the gateway configuration file in the following steps:

   ```
   sudo vim /etc/network/interfaces
   ```

   Copy and save the subnet and ranges associated with the interfaces identified in the previous step from this file (Note: These are at the bottom of the file):

```
# Note: The entries below are sample definitions to be added to the
#       system-provided /etc/network/interfaces file. The definitions
#       include custom keywords to setup the OVS bridge and network
#       configuration.
auto enp0s31f6
iface enp0s31f6 inet static
  address 192.168.1.30/24
  gateway 192.168.1.1
  dns-nameservers 8.8.8.8 8.8.4.4
#
# create an OpenVswitch bridge for Micronets management
#
auto brmn001
allow-ovs brmn001
iface brmn001 inet manual
  ovs_type OVSBridge
  # This is the port that's connected to the Internet
  ovs_bridge_uplink_port enp0s31f6
  # the ovs_ports should list all wired and wireless interfaces under Micronets management
  ovs_ports diagout1 wlp2s0 enp1s0
  ovs_protocols OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13

# Assign IP addresses to the bridge that may be configured as Micronets
# Note: This will be replaced with dynamic route table entries in the future

iface brmn001 inet static
  address 10.135.1.1/24

iface brmn001 inet static
  address 10.135.2.1/24

iface brmn001 inet static
  address 10.135.3.1/24

iface brmn001 inet static
  address 10.135.4.1/24

iface brmn001 inet static
  address 10.135.5.1/24

#
# The uplink port
#

# An uplink may already be defined in the system-provided interfaces file.
# This interface should have a default gateway and must NOT be listed in the
# ovs_ports line of the bridge definition.

#
# A wireless interface managed by the Micronets gateway
```

2914

```
#
allow-brmn001 wlp2s0
iface wlp2s0 inet manual
  ovs_type OVSPort
  # The ovs_bridge must match the bridge definition (above)
  ovs_bridge brmn001
  # The port number needs to be unique for the bridge
  ovs_port_req 3
  # Indicates that the port is blocked at startup (until enabled via command)
  ovs_port_initial_state blocked

#
# A wired interface managed by the Micronets gateway
#
allow-brmn001 enp1s0
iface enp1s0 inet manual
  ovs_type OVSPort
  ovs_bridge brmn001
  ovs_port_req 4
  ovs_port_initial_state blocked

#
# Create a local interface/tap for diagnostic output
#
# Note: The OVS rules written by the Micronets Manager will output
#       packets to port 42 to drop them from flows. This interface
#       can be used to capture dropped packets, for diagnostics.
allow-brmn001 diagout1
iface diagout1 inet manual
  ovs_type OVSIntPort
  ovs_bridge brmn001
  ovs_port_req 42
  ovs_port_initial_state blocked
```

c. Create the gateway config file by entering the following command:

```
sudo vim gateway-config-001.json
```

d. Modify the following configuration to include your gateway's MAC address and subnets as seen below and copy them into the *gateway-config-001.json* file:

Be sure to modify the **ipv4SubnetRanges** definition to match the bridge subnet range—e.g., the file above defines five different subnets ranging from 10.135.1.1/24–10.135.5.1/24, so we set **octetC** to have a minimum of 1 and a maximum of 5 and **octetD** to have a minimum of 2 and a maximum of 254 as seen in the config below:

```
{
    "version": "1.0",
    "gatewayId": "micronets-gw",
```

```
                    "gatewayModel": "proto-gateway",
                    "gatewayVersion": {"major":1, "minor":0, "micro":0},
                    "configRevision": 1,
                    "vlanRanges": [
                        {"min":1000, "max":4095}
                    ],
                    "micronetInterfaces": [
                        {
                            "medium": "wifi",
                            "name": "wlp2s0",
                            "macAddress": "20:16:d8:2b:4b:41",
                            "ssid": "micronets-gw",
                            "dpp": {
                                "supportedAkms": ["psk"]
                            },
                            "ipv4SubnetRanges": [
                                {
                                    "id": "range001",
                                    "subnetRange": {"octetA": 10,
                                                    "octetB": 135,
                                                    "octetC": {"min":1, "max":5}
                                    },
                                    "subnetGateway": {"octetD": 1},
                                    "deviceRange": {"octetD": {"min":2, "max":254}}
                                }
                            ]
                        },
                        {
                            "medium": "ethernet",
                            "name": "enp1s0",
```

```
                    "macAddress": "80:ee:73:dc:64:1d",
                    "ipv4Subnets": [
                        {
                            "id": "range001",
                            "subnetRange": {"octetA": 10,
                                            "octetB": 135,
                                            "octetC": 250
                            },
                            "subnetGateway": {"octetD": 1},
                            "deviceRange": {"octetD": {"min":2, "max":254}}
                        }
                    ]
                }
            ]
        }
```

Register a gateway configuration for a subscriber with the subscriber's Micronets Manager instance by entering the following command (with the subscriber being subscriber-001 in this case):

```
curl -s -X POST https://nccoe-server1.micronets.net/sub/subscriber-
001/api/mm/v1/micronets/odl \
-H "Content-Type: application/json" -d @./gateway-config-001.json | json_pp
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl -s -X POST https://nccoe-server1.micronets.net/sub/subscr
iber-001/api/mm/v1/micronets/odl \
> -H "Content-Type: application/json" -d @./gateway-config-001.json | json_pp
{
   "vlanRanges" : [
      {
         "min" : "1000",
         "max" : "4095"
      }
   ],
   "gatewayId" : "micronets-gw",
   "__v" : 0,
   "gatewayModel" : "proto-gateway",
   "gatewayVersion" : {
      "minor" : "0",
      "major" : "1",
      "micro" : "0"
   },
   "configRevision" : "1",
   "createdAt" : "2020-07-08T16:03:08.376Z",
   "updatedAt" : "2020-07-08T16:03:08.376Z",
   "_id" : "5f05ee3c8a84ec9329eab59a",
   "version" : "1.0",
   "micronetInterfaces" : [
      {
         "ssid" : "micronets-gw",
         "macAddress" : "20:16:d8:2b:4b:41",
         "medium" : "wifi",
         "ipv4SubnetRanges" : [
            {
               "deviceRange" : {
                  "octetD" : {
                     "max" : "254",
                     "min" : "2"
                  }
               },
               "subnetGateway" : {
                  "octetD" : "1"
               },
               "subnetRange" : {
                  "octetB" : "135",
                  "octetC" : {
                     "max" : "5",
                     "min" : "1"
                  },
                  "octetA" : "10"
               },
               "id" : "range001"
            }
         ],
         "ipv4Subnets" : [],
         "name" : "wlp2s0",
         "dpp" : {
```

```
            "supportedAkms" : [
               "psk"
            ]
         }
      },
      {
         "medium" : "ethernet",
         "macAddress" : "80:ee:73:dc:64:1d",
         "name" : "enp1s0",
         "ipv4SubnetRanges" : [],
         "ipv4Subnets" : [
            {
               "subnetRange" : {
                  "octetC" : "250",
                  "octetA" : "10",
                  "octetB" : "135"
               },
               "subnetGateway" : {
                  "octetD" : "1"
               },
               "deviceRange" : {
                  "octetD" : {
                     "max" : "254",
                     "min" : "2"
                  }
               }
            }
         ]
      }
   ]
}
```

Confirm that the gateway ID is updated in the MSO portal by executing the following command:

```
curl -s https://nccoe-server1.micronets.net/micronets/mso-
portal/portal/v1/subscriber/subscriber-001 | json_pp
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl -s https://nccoe-server1.micronets.net/micronets/mso-port
al/portal/v1/subscriber/subscriber-001 | json_pp
{
   "id" : "subscriber-001",
   "ssid" : "micronets-gw",
   "name" : "Subscriber 001",
   "registry" : "https://nccoe-server1.micronets.net/sub/subscriber-001/api",
   "gatewayId" : "micronets-gw"
}
```

Configure the Micronets Gateway with the Websocket Proxy keys provisioned for the gateway:

- Copy the client cert and key as well as the Websocket root certificate, created in the product installation section, from the cloud server into the gateway by executing the following commands from the gateway:

i. Copy the *micronets-gw-service.pkeycert.pem* to the gateway:

```
scp micronets-dev@nccoe-server1.micronets.net:Projects/mi-
cronets/micronets-gw-service.pkeycert.pem .
```

You should see the following output:

```
micronets-gw-service.pkeycert.pem                     100%  933    15.4KB/s   00:00
```

ii. Copy the *micronets-ws-root.cert.pem* to the gateway:

```
scp micronets-dev@nccoe-server1.micronets.net:Projects/mi-
cronets/micronets-ws-root.cert.pem .
```

You should see the following output:

```
micronets-ws-root.cert.pem                            100%  656    10.8KB/s   00:00
```

b. Copy them into the gateway service library to be loaded when the gateway is restarted:

```
sudo cp -v micronets-gw-service.pkeycert.pem micronets-ws-root.cert.pem
/opt/micronets-gw/lib/
```

Change the Websocket lookup URL to use the MSO portal service on your server by completing the following commands:

a. Open the Micronets Gateway config file by executing the following command:

```
sudo vim /opt/micronets-gw/config.py
```

b. Modify the **WEBSOCKET_LOOKUP_URL** and **GATEWAY_ID** to match the MSO portal Websocket lookup end point created in the product installation section and the Micronets Gateway ID:

```
WEBSOCKET_LOOKUP_URL = 'https://nccoe-
server1.micronets.net/micronets/mso-
portal/portal/v1/socket?gatewayId={gateway_id}'

GATEWAY_ID = 'micronets-gw'
```

```
import os, sys, pathlib, logging

app_dir = os.path.abspath (os.path.dirname (__file__))

class BaseConfig:
    GATEWAY_ID = 'micronets-gw'
    LOGGING_LEVEL = logging.DEBUG
    SECRET_KEY = os.environ.get ('SECRET_KEY') or 'A SECRET KEY'
    LISTEN_HOST = "0.0.0.0"
    LISTEN_PORT = 5000
    MIN_DHCP_UPDATE_INTERVAL_S = 2
    DEFAULT_LEASE_PERIOD = '2m'
    SERVER_BASE_DIR = pathlib.Path (__file__).parent
    SERVER_BIN_DIR = SERVER_BASE_DIR.joinpath ("bin")
    WEBSOCKET_CONNECTION_ENABLED = False
    WEBSOCKET_LOOKUP_URL = 'https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v
1/socket?gatewayId={gateway_id}'
    WEBSOCKET_TLS_CERTKEY_FILE = pathlib.Path (__file__).parent.joinpath ('lib/micronets-gw-s
ervice.pkeycert.pem')
    WEBSOCKET_TLS_CA_CERT_FILE = pathlib.Path (__file__).parent.joinpath ('lib/micronets-ws-r
oot.cert.pem')
    FLOW_ADAPTER_NETWORK_INTERFACES_PATH = "/etc/network/interfaces"
    # For this command, the first parameter will be the bridge name and the second the flow f
ilename
    FLOW_ADAPTER_ENABLED = False
    DPP_HANDLER_ENABLED = False
    DPP_CONFIG_KEY_FILE = pathlib.Path (__file__).parent.joinpath ("lib/hostapd-dpp-configura
tor.key")
    DPP_AP_CONNECTOR_FILE = pathlib.Path (__file__).parent.joinpath ("lib/hostapd-dpp-ap-conn
ector.json")
    HOSTAPD_ADAPTER_ENABLED = False
    SIMULATE_ONBOARD_RESPONSE_EVENTS = False

class BaseGatewayConfig:
    LOGFILE_PATH = pathlib.Path (__file__).parent.joinpath ("micronets-gw.log")
    FLOW_ADAPTER_APPLY_FLOWS_COMMAND = '/usr/bin/ovs-ofctl add-flows {ovs_bridge} {flow_file}
'
    HOSTAPD_PSK_FILE_PATH = '/opt/micronets-hostapd/lib/hostapd.wpa_psk'
    HOSTAPD_CLI_PATH = '/opt/micronets-hostapd/bin/hostapd_cli'
    # Set this iff you want to disable websocket URL lookup using MSO Portal (MSO_PORTAL_WEBS
OCKET_LOOKUP_ENDPOINT)
    #    WEBSOCKET_URL = "wss://ws-proxy-api.micronets.in:5050/micronets/v1/ws-proxy/gw-test/
{gateway_id}"

#
# Mock Adapter Configurations
#

class BaseMockConfig (BaseConfig):
    DHCP_ADAPTER = "Mock"
```

Restart the Micronets Gateway Service by executing the following command:

```
sudo systemctl restart micronets-gw.service
```

Check the Micronets Gateway Service log (**/opt/micronets-gw/micronets-gw.log**) to verify that the gateway's Websocket registration status was successful:

```
cat /opt/micronets-gw/micronets-gw.log
```

You should see output similar to the following:

```
2020-07-06 10:41:17,838 hostapd_adapter: INFO HostapdAdapter.update: PSK reload successful
2020-07-06 10:41:34,697 micronets-gw-service: INFO WSConnector: get_websocket_url_for_gateway
(micronets-gw)...
2020-07-06 10:41:34,698 micronets-gw-service: INFO WSConnector: get_websocket_url_for_gateway
(micronets-gw): Retrieving https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1
/socket?gatewayId=micronets-gw
2020-07-06 10:41:34,997 micronets-gw-service: INFO WSConnector: get_websocket_url_for_gateway
(micronets-gw): Received response: {'socketUrl': 'wss://nccoe-server1.micronets.net:5050/micr
onets/v1/ws-proxy/gw/subscriber-001', 'subscriberId': 'subscriber-001', 'gatewayId': 'microne
ts-gw'}
2020-07-06 10:41:34,997 micronets-gw-service: INFO WSConnector: get_websocket_url_for_gateway
(micronets-gw): Received URL: wss://nccoe-server1.micronets.net:5050/micronets/v1/ws-proxy/gw
/subscriber-001
2020-07-06 10:41:34,997 micronets-gw-service: INFO WSConnector: init_connect opening wss://nc
coe-server1.micronets.net:5050/micronets/v1/ws-proxy/gw/subscriber-001...
2020-07-06 10:41:35,038 websockets.protocol: DEBUG client - state = CONNECTING
2020-07-06 10:41:35,138 websockets.protocol: DEBUG client - event = connection_made(<asyncio.
sslproto._SSLProtocolTransport object at 0x7fc20c53c2b0>)
2020-07-06 10:41:35,188 websockets.protocol: DEBUG client - state = OPEN
2020-07-06 10:41:35,189 micronets-gw-service: INFO WSConnector: init_connect opened wss://ncc
oe-server1.micronets.net:5050/micronets/v1/ws-proxy/gw/subscriber-001.
2020-07-06 10:41:35,189 micronets-gw-service: INFO WSConnector Sending HELLO message...
2020-07-06 10:41:35,189 micronets-gw-service: DEBUG ws_connector: > sending event message: {'
messageType': 'CONN:HELLO', 'requiresResponse': False, 'peerClass': 'micronets-gateway-servic
e', 'peerId': 'gw service 140471400513432', 'messageId': 0}
2020-07-06 10:41:35,189 websockets.protocol: DEBUG client > Frame(fin=True, opcode=1, data=b'
{"message": {"messageId": 0, "messageType": "CONN:HELLO", "peerClass": "micronets-gateway-ser
vice", "peerId": "gw service 140471400513432", "requiresResponse": false}}', rsv1=False, rsv2
=False, rsv3=False)
2020-07-06 10:41:35,189 micronets-gw-service: INFO WSConnector: Waiting for HELLO messages...
2020-07-06 10:41:35,191 websockets.protocol: DEBUG client < Frame(fin=True, opcode=9, data=b'
\xf5\xa5\x18\xce', rsv1=False, rsv2=False, rsv3=False)
2020-07-06 10:41:35,191 websockets.protocol: DEBUG client - received ping, sending pong: f5a5
18ce
2020-07-06 10:41:35,191 websockets.protocol: DEBUG client > Frame(fin=True, opcode=10, data=b
'\xf5\xa5\x18\xce', rsv1=False, rsv2=False, rsv3=False)
2020-07-06 10:41:35,245 websockets.protocol: DEBUG client < Frame(fin=True, opcode=1, data=b'
{"message": {"messageId": 0, "messageType": "CONN:HELLO", "peerClass": "micronets-ws-test-cli
ent", "peerId": "12345678", "requiresResponse": false}}', rsv1=False, rsv2=False, rsv3=False)
2020-07-06 10:41:35,245 micronets-gw-service: DEBUG ws_connector: process_hello_messages: Rec
eived message: {'message': {'messageId': 0, 'messageType': 'CONN:HELLO', 'peerClass': 'micron
ets-ws-test-client', 'peerId': '12345678', 'requiresResponse': False}}
2020-07-06 10:41:35,245 micronets-gw-service: DEBUG ws_connector: process_hello_messages: Rec
eived HELLO message
2020-07-06 10:41:35,245 micronets-gw-service: INFO WSConnector: HELLO handshake complete.
2020-07-06 10:41:35,245 micronets-gw-service: DEBUG WSConnector: sender: starting...
2020-07-06 10:41:35,245 micronets-gw-service: DEBUG WSConnector: sender: exiting.
2020-07-06 10:41:35,245 micronets-gw-service: DEBUG WSConnector: receiver: starting...
```

Confirm the establishment of the gateway-manager control connection by examining the Websocket Proxy connection reports in the Websocket Proxy log:

```
/etc/micronets/micronets-ws-proxy docker-logs | less
```

Look for the following in the log (with the **MEETUP ID** matching the subscriber name in question):

```
2020-07-07T21:20:48.645800508Z ---------------------------------------------------------------
-------------------------
2020-07-07T21:20:48.645803778Z WEBSOCKET MEETUP TABLE REPORT FOR 0.0.0.0:5050//micronets/v1/w
s-proxy/
2020-07-07T21:20:48.645824049Z
2020-07-07T21:20:48.645849999Z   MEETUP ID: gw/subscriber-001
2020-07-07T21:20:48.645854809Z     Client 1: Client 139799006767424 (peer: gw service 1404714
00513432) @ ('173.73.49.216', 41150))
2020-07-07T21:20:48.645857689Z     Client 2: Client 139799006768712 (peer: 12345678) @ ('172.
17.0.1', 37962))
2020-07-07T21:20:48.645860509Z ---------------------------------------------------------------
--------------------------
```

This indicates that the Micronets Gateway Service and the Micronets Manager for the subscriber connected and can exchange provisioning commands and event indications.

4.2.3 Integrating Micronets Proto-Pi Device

This section describes associating an MSO portal subscriber with the Micronets Gateway. For additional instructions not detailed in this documentation, please follow the link to the CableLabs documentation: https://github.com/cablelabs/micronets-pi3/blob/nccoe-build-3/README.md#Operation.

4.2.3.1 Prerequisites

To successfully complete this section, be sure to have completed the product installation section associated with the Micronets Proto-Pi device. Ensure all steps have been successfully completed before proceeding to the instructions.

4.2.3.2 Instructions

1. Connect to the Raspberry Pi via SSH by entering the following command:

   ```
   ssh pi@192.168.30.191
   ```

   You will be prompted to enter the device password the password will remain the same.

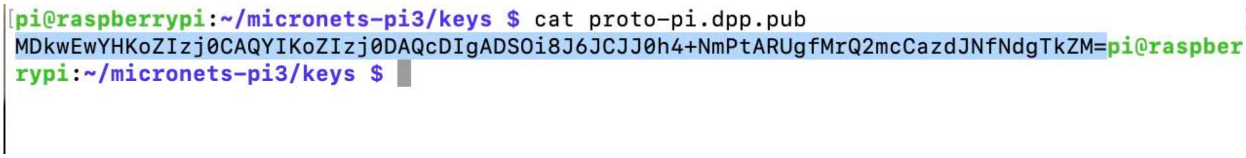
2. Change to the keys directory by entering the following command:

   ```
   cd micronets-pi3/keys/
   ```

3. Output the content of the **proto-pi.dpp.pub** file to copy the public key for this device (Note: You will need to store this device key for registering the device with the MUD registry if doing so manually):

```
cat proto-pi.dpp.pub
```

Highlight and copy the key that was output by the previous command:



```
pi@raspberrypi:~/micronets-pi3/keys $ cat proto-pi.dpp.pub
MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=pi@raspber
rypi:~/micronets-pi3/keys $
```

4. Modify the **config.json** file to include the key that was copied in the previous step, and modify the parameters of the file to match your setup:

```
sudo vim ~/micronets-pi3/config/config.json
```

The original file before editing should be similar to the following screenshot:

```
{
    "channel": 1,
    "channelClass": 81,
    "comcast": false,
    "demo": true,
    "deviceModelUID": "AgoNDQcDDgg",
    "deviceProfile": "device-0",
    "disableMUD": false,
    "dppName": "myDevice",
    "dppProxy": {
        "msoPortalUrl": "https://mso-portal-api.micronets.in",
        "password": "grandma",
        "username": "grandma"
    },
    "messageTimeoutSeconds": 45,
    "mode": "dpp",
    "onboardAnimationSeconds": 5,
    "qrcodeCountdown": 30,
    "registrationServer": "https://alpineseniorcare.com/micronets",
    "splashAnimationSeconds": 10,
    "vendorCode": "DAWG"
}
~
~
```

If doing manual device registration edit the file to reflect the correct **DeviceModelUID** (should be the same name as the MUD file associated with this device), **dppMUDUrl**, **msoPortalUrl**, **registrationServer**, **vendorCode** as seen below:

```
{
    "channel": 1,
    "channelClass": 81,
```

```
        "comcast": false,
        "demo": true,
        "deviceModelUID": "nist-model-fe_northsouth.json",
        "deviceProfile": "device-0",
        "disableMUD": false,
        "dppMUDUrl": "https://nccoe-server1.microents.net/mud/v1/mud-
url/TEST/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACxjMF8Ucp6d3gRBImv78eGEMwB5igS2Kt5b
nXI7VeBrc=",
        "dppName": "myDevice",
        "dppProxy": {
            "msoPortalUrl": "https://nccoe-server1.micronets.net/micronets/mso-por-
tal/",
            "password": "grandma",
            "username": "grandma"
        },
        "messageTimeoutSeconds": 45,
        "mode": "dpp",
        "onboardAnimationSeconds": 5,
        "qrcodeCountdown": 30,
        "registrationServer": "https://nccoe-server1.micronets.net/registry/de-
vices",
        "splashAnimationSeconds": 10,
        "vendorCode": "TEST"
    }
```

If enabling self-registry, follow the steps described in the following documentation: https://github.com/cablelabs/micronets-pi3/blob/nccoe-build-3/README.md#dpp-mode-mud-registry .

5. Reboot the device for the new config file to take effect:

```
sudo reboot
```

4.2.4 Updating MUD Registry

This section describes the HTTP API operations for interacting with the MUD registry. The instructions detail how to register a MUD-capable device and its MUD URL with a vendor. For additional API operations not documented here, follow the link to the CableLabs MUD registry operation documentation: https://github.com/cablelabs/micronets-mud-registry/blob/nccoe-build-3/README.md#Operation.

4.2.4.1 Prerequisites

To successfully complete this section, be sure to have completed the product installation section.

4.2.4.2 Instructions

1. Retrieve the device registry URL for a vendor by entering the following curl command:

/mud/v1/device-registry/:vendor-code

```
curl -L https://nccoe-server1.micronets.net/mud/v1/device-registry/TEST
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl -L https://nccoe-server1.micronets.net/mud/v1/dev
ice-registry/TEST
https://nccoe-server1.micronets.net/registry/devices/register-devicemicronets-dev@ncc
```

2. Register a device with a vendor's registry. This requires the device model UID and the public key, which can be modified and retrieved through the Micronets Proto-Pi:

   /registry/devices/register-device/:device-model-UID64/:public-key

   ```
   curl -X POST https://nccoe-server1.micronets.net/registry/devices/register-device/nist-model-fe_northsouth.json/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=
   ```

   You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl -X POST https://nccoe-server1.micronets.net/registry/devi
ces/register-device/nist-model-fe_northsouth.json/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6J
CJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=
Device registered (update): {
  "model": "nist-model-fe_northsouth.json",
  "pubkey": "MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=
",
  "timestamp": "2020-07-08 20:04:42 UTC",
  "_id": "q3Sn6E3S3NjGnf3Q"
```

Retrieve the MUD registry URL for a vendor:

/mud/v1/mud-registry/:vendor-code

```
curl https://nccoe-server1.micronets.net/mud/v1/mud-registry/TEST
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl https://nccoe-server1.micronets.net/mud/v1/mud-re
gistry/TEST
https://nccoe-server1.micronets.net/registry/devices/mud-registrymicronets-dev@nccoe-
server1:~$
```

Lookup a MUD URL from the vendor MUD registry:

/registry/devices/mud-registry/:public-key

```
curl https://nccoe-server1.micronets.net/registry/devices/mud-registry/
MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM
=
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl https://nccoe-server1.micronets.net/registry/devices/mud-
registry/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=
https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_northsouth.jsonmicronets-dev@
nccoe-server1:~$
```

Delete a device from the MUD registry (Note: If you do this step, the device will no longer be associated with a MUD file. Therefore, you should execute this command only if you do not intend to onboard the device with MUD capabilities):

/registry/devices/remove-device/:public-key

```
curl -L -X POST https://nccoe-server1.micronets.net/registry/devices/remove-de-
vice/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNd
gTkZM=
```

You should see output similar to the following:

```
micronets-dev@nccoe-server1:~$ curl -L -X POST https://nccoe-server1.micronets.net/registry/d
evices/remove-device/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNf
NdgTkZM=
Device removed: MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTk
ZM=micronets-dev@nccoe-server1:~$
```

4.2.5 Integrating the Micronets iPhone App with MSO Portal

This section describes integrating the Micronets iPhone application with the MSO portal. For additional instructions not detailed in this documentation, please follow the link to the CableLabs documentation: https://github.com/cablelabs/micronets-mobile/blob/nccoe-build-3/README.md#Operation.

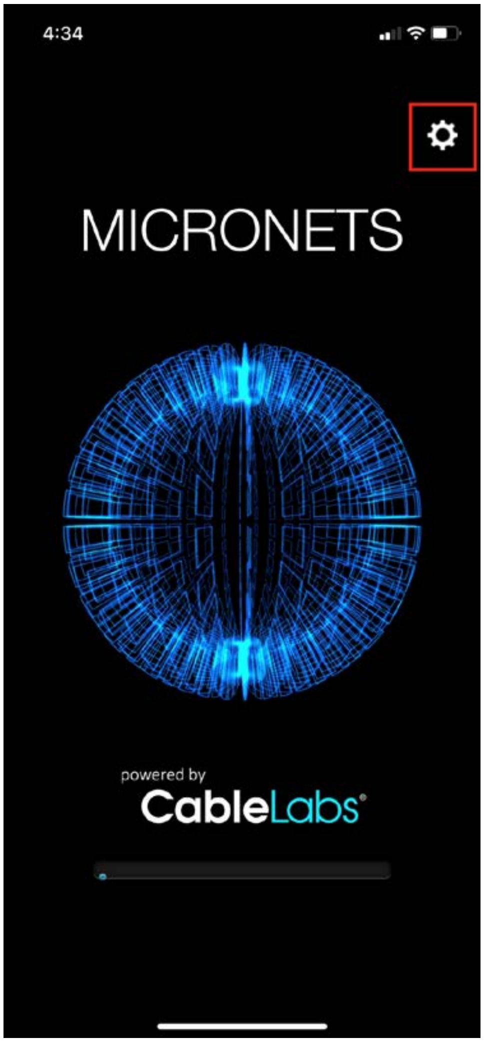
4.2.5.1 Prerequisites

A valid network connection on the iPhone is required as well as the completion of the product installation section related to the Micronets iPhone application.

4.2.5.2 Instructions

1. Open the Micronets mobile application:

2. From the splash screen click the gear button in the upper right corner to open the settings page:

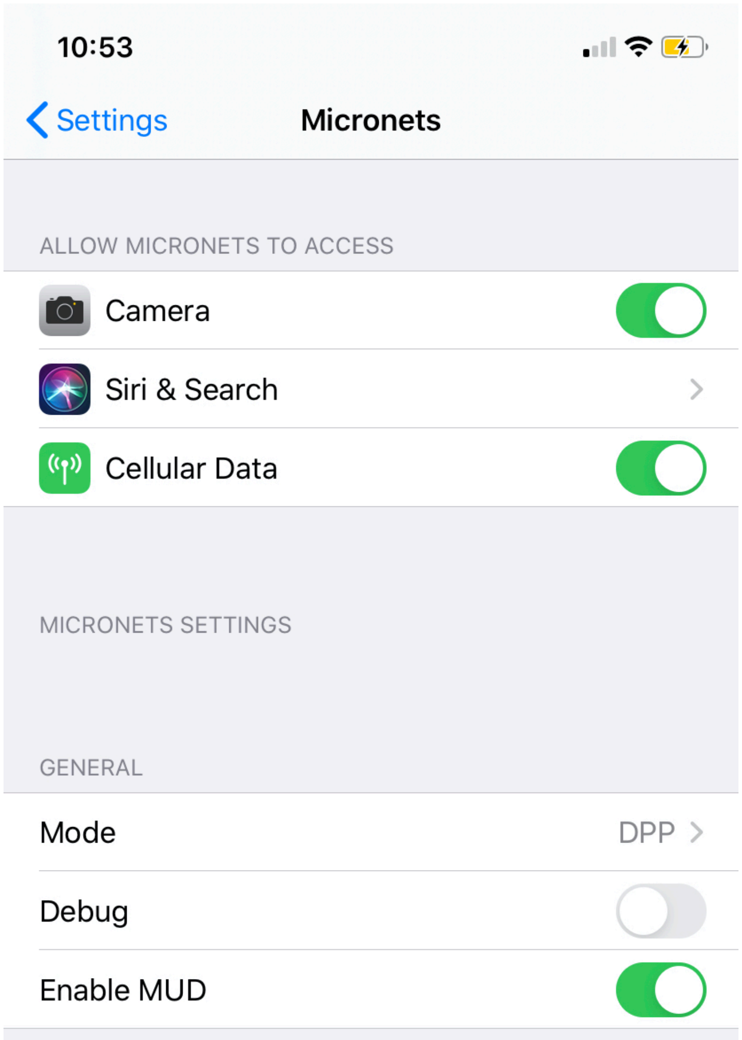


Modify the following fields in the general settings:

Mode - DPP or Clinic: We select DPP, if you are selecting the Clinic mode please follow the documentation for details related to the Clinic mode

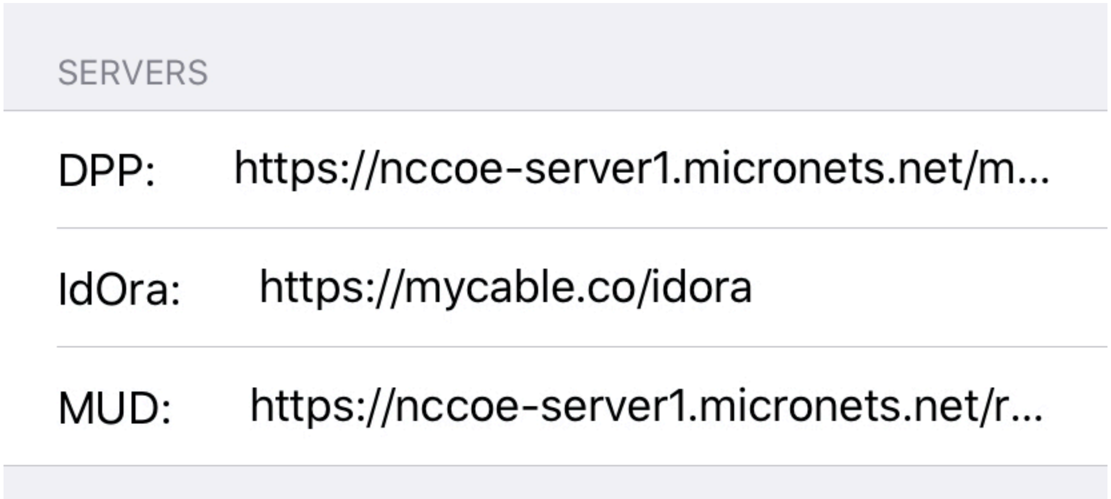
Debug - Leave this off as CableLabs will be deprecating this in the future

Enable MUD – If enabled, it will try to fetch the MUD file for the scanned device and pre-populate the Submit form prior to onboarding.

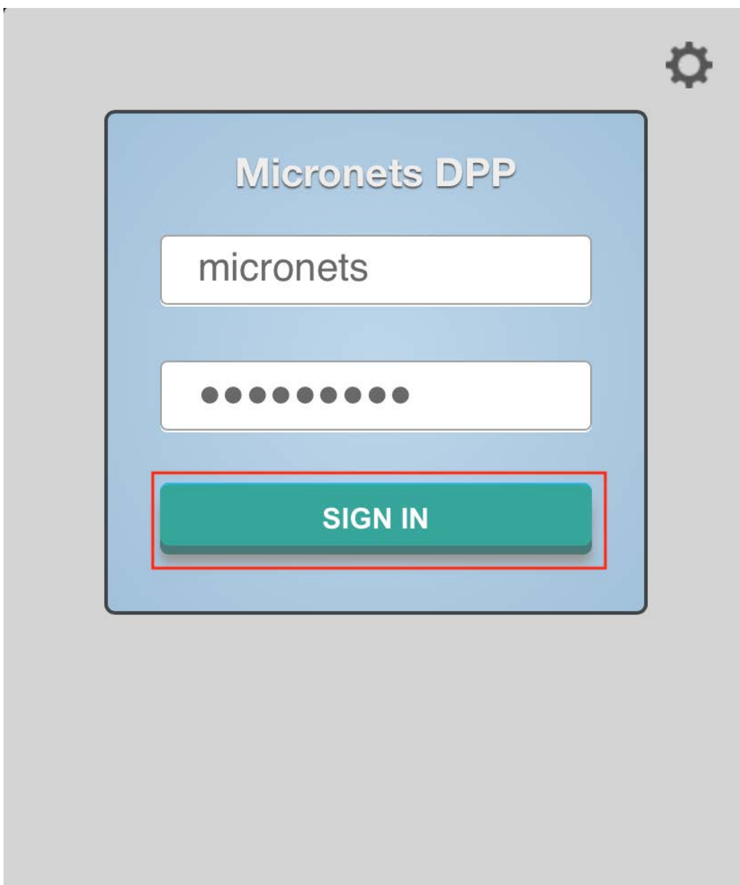


Modify the servers for the Micronets application:

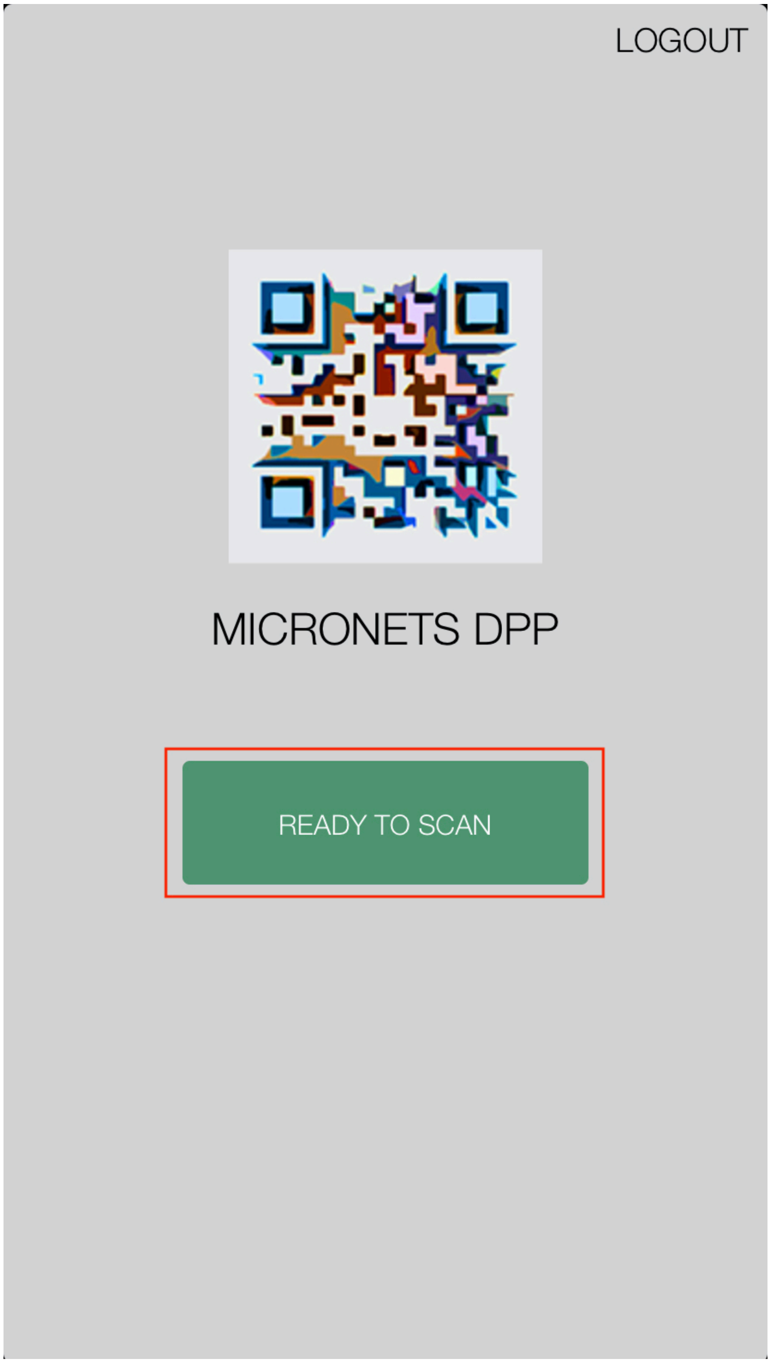
- **DPP** – MSO portal server URL for submitting onboard requests
- **IdOra** – Server for user authentication (Note: this is only required if utilizing the Clinic Mode)
- **MUD** – MUD registry server for looking up MUD files using the vendor code and public key in the QRCode. (Note: this only needs to be changed if you are deploying your own MUD registry)



Back on the Micronets mobile application, enter your subscriber credentials and click **SIGN IN**:

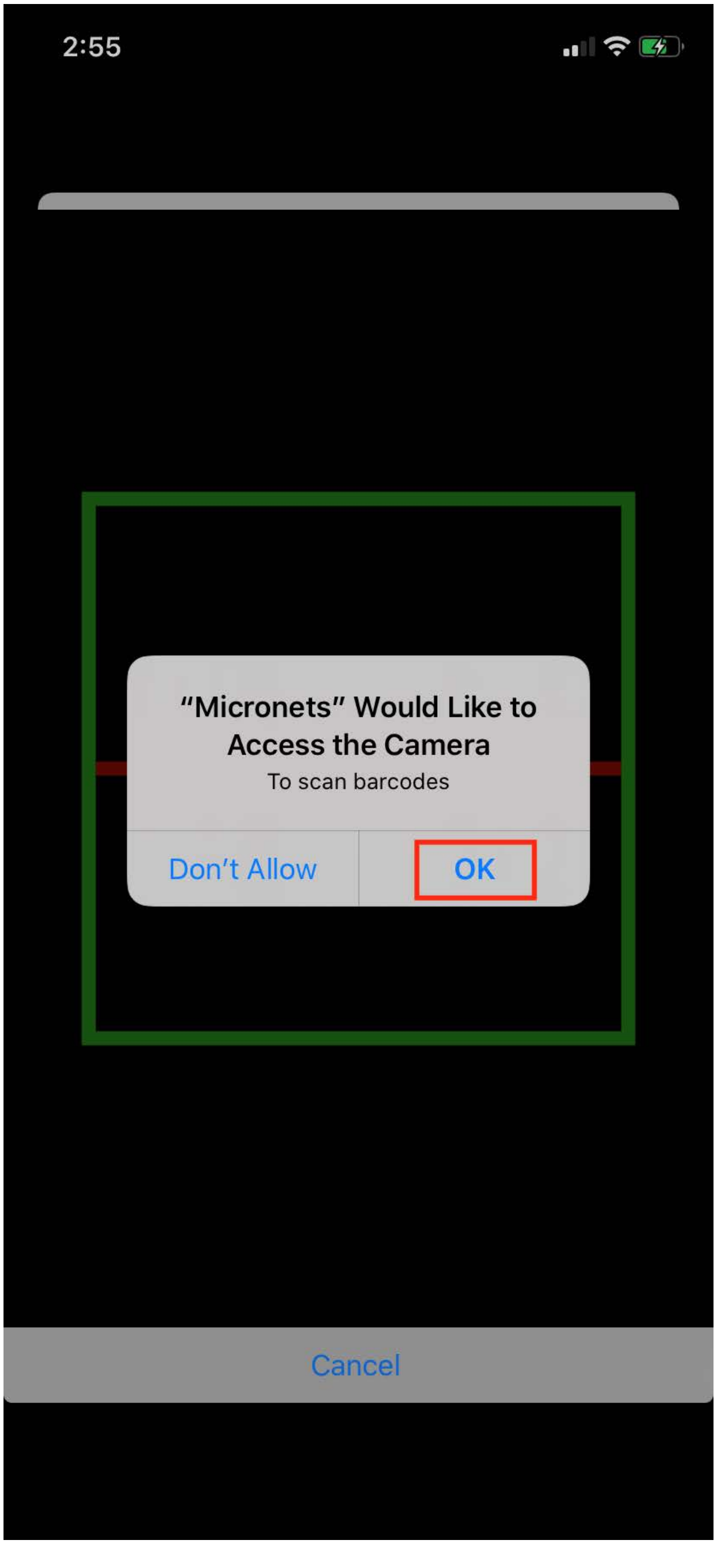


3160 Click the **READY TO SCAN** button to open the camera for onboarding:



3161

3162 If prompted, allow the Micronets application camera access, by clicking **OK**:


2:55
"Micronets" Would Like to Access the Camera
To scan barcodes
Don't Allow
OK
Cancel

3163

4.2.6 Onboarding Micronets Proto-Pi to a micronet

This section describes how to onboard a configured Micronets Proto-Pi device to a micronet using the Micronet iPhone app. For additional instructions not detailed in this documentation, please follow the link to the CableLabs documentation:https://github.com/cablelabs/micronets-pi3/blob/nccoe-build-3/README.md#Operation.

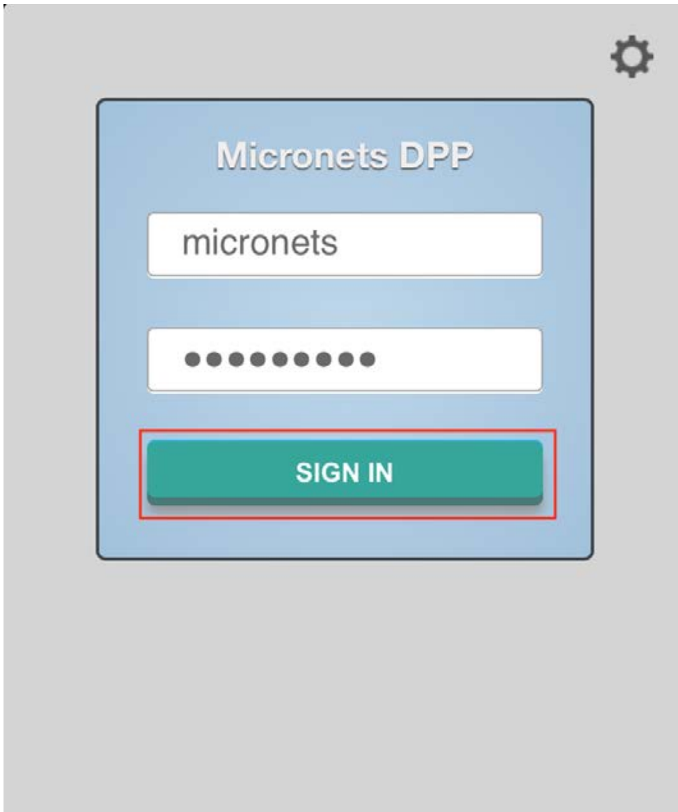
4.2.6.1 Prerequisites

To successfully complete this section the following is required:

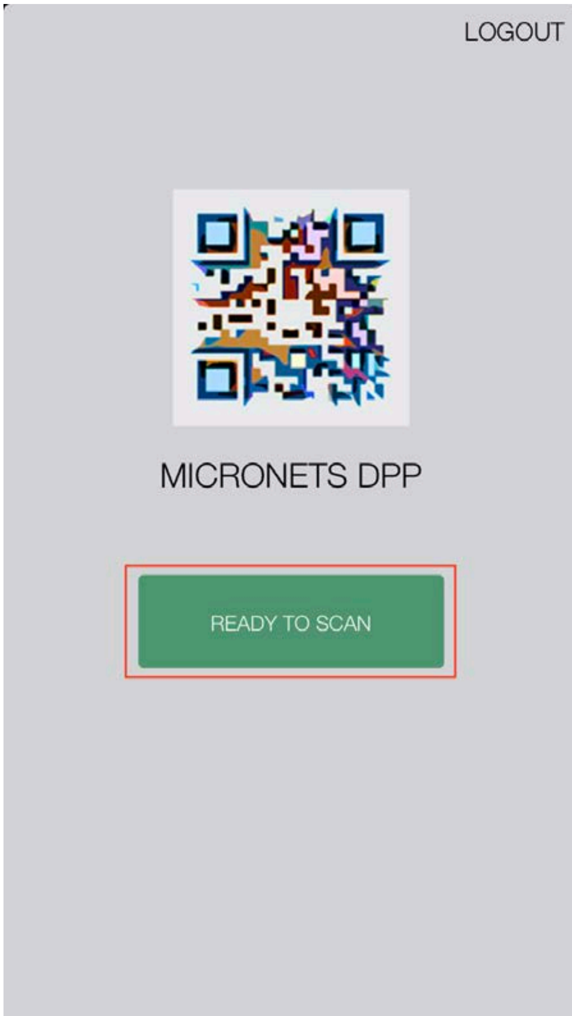
- a Raspberry Pi with the Micronets Proto-Pi software installed and configured
- an iOS or Android phone with the Micronets application installed and configured
- a Micronets subscriber account configured in Section 4.2.1
- a gateway device associated with the Micronets subscriber configured in Section 4.2.2

4.2.6.2 Instructions

1. If leveraging the self-registration feature for MUD onboarding, ensure that an ethernet cable is connected to the Raspberry Pi running the Micronets Proto-Pi software.
2. Power on the Pi device. If leveraging the self-registration feature, the device will automatically be registered on first run.
3. On the mobile device, open the Micronets mobile application and log in with your subscriber credentials.

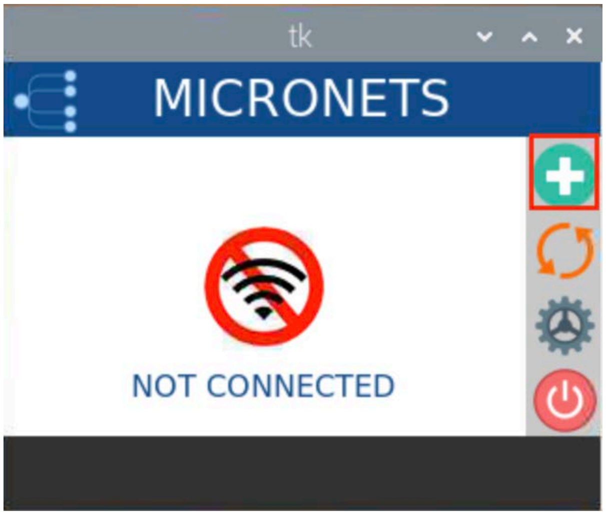


4. On the Mobile device, tap the **Ready to Scan** button:



3184

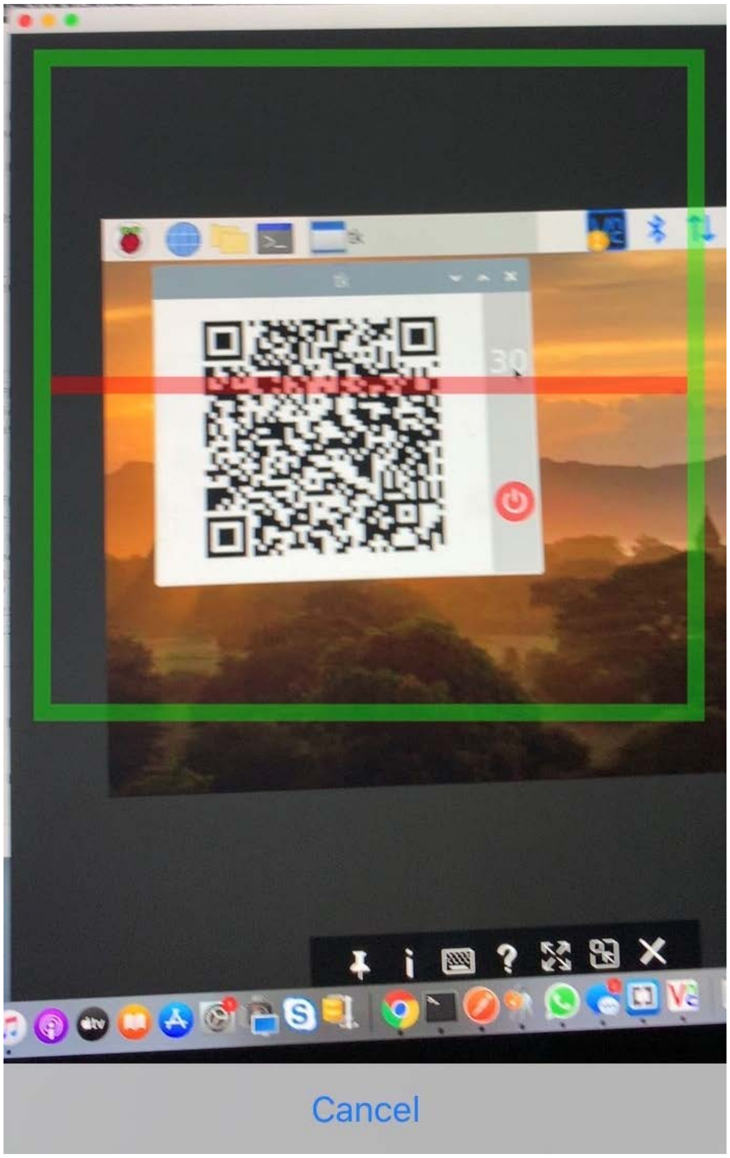
3185 5. On the Pi, click the Onboard icon:

3186 

3187 You should see a QR code appear on the screen:

3188 

3189 6. Scan the QRCode with the Micronet mobile application:

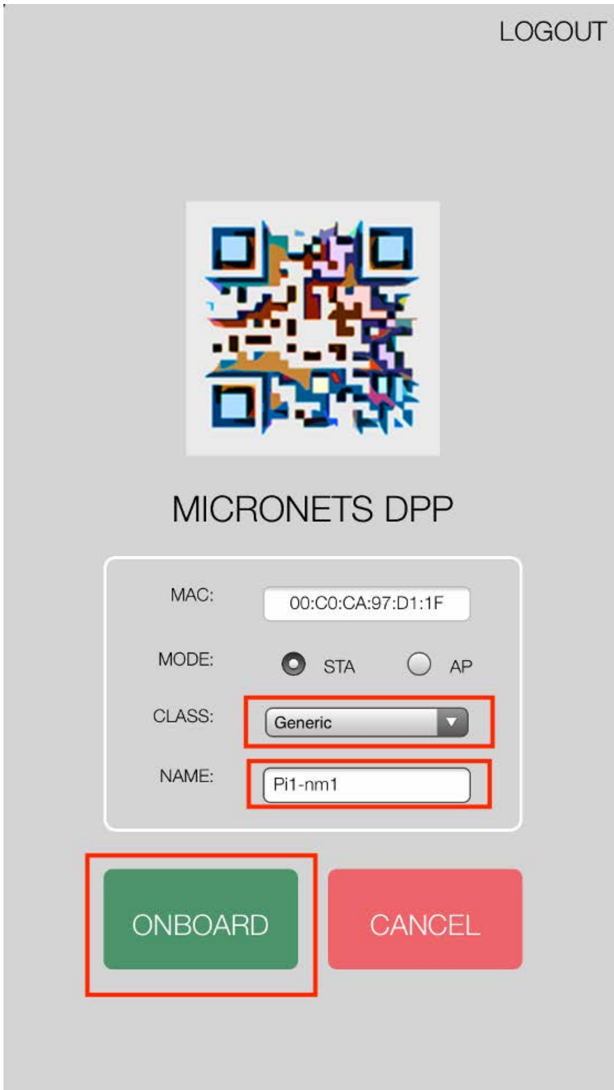


7. On the next screen that appears on the Micronets mobile application, input the following information in a timely fashion (Note: these steps must be completed while the device is still in onboard mode).

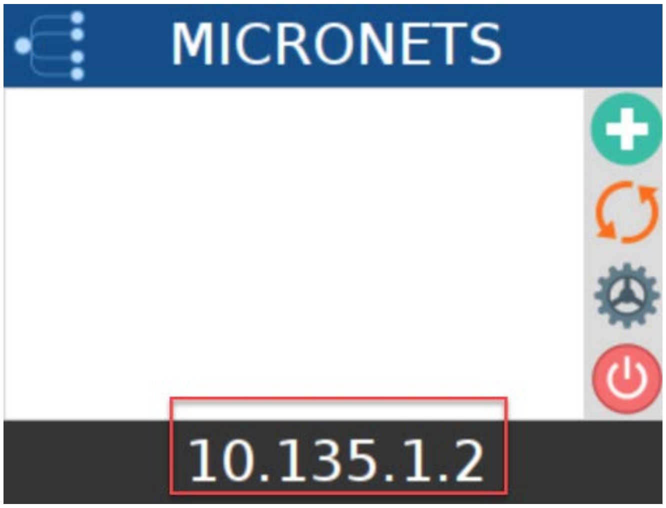
    a. If a MUD file was found, the device CLASS and NAME will be prepopulated, modify as needed. In the case that a MUD file was not found populate the **CLASS** and **NAME** manually.

b. Set the MODE to **STA** (Note: The Mode should always be STA as of the time of this implementation).

c. Tap the **ONBOARD** button to send the onboarding request to the MSO portal:



8. On the Pi you will see the device has been onboarded to the Micronets Gateway and has received an IP address:



4.2.7 Interacting with Micronets Manager

The Micronets Manager, which is hosted in the cloud, has API endpoints exposed in order to allow implementers to manage the Micronets Gateway through the Micronets Manager service. This section describes how to set up postman and execute different functions.

4.2.7.1 Prerequisites

In order to successfully complete this section of the documentation, be sure to have completed the product installation section above and downloaded the postman application onto a laptop that has internet access: https://www.postman.com/downloads/.

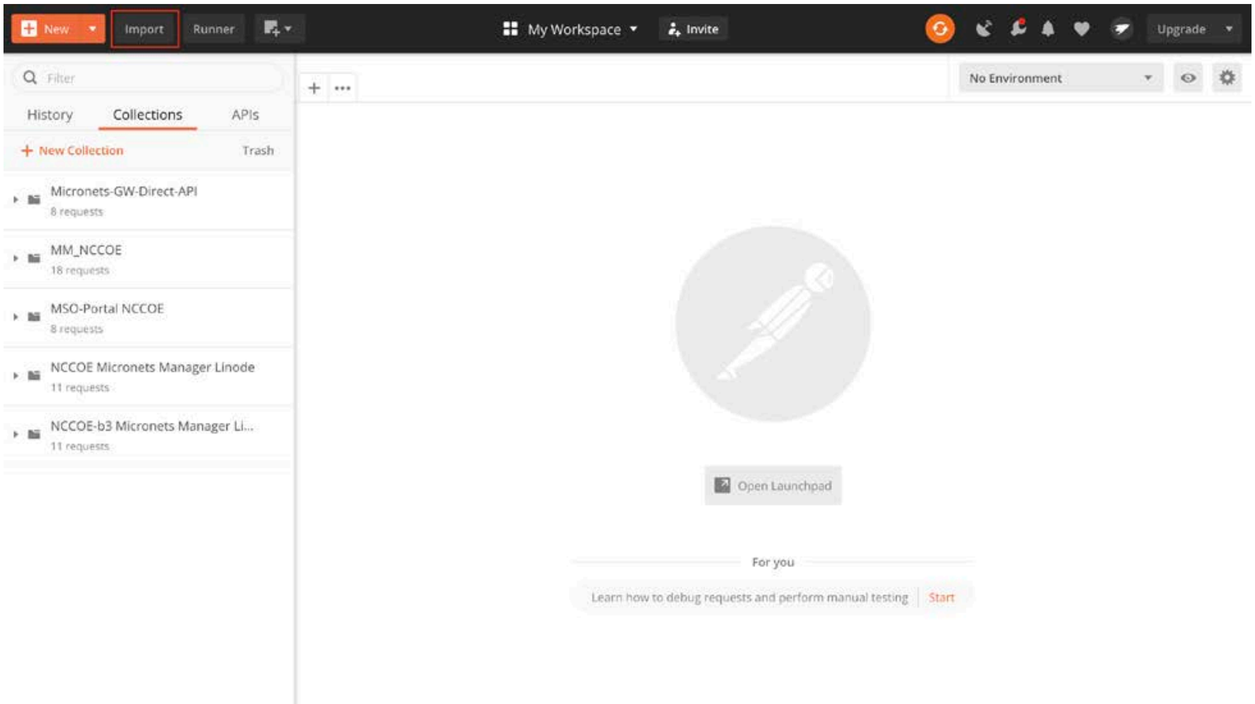
4.2.7.2 Instructions

1. Once Postman is installed and set up on the laptop, proceed to the following site to download the Micronets Manager Linode postman collections:

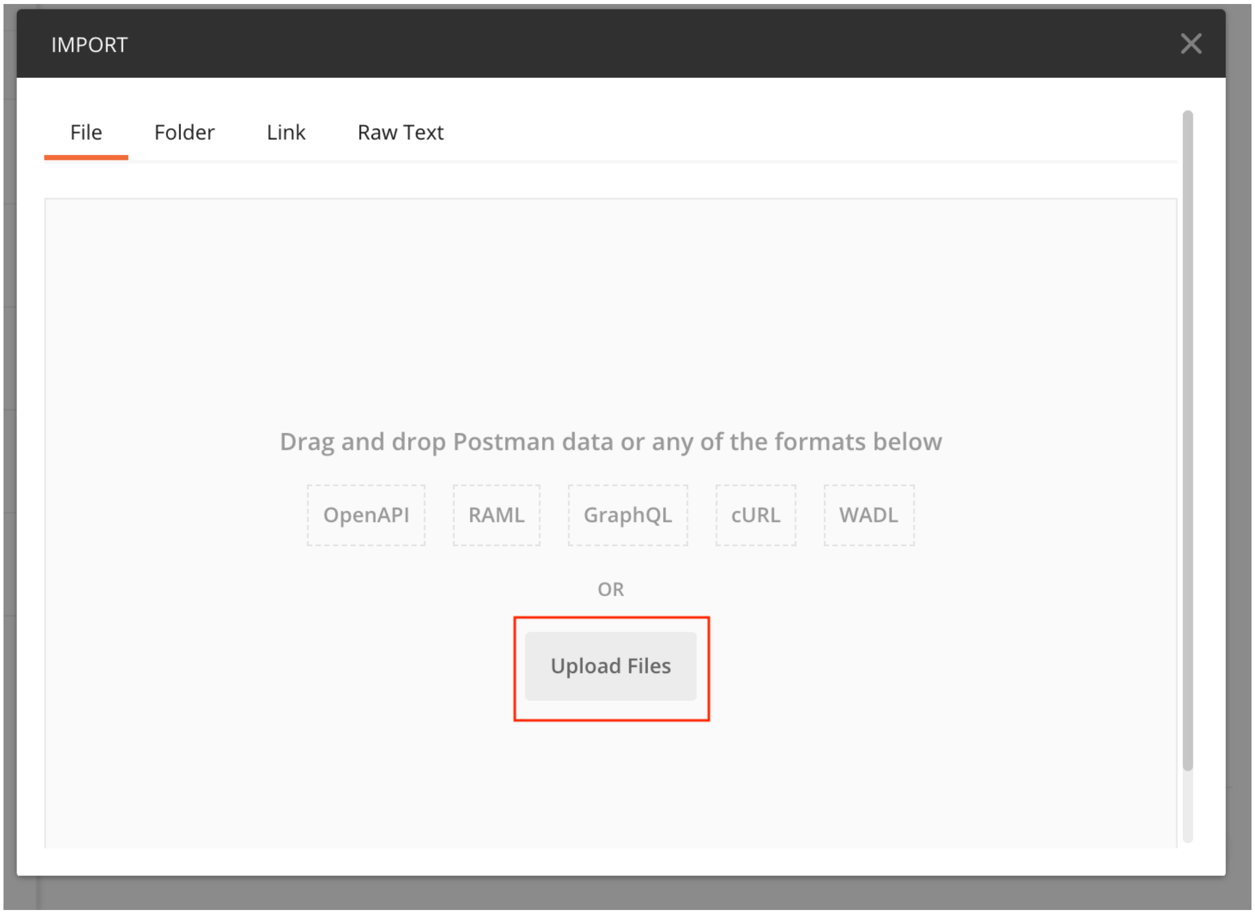
   Follow the links:
   https://raw.githubusercontent.com/cablelabs/micronets-manager/nccoe-build-3/scripts/Micronets_Manager_API.postman_collection.json

   https://raw.githubusercontent.com/cablelabs/micronets-manager/nccoe-build-3/scripts/Micronets_Manager_API.postman_globals.json

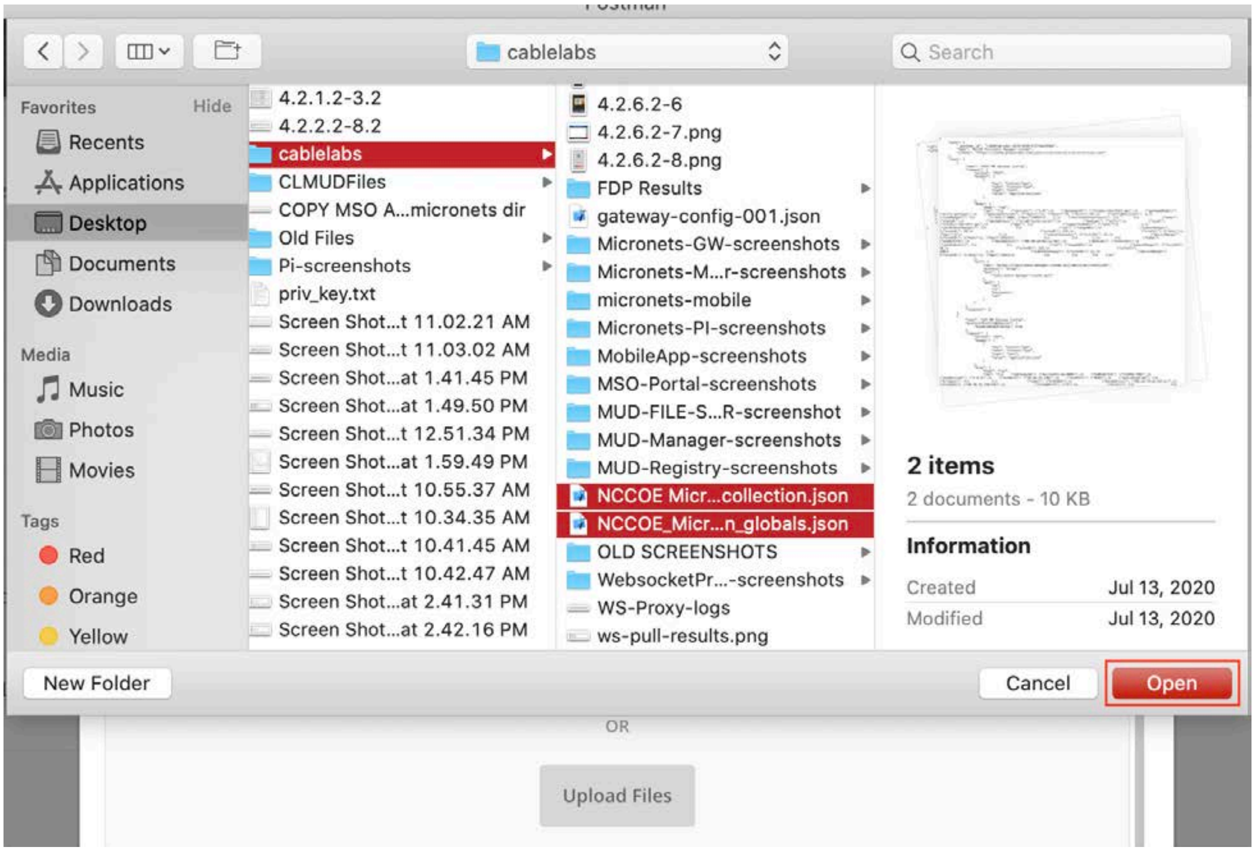
2. Open the postman application and sign in.
3. Click the import button to import the collections downloaded in step 1:



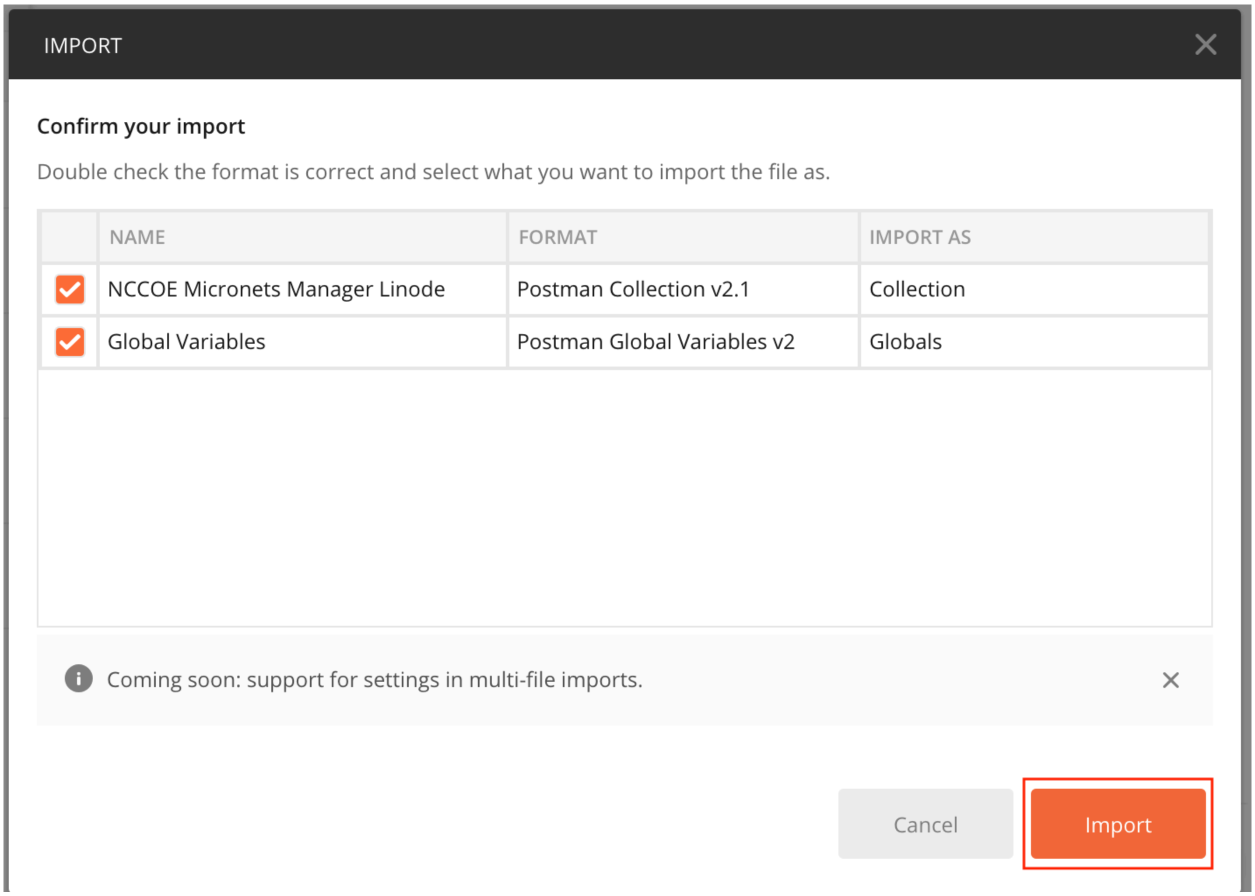
4. Next, click **upload files:**



5. Select the postman and global environmental variables collections downloaded in step 1:

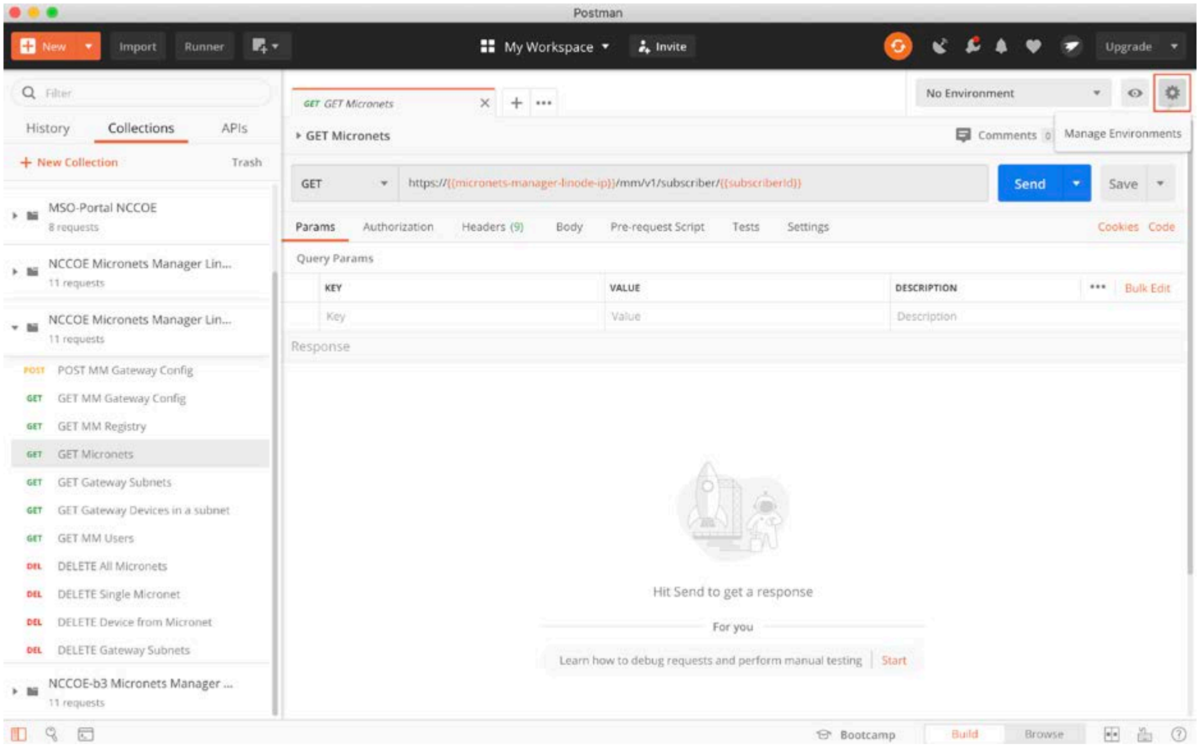


6. Confirm your import and click **Import**:

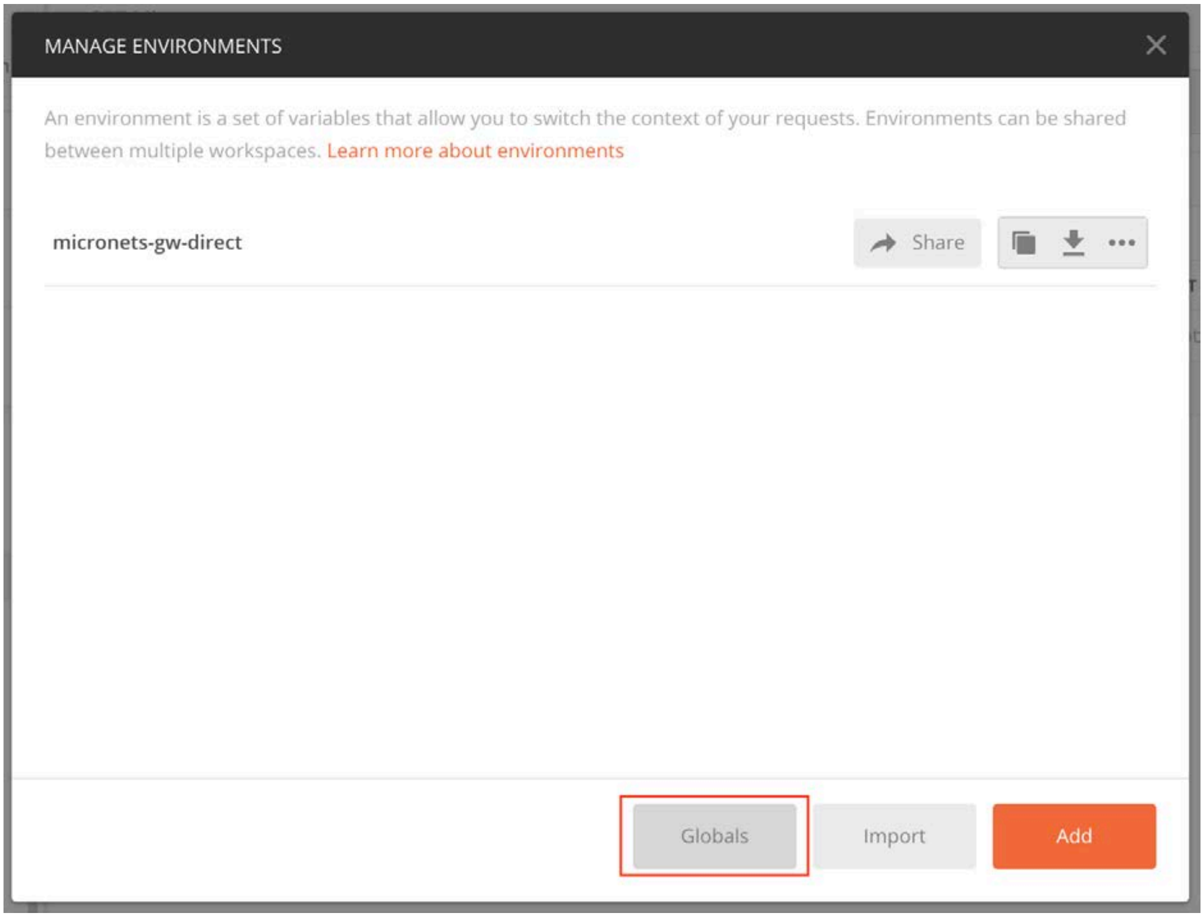


7. You will need to set the Globals for the micronets-manager-linode-ip, subscriberId and mso-portal-linode-ip:

    a. Click the gear button in the top right-hand corner of application to **Manage Environments**:



b. Click **Globals**:



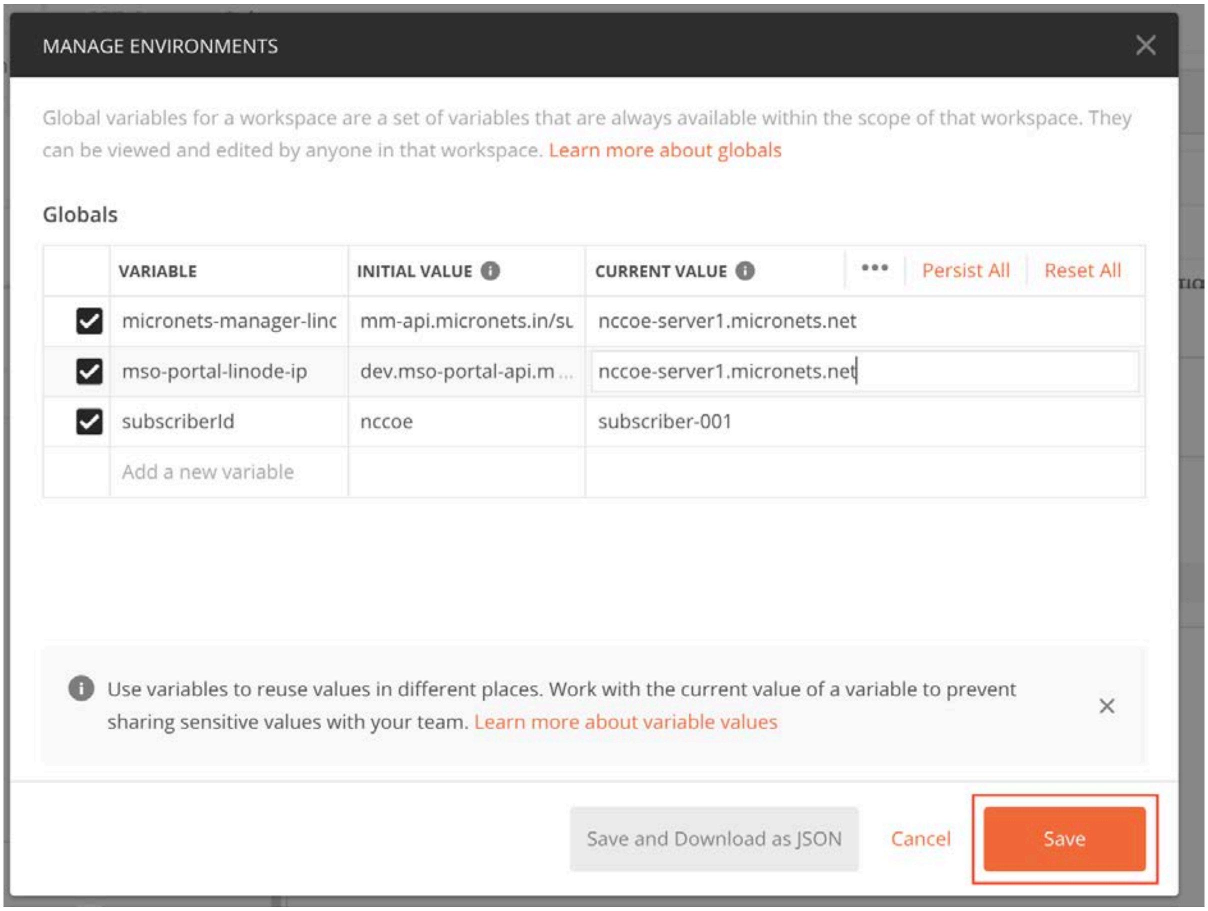
c. Modify the current values for the **micronets-manager-linode-ip, subscriberId and mso-portal-linode-ip** variables as follows and click **Save**:

micronets-manager-linode-ip: nccoe-server1.micronets.net

subscriberId: subscriber-001

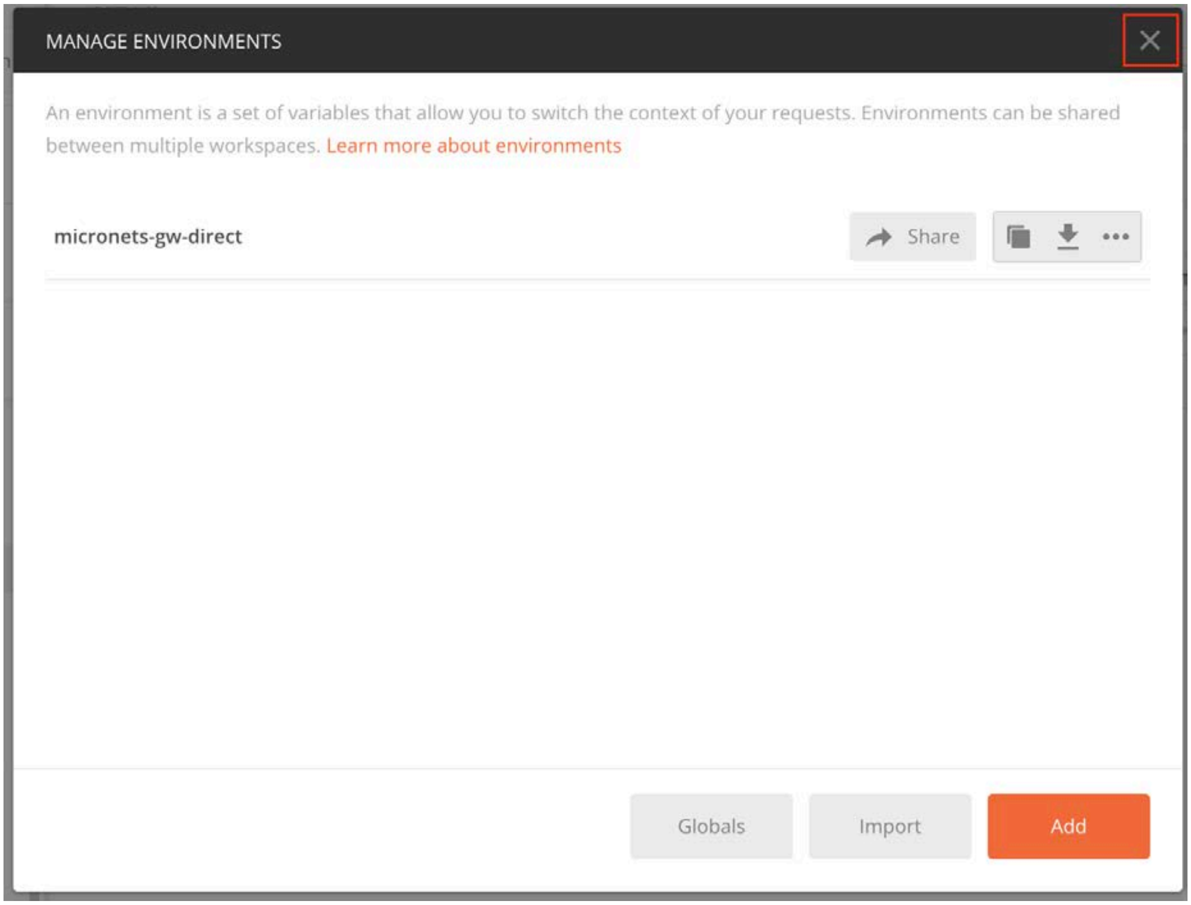
mso-portal-linode-ip: nccoe-server1.micronets.net

3242



3243

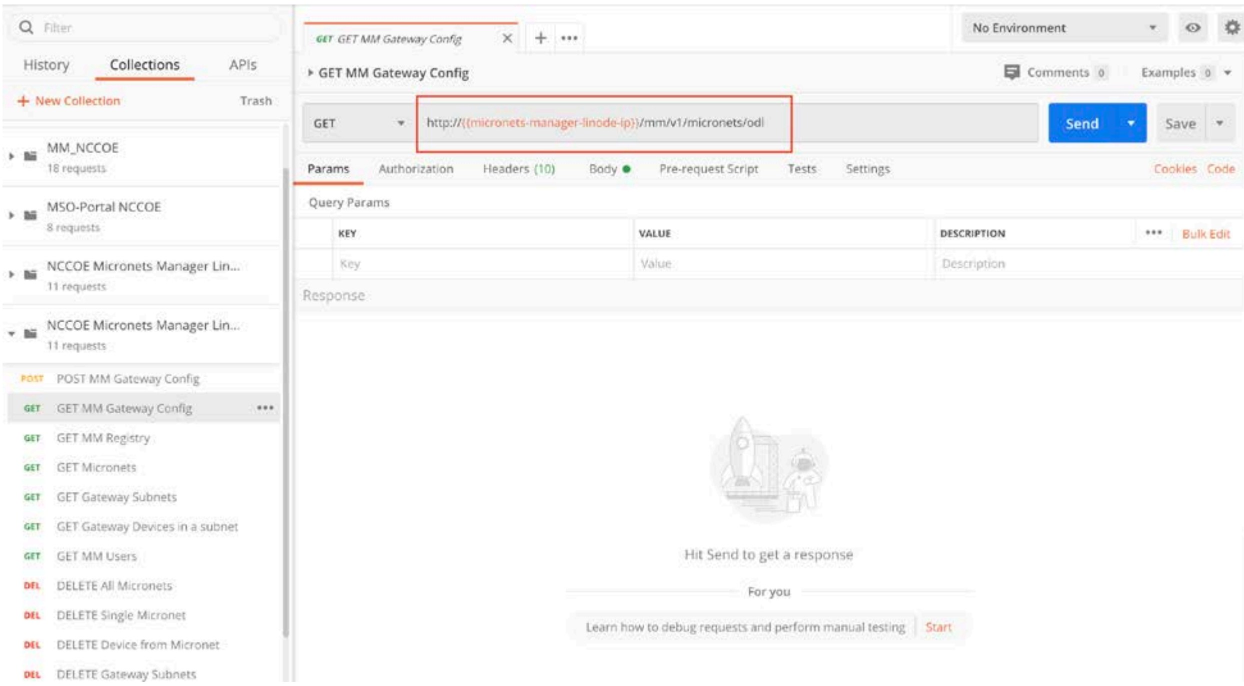
3244 d. Exit out of the menu:



Next, open the postman collection and review and modify the URLs for the calls to ensure the API endpoint paths match your implementation:

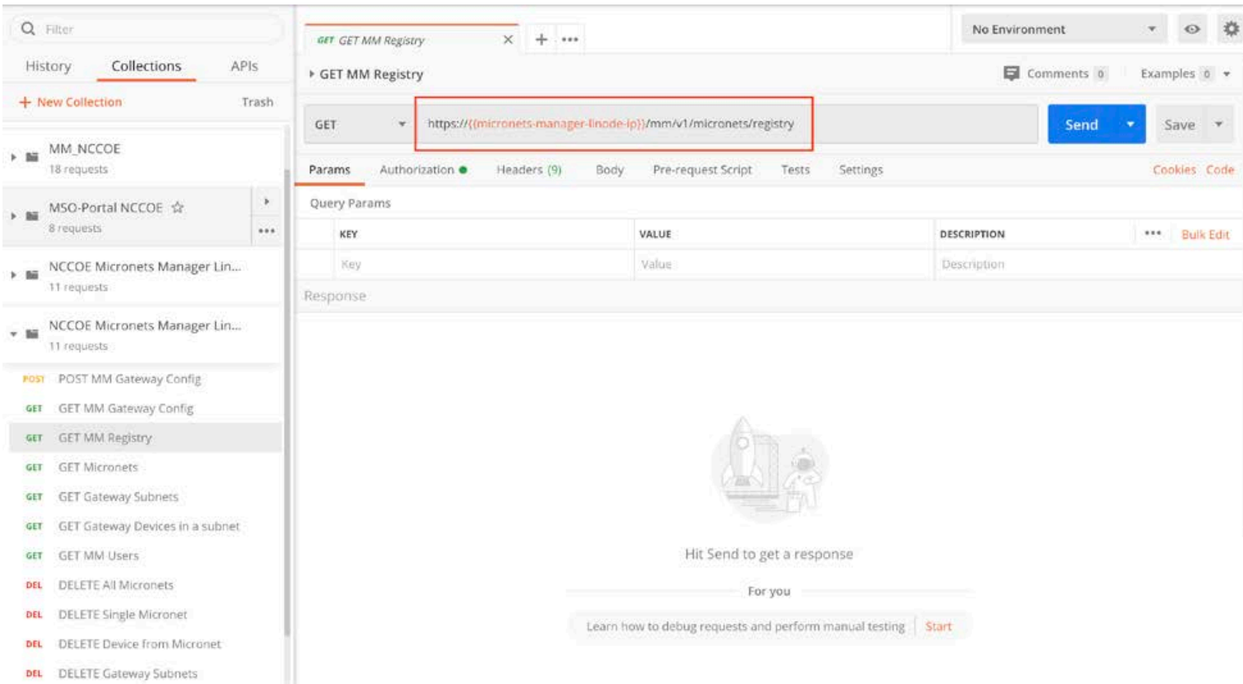
a. Modify the **GET MM Gateway Config** command to reflect the following. Executing this command will pull the current Gateway config from the Micronets Manager:

```
http://{{micronets-manager-linode-ip}}/mm/v1/micronets/odl
```



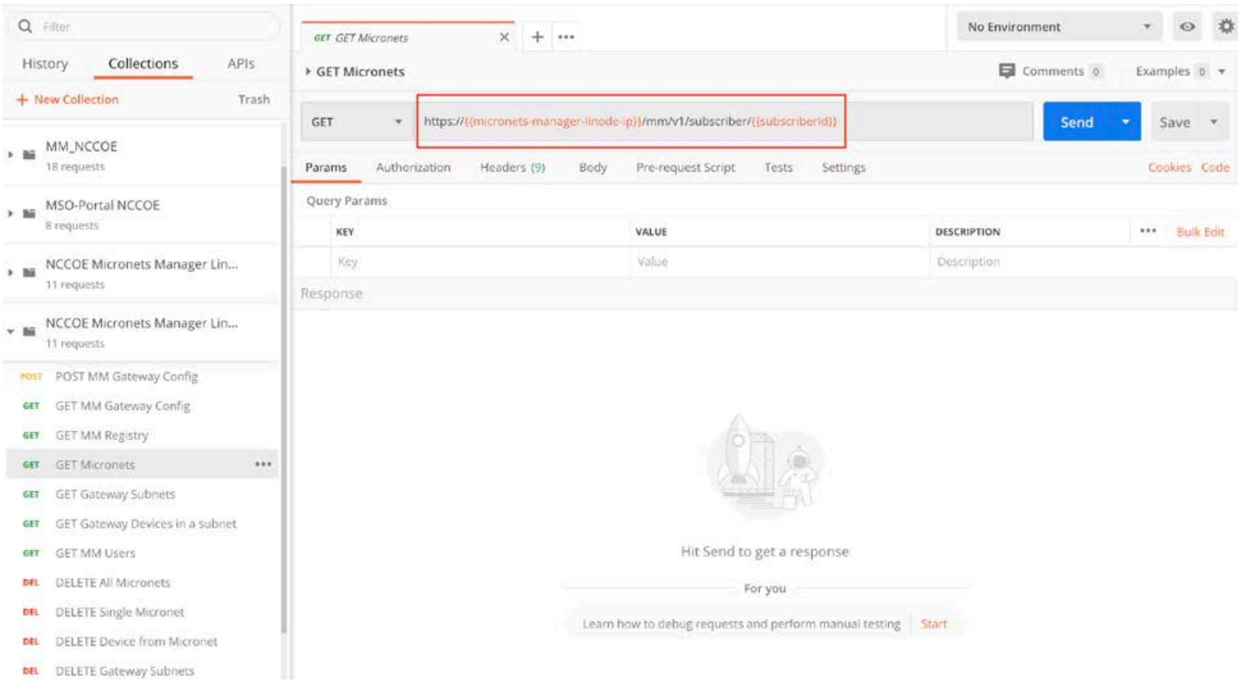
b. Modify the **GET MM Registry** command to reflect the following. Executing this command will pull the current registry from the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/mm/v1/micronets/registry
```



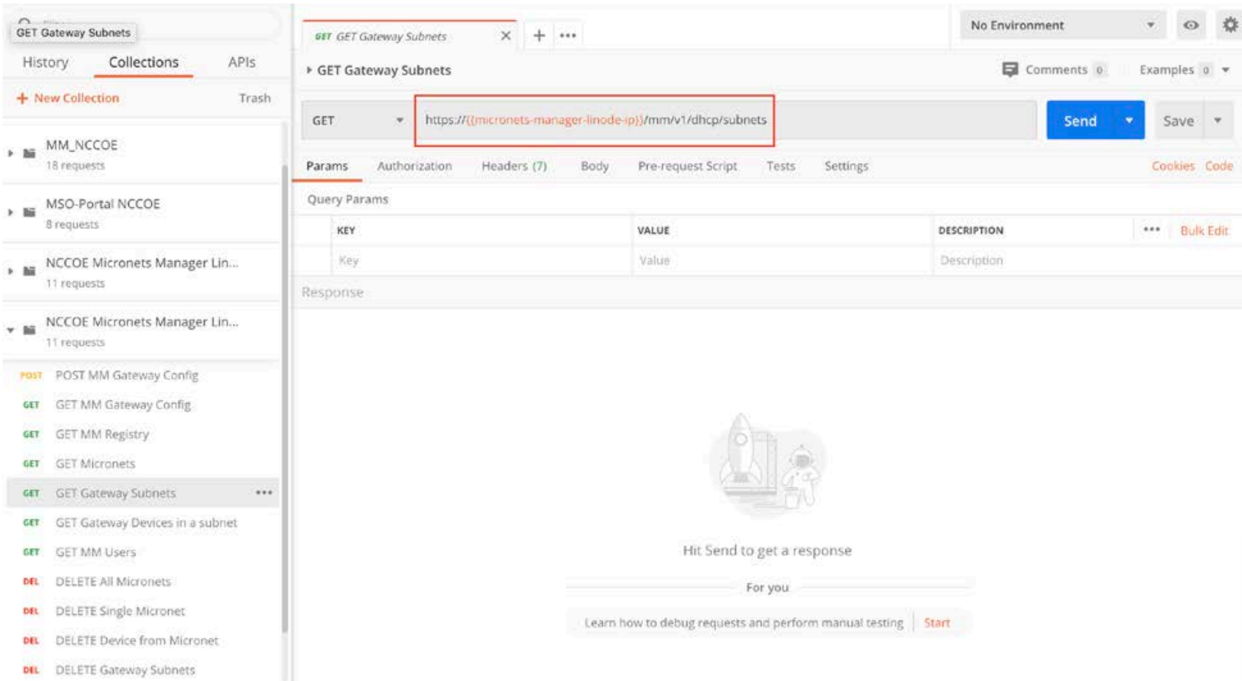
Modify the **GET Micronets** command to reflect the following. Executing this command will pull a list of the current micronets on the Gateway from the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/subscriber/{{subscriberId}}
```



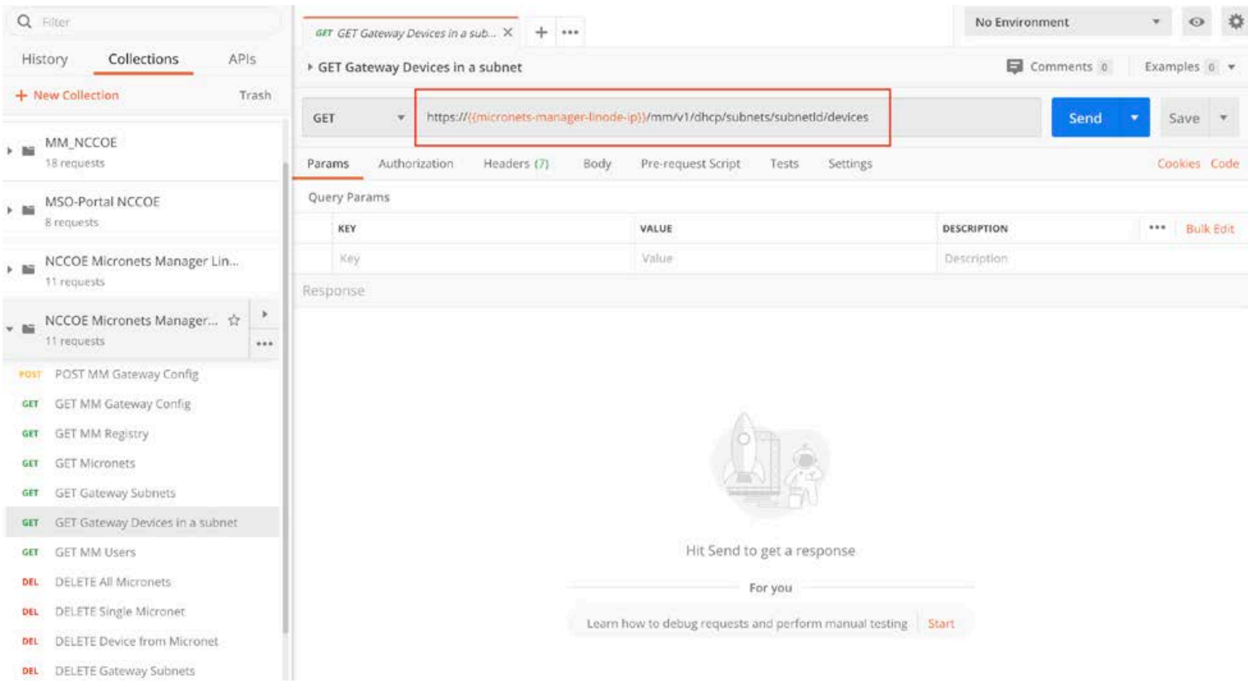
d. Modify the **GET Gateway Subnets** command to reflect the following. Executing this command will pull a list of the current subnets on the Gateway from the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/dhcp/subnets
```



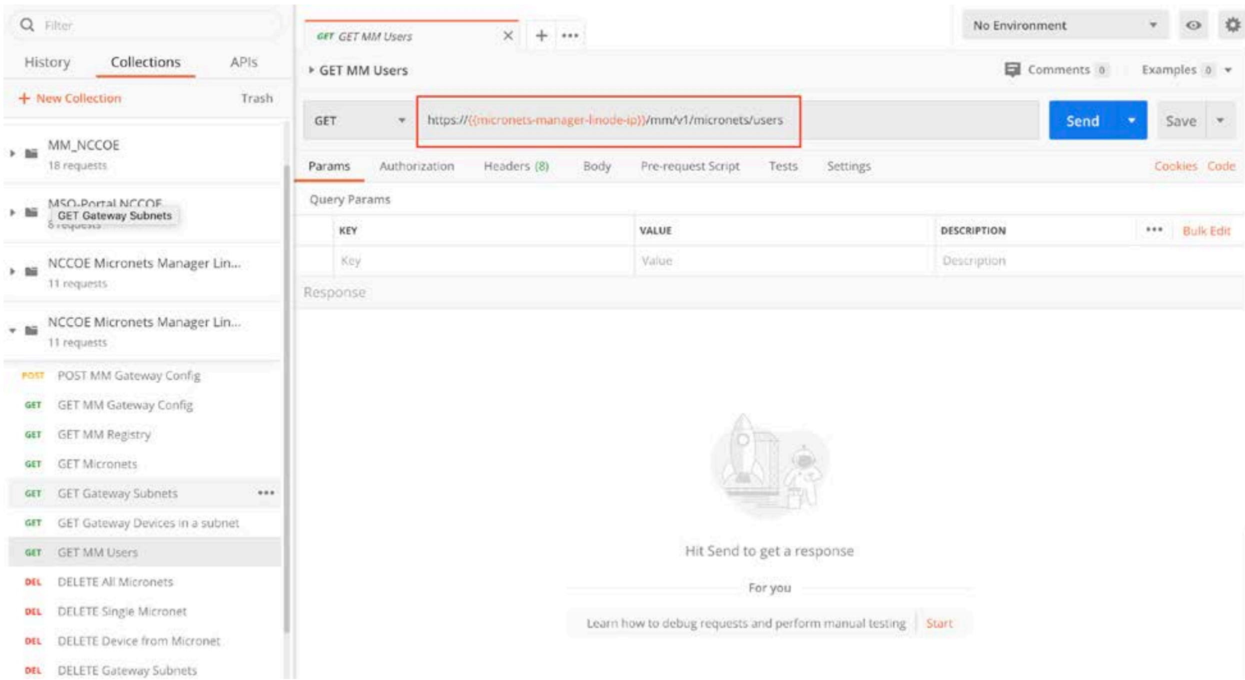
e. Modify the **GET Gateway Devices in a subnet** command to reflect the following. Executing this command will pull a list of the current devices in a subnet on the Gateway from the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/dhcp/subnets/subnetId/devices
```



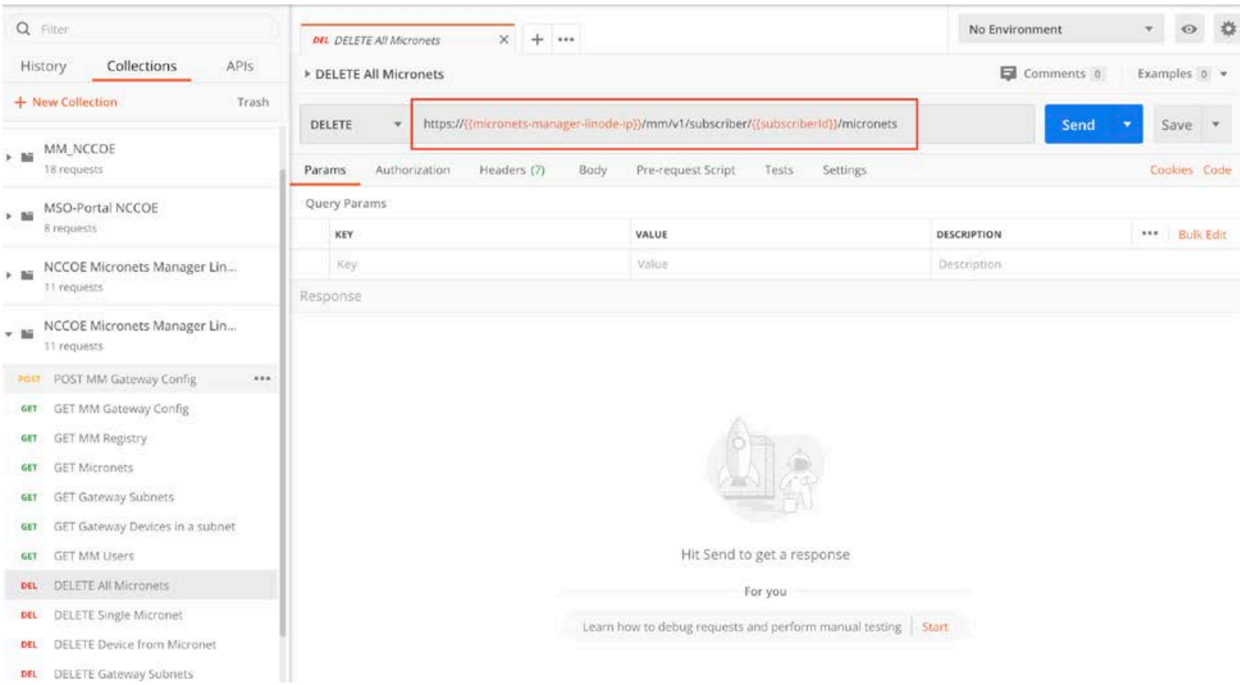
f. Modify the **GET MM Users** command to reflect the following. Executing this command will pull a list of the users associated with the subscriber ID from the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/micronets/users
```



g. Modify the **DELETE All Micronets** command to reflect the following. Executing this command will delete all of the current micronets on the Gateway via the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/subscriber/{{subscriberId}}/micronets
```

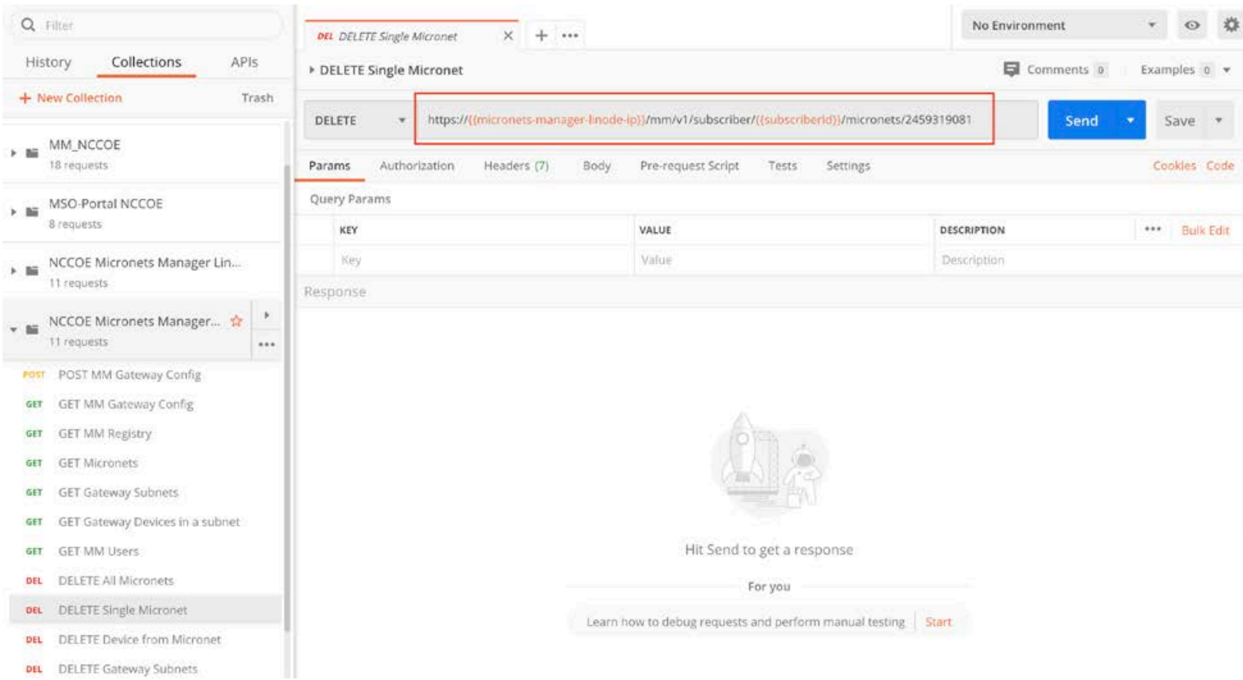


h. Modify the **DELETE Single Micronets** command to reflect the following. Executing this command will delete a specific micronet on the Gateway via the Micronets Manager. This command is to be modified before executing to specify the **<micronetID>**, which can be retrieved by executing the GET Micronets command:

https://{{micronets-manager-linode-ip}}/sub/{{subscriberId}}/api/mm/v1/subscriber/{{subscriberId}}/micronets/**<micronetID>**

Below is an example of this command:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscriberId}}/api/mm/v1/subscriber/{{subscriberId}}/micronets/2453819029
```

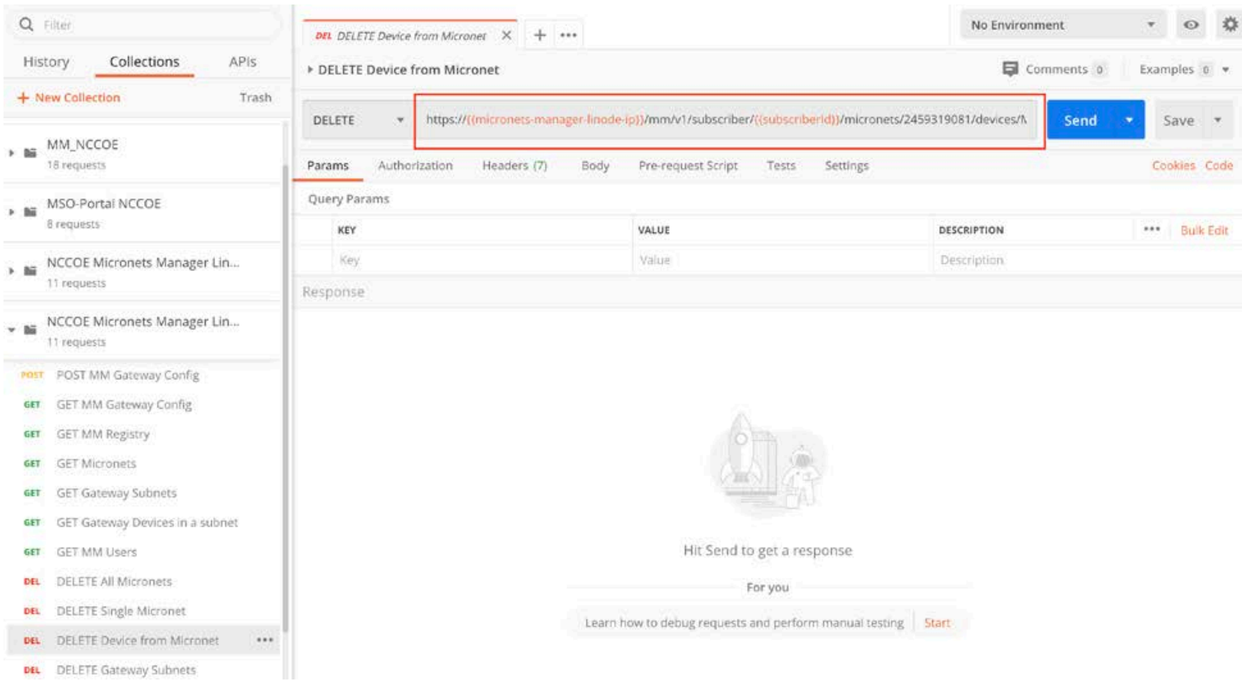


i. Modify the **DELETE Device from Micronet** command to reflect the following. Executing this command will delete a specific device from a particular micronet on the Gateway via the Micronets Manager. This command is to be modified before executing to specify the **<micronetID>** and **<deviceID>**, which can be retrieved by executing the GET Micronets command:

https://{{micronets-manager-linode-ip}}/sub/{{subscriberId}}/api/mm/v1/subscriber/{{subscriberId}}/micronets/**<micronetID>** /devices/**<deviceID>**

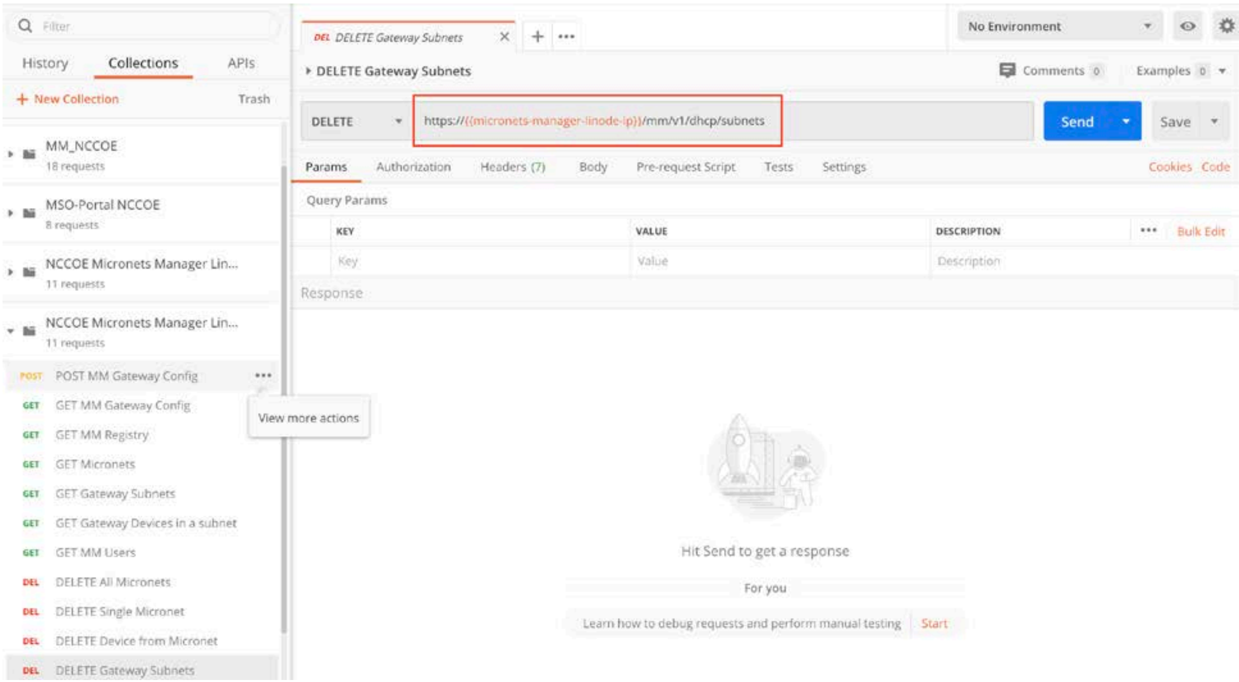
Below is an example of this command:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/subscriber/{{subscriberId}}/micronets/2136369149/de-
vices/da34c7219c2c97f0e2c2838e66c725d137f3c097
```



j. Modify the **DELETE Gateway Subnets** command to reflect the following. Executing this command will delete all subnets on the Gateway via the Micronets Manager:

```
https://{{micronets-manager-linode-ip}}/sub/{{subscrib-
erId}}/api/mm/v1/dhcp/subnets
```


Filter
History
Collections
APIs
+ New Collection
Trash
MM_NCCOE
18 requests
MSO-Portal NCCOE
8 requests
NCCOE Micronets Manager Lin...
11 requests
NCCOE Micronets Manager Lin...
11 requests
POST MM Gateway Config
GET MM Gateway Config
GET MM Registry
GET Micronets
GET Gateway Subnets
GET Gateway Devices in a subnet
GET MM Users
DELETE All Micronets
DELETE Single Micronet
DELETE Device from Micronet
DELETE Gateway Subnets
View more actions
DEL DELETE Gateway Subnets
No Environment
DELETE Gateway Subnets
Comments 0
Examples 0
DELETE
https://{{micronets-manager-linode-ip}}/mm/v1/dhcp/subnets
Send
Save
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Response
Hit Send to get a response
For you
Learn how to debug requests and perform manual testing
Start

3307

4.2.8 Removing Micronets Proto-Pi from a Micronet

Removing a Micronets Proto-Pi from a micronet will remove the network credentials from the device. For additional instructions not detailed in this documentation, please follow the link to the CableLabs documentation: https://github.com/cablelabs/micronets/blob/nccoe-build-3/docs/operation/pi-offboarding.md.

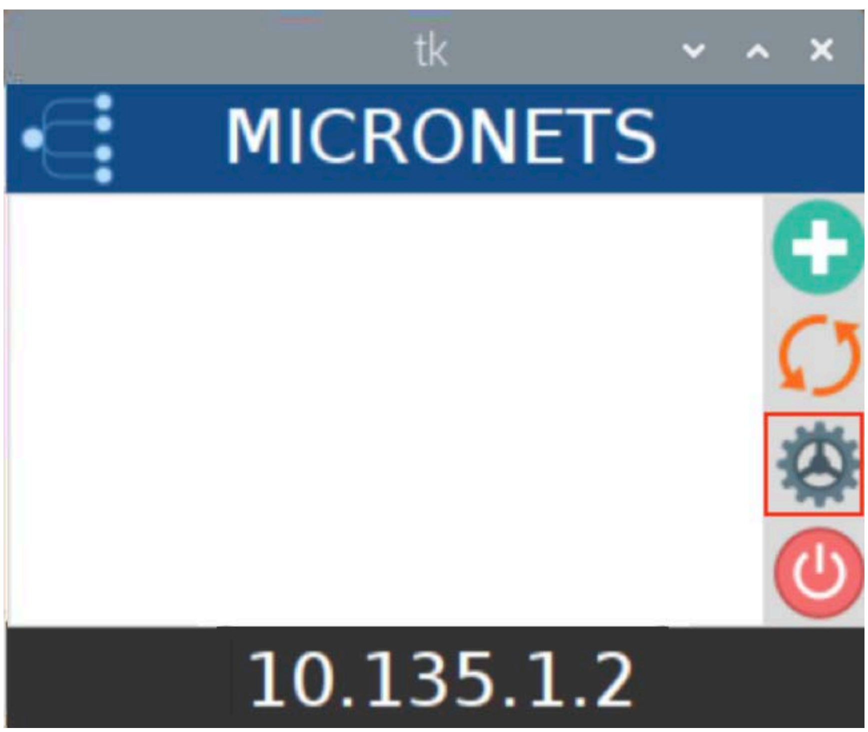
4.2.8.1 Prerequisites

To successfully complete this section, the following are required:

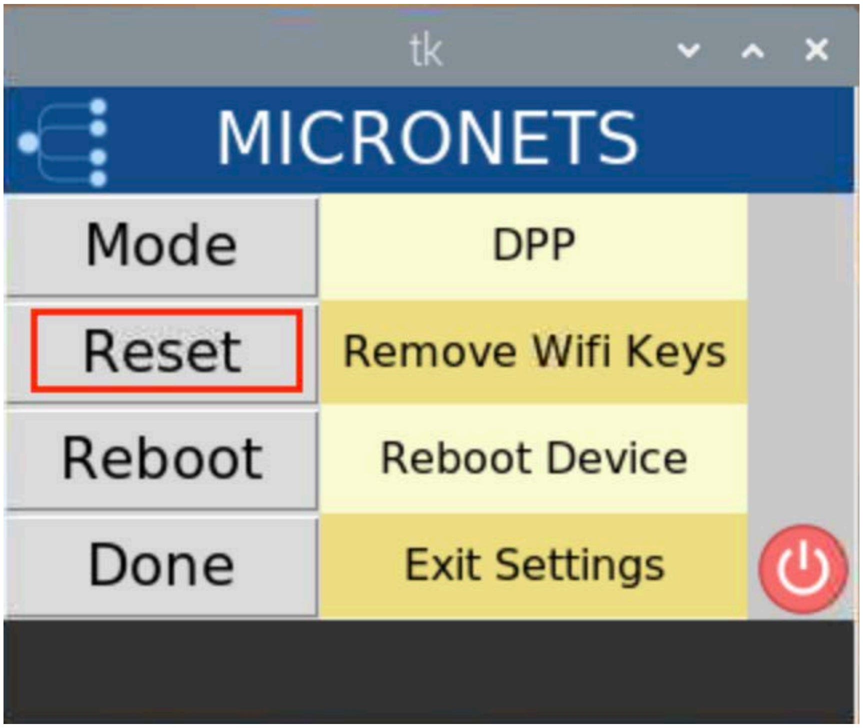
- a Raspberry Pi with the Micronets Proto-Pi software installed and configured
- a device that is currently onboarded to the Micronets Gateway

4.2.8.2 Instructions:

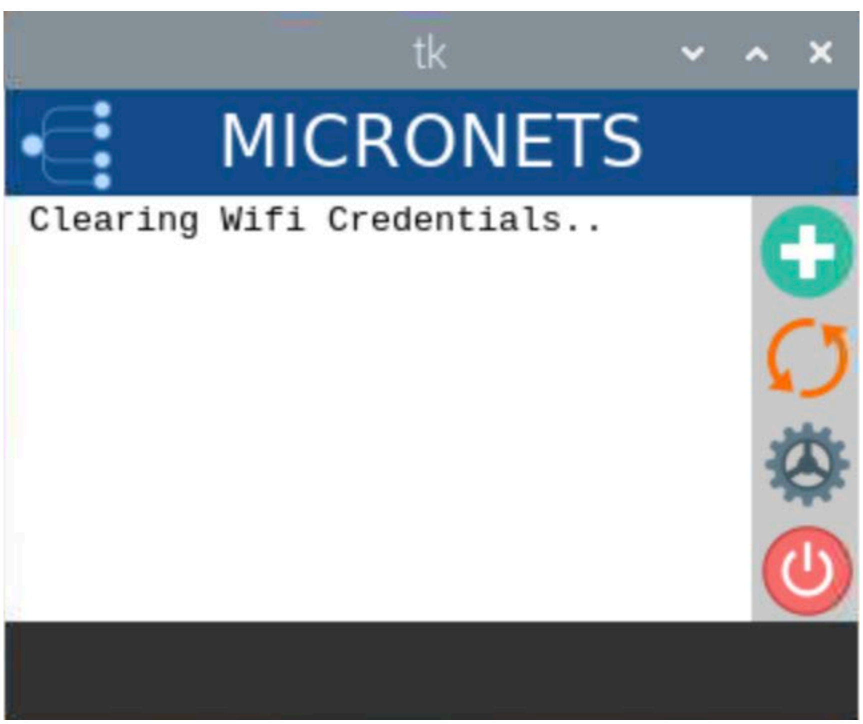
1. Power on the Micronets Proto-Pi device.
2. Tap Settings:



3. Tap Reset:



You should see output similar to the following:



4.2.9 Removing an MSO Subscriber

Removing a subscriber involves removing the subscriber from the MSO portal database, removing the subscriber's micronets, and removing the subscriber's Micronets Manager. For additional instructions not detailed in this documentation, please follow the link to the CableLabs documentation: https://github.com/cablelabs/micronets/blob/nccoe-build-3/docs/operation/pi-offboarding.md.

4.2.9.1 Prerequisites

To successfully complete this section be sure to have completed both the product installation section and . Ensure all steps have been successfully completed before proceeding to the instructions.

4.2.9.2 Instructions

1. Remove the subscriber from the MSO portal using:

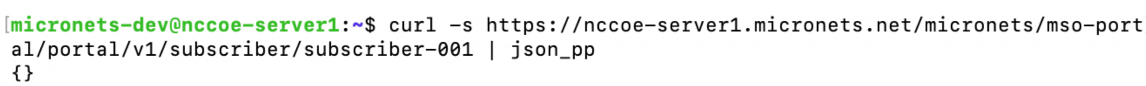
   ```
   curl -s -X DELETE https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1/subscriber/subscriber-001 | json_pp
   ```

2. Verify that the subscriber is removed from the MSO portal by executing the following commands:

   a. Check if the subscriber ID is present in the subscriber list:

      ```
      curl -s https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1/subscriber/subscriber-001
      ```

      You should see output similar to the following:

   b. Next, check if the user is present in the list of users in the MSO portal:

      ```
      curl -s https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1/users | json_pp
      ```

      You should see output similar to the following:

```
[micronets-dev@nccoe-server1:~$ curl -s https://nccoe-server1.micronets.net/micronets/mso-port]
al/portal/v1/users | json_pp
{
   "limit" : 500,
   "data" : [],
   "skip" : 0,
   "total" : 0
}
```

c. Finally, check to see if there is a socket present for the subscriber ID:

```
curl -s https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1/socket/subscriber-001 | json_pp
```

You should see output similar to the following:

```
[micronets-dev@nccoe-server1:~$ curl -s https://nccoe-server1.micronets.net/micronets/mso-port]
al/portal/v1/socket/subscriber-001 | json_pp
{
   "name" : "NotFound",
   "className" : "not-found",
   "errors" : {},
   "code" : 404,
   "message" : "No record found for id 'subscriber-001'"
}
```

Note: There could be scenarios where the commands above do not show empty lists. If that is the case, the subscriber has not been deleted properly. You can delete the subscriber entries in the MSO portal subtables by executing the following commands:

d. Delete the subscriber ID from the user list manually:

```
curl -s -X DELETE https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1/users/subscriber-001 | json_pp
```

e. Delete the subscriber ID from the socket list manually:

```
curl -s -X DELETE https://nccoe-server1.micronets.net/micronets/mso-portal/portal/v1/socket/subscriber-001
```

3. Remove all the micronets for the subscriber using:

```
curl -s -X DELETE https://nccoe-server1.micronets.net/sub/subscriber-001/api/mm/v1/subscriber/subscriber-001/micronets
```

You should see output similar to the following:

```
[micronets-dev@nccoe-server1:~$ curl -s -X DELETE https://nccoe-server1.micronets.net/sub/subs
criber-001/api/mm/v1/subscriber/subscriber-001/micronets
{"_id":"5f04e7308a84ec1a8feab599","id":"subscriber-001","name":"Subscriber 001","ssid":"micro
nets-gw","gatewayId":"micronets-gw","micronets":[],"createdAt":"2020-07-07T21:20:48.597Z","up
datedAt":"2020-07-13T21:19:36.184Z","__v":0}micronets-dev@nccoe-server1:~$
```

This will remove the micronets on the connected Micronets Gateway. If the gateway is not connected to its peer Micronets Manager, the micronets can be deleted directly on the gateway using:

```
curl -s -X DELETE http://localhost:5000/micronets/v1/gateway/micronets
```

4. You can verify that the micronets have been deleted by running:

```
curl -s https://nccoe-server1.micronets.net/sub/subscriber-001/api/mm/v1/subscriber/subscriber-001/micronets
```

This should return an empty micronets list.

5. Remove the Micronets Manager docker container for a subscriber by running:

```
/etc/micronets/micronets-manager.d/mm-container delete subscriber-001
```

You will be prompted to remove the config file:

```
micronets-dev@nccoe-server1:~$ /etc/micronets/micronets-manager.d/mm-container delete subscri
ber-001
Deleting resources for subscriber subscriber-001...
Stopping sub-subscriber-001_api_1     ... done
Stopping sub-subscriber-001_mongodb_1 ... done
Removing sub-subscriber-001_api_1     ... done
Removing sub-subscriber-001_mongodb_1 ... done
Removing network sub-subscriber-001_mm-priv-network
Removing volume sub-subscriber-001_mongodb
rm: remove write-protected regular file '/etc/nginx/micronets-subscriber-forwards/sub-subscri
ber-001.conf'? y
```

Lastly, you will be prompted to provide sudo privileges:

```
micronets-dev@nccoe-server1:~$ /etc/micronets/micronets-manager.d/mm-container delete subscri
ber-001
Deleting resources for subscriber subscriber-001...
Stopping sub-subscriber-001_api_1     ... done
Stopping sub-subscriber-001_mongodb_1 ... done
Removing sub-subscriber-001_api_1     ... done
Removing sub-subscriber-001_mongodb_1 ... done
Removing network sub-subscriber-001_mm-priv-network
Removing volume sub-subscriber-001_mongodb
rm: remove write-protected regular file '/etc/nginx/micronets-subscriber-forwards/sub-subscri
ber-001.conf'? y
removed '/etc/nginx/micronets-subscriber-forwards/sub-subscriber-001.conf'
Issuing nginx reload (running 'sudo nginx -s reload')
[sudo] password for micronets-dev:
```

6. Confirm the Micronets Manager for the subscriber is removed by executing the following command:

```
curl -s https://nccoe-server1.micronets.net/sub/subscriber-001/api/mm/v1/subscriber/subscriber-001
```

5 Build 4 Product Installation Guides

This section of the practice guide contains detailed instructions for installing and configuring the products used to implement Build 4. For additional details on Build 4's logical and physical architectures, please refer to NIST SP 1800-15B.

5.1 NIST SDN Controller/MUD Manager

5.1.1 NIST SDN Controller/MUD Manager Overview

This is a limited implementation that is intended to introduce a MUD manager build on top of an SDN controller. Build 4 implements all the abstractions in the MUD specification. At testing, this build uses strictly IPv4, and DHCP is the only standardized mechanism that it supports to associate MUD URLs with devices.

Build 4 uses a MUD manager built on the OpenDaylight SDN controller. This build works with IoT devices that emit their MUD URLs through DHCP. The MUD manager works by snooping the traffic passing through the controller to detect the emission of a MUD URL. The MUD URL extracted by the MUD manager is then used to retrieve the MUD file and corresponding signature file associated with the MUD URL. The signature file is used to verify the legitimacy of the MUD file. The MUD manager then translates the access control entries in the MUD file into flow rules that are pushed to the switch.

5.1.2 Configuration Overview

The following subsections document the software, hardware, and network configurations for the Build 4 SDN controller/MUD manager.

5.1.2.1 Hardware Configuration

This build requires installing the SDN controller/MUD manager on a server with at least two gigabytes of random access memory. This server must connect to at least one SDN-capable switch or router on the network, which is the MUD policy enforcement point. The MUD manager works with any OpenFlow 1.3-enabled SDN switch. For this implementation, a Northbound Networks Zodiac WX wireless SDN access point was used as the SDN switch.

5.1.2.2 Network Configuration

The SDN controller/MUD manager instance was installed and configured on a dedicated machine leveraged for hosting virtual machines in the Build 4 lab environment. The SDN controller/MUD manager listens on port 6653 for Open vSwitch (OVS) inbound connections, which are initiated by the OVS instance running on the Northbound Networks access point.

5.1.2.3 Software Configuration

For this build, the SDN controller/MUD manager was installed on an Ubuntu 18.04.01 64-bit server.

The SDN controller/MUD manager requires the following installations and components:

- Java SE Development Kit 8
- Apache Maven 3.5 or higher

5.1.3 Preinstallation

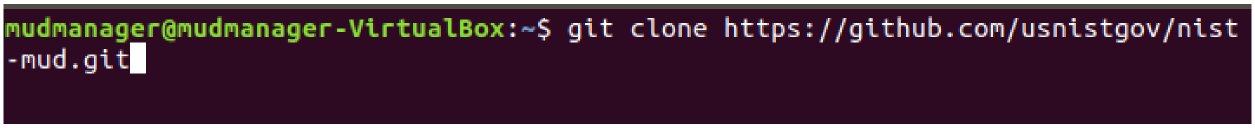
Build 4's GitHub page provides documentation that was followed to complete this section: https://github.com/usnistgov/nist-mud.

- Install JDK 1.8: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.
- Install Maven 3.5 or higher: https://maven.apache.org/download.cgi.

5.1.4 Setup

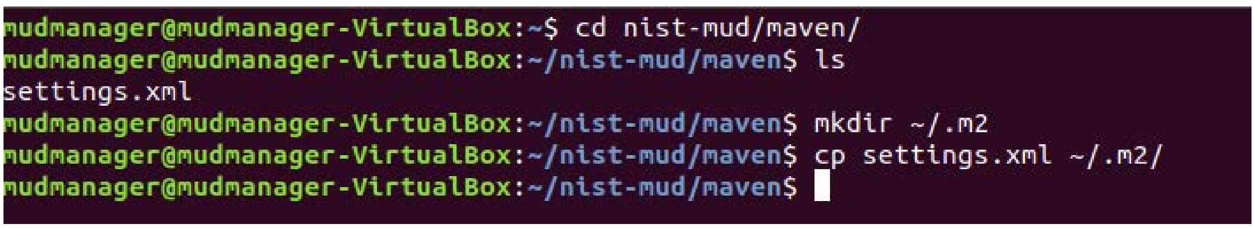
1. Execute the following command to clone the Git project:

```
git clone https://github.com/usnistgov/nist-mud.git
```



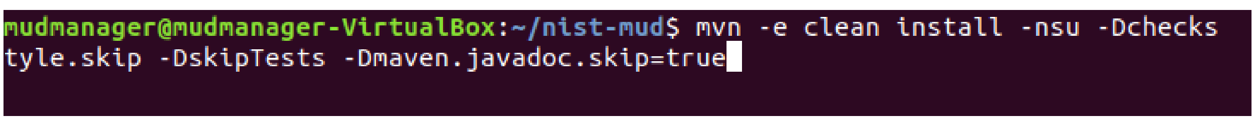
2. Copy the contents of `nist-mud/maven/settings.xml` to `~/.m2` by executing the commands below:

```
cd nist-mud/maven/
mkdir ~/.m2
cp settings.xml ~/.m2
```



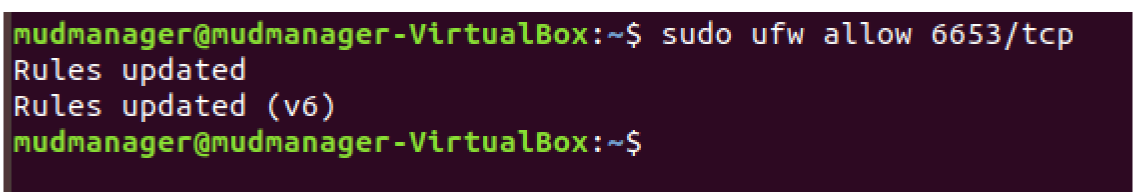
3. In the nist-mud directory, run the commands below:

```
cd
cd nist-mud/
mvn -e clean install -nsu -Dcheckstyle.skip -DskipTests -
Dmaven.javadoc.skip=true
```



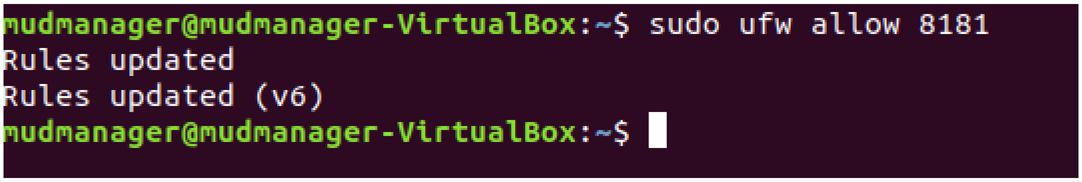
4. Open port 6653 on the controller stack for TCP access so the switches can connect by executing the command below:

```
sudo ufw allow 6653/tcp
```



5. OpenDaylight uses port 8181 for the REST API. That port should be opened if access to the REST API is desired from outside the controller machine. Open port 8181 by executing the command below:

```
sudo ufw allow 8181
```

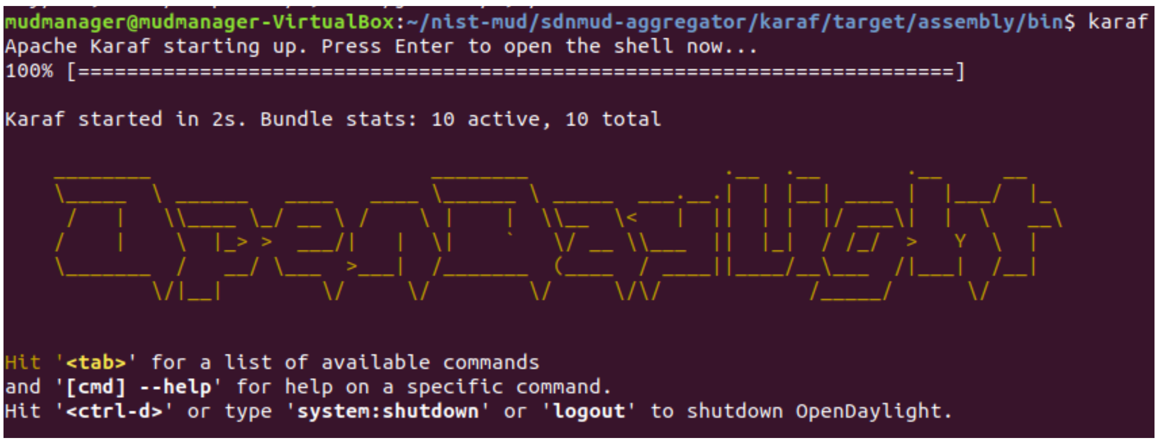


6. Change to the bin directory by executing the command below:

   `~/nist-mud/sdnmud-aggregator/karaf/target/assembly/bin`

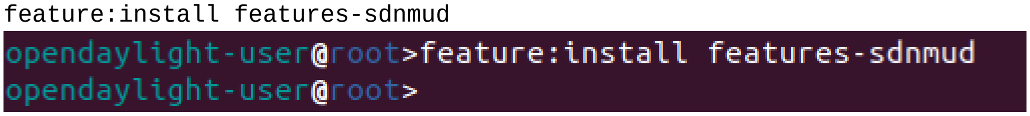
7. Run the command below:

   `./karaf clean`

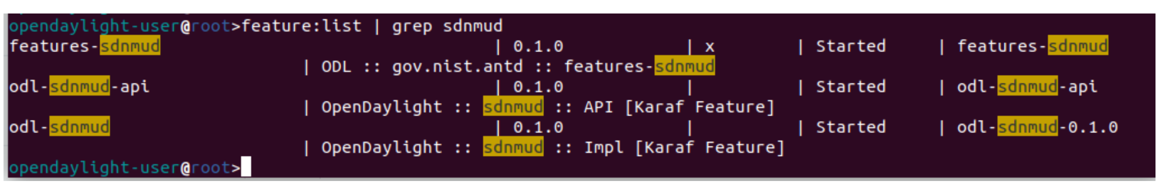


8. At the Karaf prompt, install MUD capabilities using:

   `feature:install features-sdnmud`

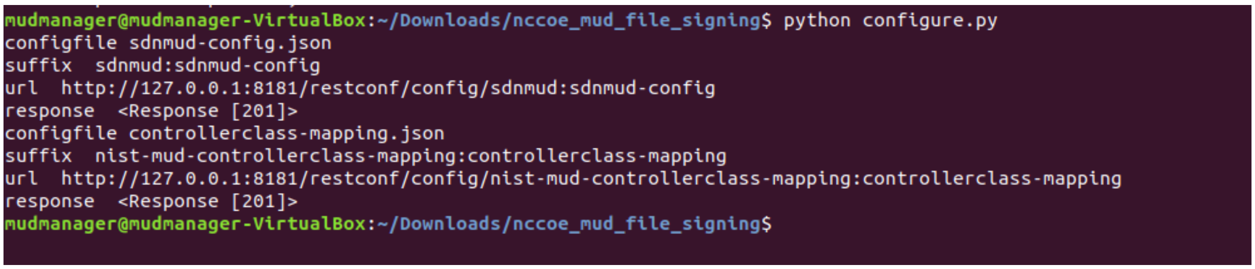


9. Check if the feature is running by using the command `feature:list | grep sdnmud` in Karaf.



10. On the SDN controller/MUD manager host, run a script to configure the SDN controller and add bindings for the controller abstractions defined in the test MUD files. This script pushes configuration information for the MUD manager application (`sdnmud-config.json`) as well as network configuration information for the managed local area network (LAN) (`controllerclass-mapping.json`). The latter file specifies bindings for the controller classes that are used in the MUD

file as well as subnet information for classification of local addresses. These are scoped to a single policy enforcement point, which is identified by a switch-id. By default, the switch ID is `openflow:MAC-address` where `MAC-address` is the MAC address of the switch interface that connects to the SDN controller (in decimal). This must be unique per switch. Note too, that we identify whether a switch is wireless.



Example Python script (`configure.py`):

```
import requests
import json
import argparse
import os

if __name__=="__main__":
   if os.environ.get("CONTROLLER_ADDR") is None:
      print "Please set environment variable CONTROLLER_ADDR to the address of the
opendaylight controller"

   controller_addr = os.environ.get("CONTROLLER_ADDR")

   headers= {"Content-Type":"application/json"}
   for (configfile,suffix) in {
       ("sdnmud-config.json", "sdnmud:sdnmud-config"),
       ("controllerclass-mapping.json","nist-mud-controllerclass-
mapping:controllerclass-mapping") }:
       data = json.load(open(configfile))
       print "configfile", configfile
       print "suffix ", suffix
       url = "http://" + controller_addr + ":8181/restconf/config/" + suffix
       print "url ", url
       r = requests.put(url, data=json.dumps(data), headers=headers , auth=('admin',
'admin'))
       print "response ", r
```

Example controller class mapping (`controllerclass-mapping.json`):

```
{
"controllerclass-mapping" :  {
   "switch-id" : "openflow:123917682138002",
   "controller" : [
       {
             "uri" :  "urn:ietf:params:mud:dns",
             "address-list" : [ "10.0.41.1" ]
```

```
        },
        {
                "uri" :  "urn:ietf:params:mud:dhcp",
                "address-list" : [ "10.0.41.1" ]
        },
        {
                "uri" :  "https://controller.nist.local",
                "address-list" : [ "10.0.41.225" ]
        },
        {
                "uri" :  "https://sensor.nist.local/nistmud1",
                "address-list" : [ "10.0.41.225" ]
        }
   ],
   "local-networks": [ "10.0.41.0/24" ],
   "wireless" : true
 }
}
```

Example SDN MUD configuration (`sdnmud-config.json`):

```
{
 "sdnmud-config" : {
        "ca-certs": "lib/security/cacerts",
        "key-pass" : "changeit",
        "trust-self-signed-cert" : true,
        "mfg-id-rule-cache-timeout": 120,
        "relaxed-acl" : false
  }
}
```

5.2 MUD File Server

5.2.1 MUD File Sever Overview

The MUD file server is responsible for serving the MUD file and the corresponding signature file upon request from the MUD manager. For testing purposes, the MUD file server is run on 127.0.0.1 on the same machine as the MUD manager. This allows us to examine the logs to check if the MUD file has been retrieved. For testing purposes, host name verification for the TLS connection to the MUD file server is disabled in the configuration of the MUD manager.

5.2.2 Configuration Overview

The following subsections document the software, hardware, and network configurations for the MUD file server.

5.2.2.1 Hardware Configuration

The MUD file server was hosted on the same machine as the SDN controller.

5.2.2.2 Network Configuration

The MUD file server was hosted on the same machine as the SDN controller. To direct the MUD manager to retrieve the MUD files from the MUD file server, the host name of the two manufacturers that are present in the MUD URLs used for testing are both mapped to 127.0.0.1 in the `/etc/hosts` file of the Java Virtual Machine in which the MUD manager is running. This static configuration is read by the MUD manager when it starts. The name resolution information in the `/etc/hosts` file directs the MUD manager to retrieve the test MUD files from the MUD file server.

5.2.2.3 Software Configuration

In this build, serving MUD files requires Python 2.7 and the Python requests package. These may be installed using *apt* and *pip*. After creation of the MUD files by using mudmaker.org, the MUD files were signed, and the certificates used for signing were imported into the trust store of the Java Virtual Machine in which the MUD manager is running.

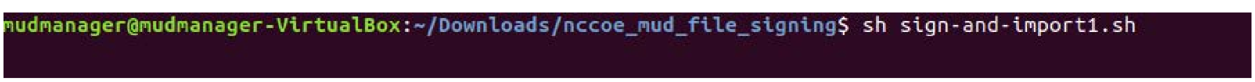
5.2.3 Setup

5.2.3.1 MUD File Creation

This build also leveraged the MUD Maker online tool found at www.mudmaker.org. For detailed instructions on creating a MUD file using this online tool, please refer to Build 1's [MUD File Creation](#) section.

5.2.3.2 MUD File Signing

1. Sign and import the desired MUD files. An example script (`sign-and-import1.sh`) can be found below.



The shell script that was used in this build is shown below. This script generates a signature based on the private key of a DigiCert-issued certificate and imports the certificate into the trust store of the Java Virtual Machine. This is done for both MUD files.

```
CACERT=DigiCertCA.crt
MANUFACTURER_CRT=nccoe_mud_file_signing.crt
MANUFACTURER_KEY=mudsign.key.pem
MANUFACTURER_ALIAS=sensor.nist.local
MANUFACTURER_SIGNATURE=mudfile-sensor.p7s
MUDFILE=mudfile-sensor.json

openssl cms -sign -signer $MANUFACTURER_CRT -inkey $MANUFACTURER_KEY -in $MUDFILE -
binary -noattr -outform DER -certfile $CACERT  -out $MANUFACTURER_SIGNATURE
openssl cms -verify -binary  -in $MANUFACTURER_SIGNATURE  -signer $MANUFACTURER_CRT -
inform DER  -content $MUDFILE
```

```
MANUFACTURER_ALIAS=otherman.nist.local
MUDFILE=mudfile-otherman.json
MANUFACTURER_SIGNATURE=mudfile-otherman.p7s
openssl cms -sign -signer $MANUFACTURER_CRT -inkey $MANUFACTURER_KEY -in $MUDFILE -
binary -noattr -outform DER -certfile $CACERT  -out $MANUFACTURER_SIGNATURE
openssl cms -verify -binary  -in $MANUFACTURER_SIGNATURE  -signer $MANUFACTURER_CRT -
inform DER  -content $MUDFILE

sudo -E $JAVA_HOME/bin/keytool -delete -alias digicert -keystore
$JAVA_HOME/jre/lib/security/cacerts -storepass changeit
sudo -E $JAVA_HOME/bin/keytool -importcert -file $CACERT -alias digicert -keystore
$JAVA_HOME/jre/lib/security/cacerts -storepass changeit
```

5.2.3.3 MUD File Serving

Run a script that serves desired MUD files and signatures. An example Python script (`mudfile-server.py`) can be found below.

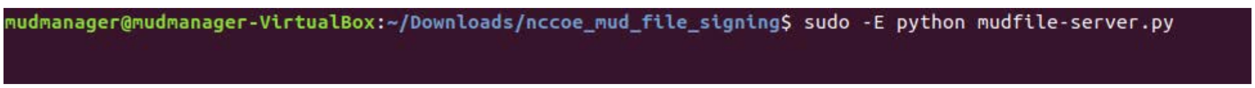
1. Save a copy of the **mudfile-server.py** Python script onto the NIST SDN controller/MUD manager configured in Section 5.1:

```
import BaseHTTPServer, SimpleHTTPServer
import ssl
import urlparse
# Dummy manufacturer server for testing

class MyHTTPRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        print ("DoGET " + self.path)
        self.send_response(200)
        if self.path == "/nistmud1" :
           with  open("mudfile-sensor.json", mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        elif self.path == "/nistmud2" :
           with  open("mudfile-otherman.json", mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        elif self.path == "/nistmud1/mudfile-sensor.p7s":
           with open("mudfile-sensor.p7s",mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        elif self.path == "/nistmud2/mudfile-otherman.p7s":
           with open("mudfile-otherman.p7s",mode="r") as f:
                data = f.read()
```

```
            print("Read " + str(len(data)) + " chars ")
                  self.send_header("Content-Length", len(data))
                  self.end_headers()
                  self.wfile.write(data)
          else:
             print("UNKNOWN URL!!")
             self.wfile.write(b'Hello, world!')

  httpd = BaseHTTPServer.HTTPServer(('0.0.0.0', 443), MyHTTPRequestHandler)
  httpd.socket = ssl.wrap_socket (httpd.socket, keyfile='./mudsigner.key',
  certfile='./mudsigner.crt', server_side=True)
  httpd.serve_forever()
```

2. From the same directory as the previous step, execute the command below to start the MUD file server:

```
sudo -E python mudfile-server.py
```



5.3 Northbound Networks Zodiac WX Access Point

5.3.1 Northbound Networks Zodiac WX Access Point Overview

The Zodiac WX, in addition to being a wireless access point, includes the following logical components: an SDN switch, a NAT router, a DHCP server, and a DNS server. The Zodiac WX is powered by OpenWRT and Open vSwitch. Open vSwitch directly integrates into the wireless configuration. The Zodiac WX works with any standard OpenFlow-compatible controllers and requires no modifications because it appears to the controller as a standard OpenFlow switch.

5.3.2 Configuration Overview

The following subsections document the network, software, and hardware configurations for the SDN-capable Northbound Networks Zodiac WX.

5.3.2.1 Network Configuration

The access point is configured to have a static public address on the public side of the NAT. For purposes of testing, we use 203.0.113.x addresses on the public network. The public side of the NAT is given the address of 203.0.113.1. The DHCP server is set up to allocate addresses to wireless devices on the LAN. The SDN controller/MUD manager is connected to the public side of the NAT. The Open vSwitch configuration for the access point is given the address of the SDN controller, which is shown in the setup below.

5.3.2.2 Software Configuration

At this implementation, no additional software configuration was required.

5.3.2.3 Hardware Configuration

At this implementation, no additional hardware configuration was required.

5.3.3 Setup

On the Zodiac WX, DNSmasq supports both DHCP and DNS. For testing purposes, it will be necessary to access several web servers (two update servers called www.nist.local and an unapproved server called www.antd.local). The following commands enable the Zodiac WX to resolve the web server host names to their IP addresses.

1. Set up the access point to resolve the addresses for the web server host names by opening the file */etc/dnsmasq.conf* on the access point.
2. Add the following line to the *dnsmasq.conf* file:

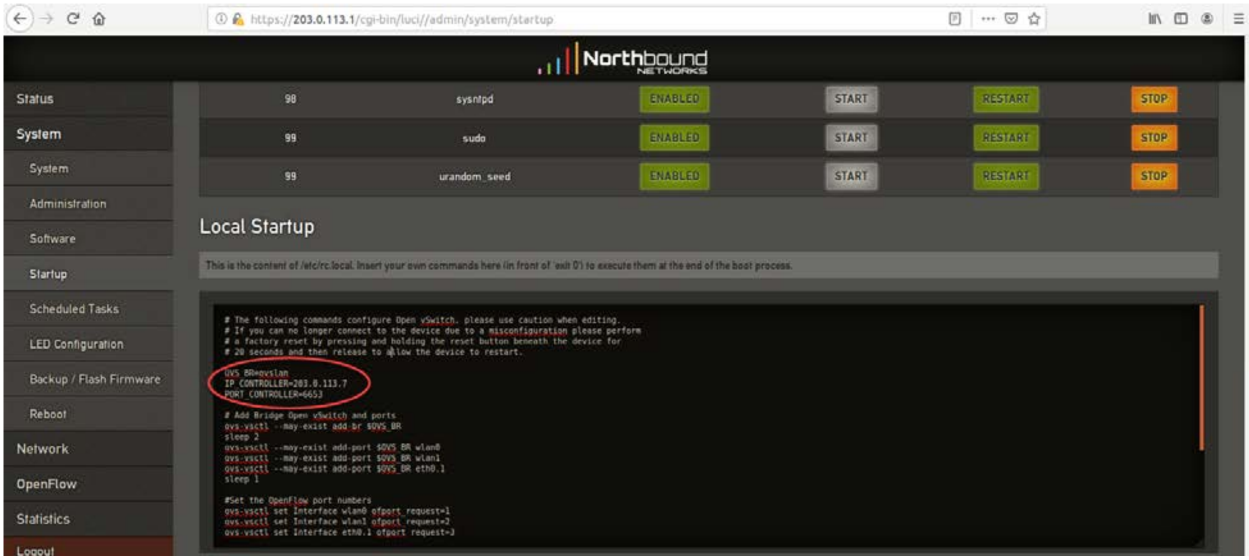
   `addn-hosts=/etc/hosts.nist.local`

   ```
   addn-hosts=/etc/hosts.nist.local
   - /etc/dnsmasq.conf [Readonly] 38/38 100%
   ```

3. The file */etc/hosts.nist.local* has the host name to address mapping. The mapping used for our tests is shown below (Note that the host www.nist.local maps to two addresses on the public side).

   ```
   203.0.113.13 www.nist.local
   203.0.113.15 www.nist.local
   203.0.113.14 www.antd.local
   ~
   ```

4. On the Zodiac WX configuration web page in the System->Startup tab, indicate where (IP address and port) the Open vSwitch Daemon connects to the controller.



5.4 DigiCert Certificates

DigiCert's CertCentral web-based platform allows provisioning and management of publicly trusted X.509 certificates for a variety of purposes. After establishing an account, clients can log in, request, renew, and revoke certificates by using only a browser. For Build 4, the Premium Certificate created in Build 1 was leveraged for signing the MUD files. To request and implement DigiCert certificates, follow the documentation in Build 1's DigiCert Certificates section and subsequent sections.

5.5 IoT Devices

5.5.1 IoT Devices Overview

This section provides configuration details for the Linux-based Raspberry Pis used in the build, which emit MUD URLs by using DHCP.

5.5.2 Configuration Overview

The devices used in this build were multiple Raspberry Pi development kits that were configured to act as IoT devices. The devices run Raspbian 9, a Linux-based operating system, and are configured to emit a MUD URL during a typical DHCP transaction. These devices were used to test interactions related to MUD capabilities.

5.5.2.1 Network Configuration

The kits are connected to the network over a wireless connection. Their IP addresses are assigned dynamically by the DHCP server on the Zodiac WX access point.

5.5.2.2 Software Configuration

The Raspberry Pis are configured on Raspbian. They also utilized dhclient as their default DHCP clients to manually initiate a DHCP interaction. This DHCP client is installed natively on many Linux distributions and can be installed using a preferred package manager if not currently present. Dhclient uses a configuration file: */etc/dhclient.conf*. This needs to be modified to include the MUD URL that the device will emit in its DHCP requests. (The modification details are provided in the setup information below.)

5.5.2.3 Hardware Configuration

Multiple Raspberry Pi 3 Model B devices were used.

5.5.3 Setup

Each Raspberry Pi used in this build was intended to represent a different class of device (manufacturer, other manufacturer, local networks, controller classes). The type of device was determined by the MUD URL being emitted by the device. If no MUD URL is emitted, the device is an unclassified local network device.

1. On each Pi, changes were made to `/etc/network/interfaces` to add a line that allows the Pi to authenticate to the access point. The following line is added to the network interface as shown below:

   `wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf.northbound`

```
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf.northbound
```

   The file (*/etc/wpa_supplicant/wpa_supplicant.conf.northbound*) is shown below:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
        ssid="ZodiacWX_24GHz"
        psk="66666666"
}
```

2. A dhclient configuration file can be altered (by adding information) to allow for emission of a MUD URL in the DHCP transaction. Modify the *dhclient.conf* file with the command:

```
vi /etc/dhcp/dhclient.conf
```

3. A send MUD URL line must be added as well as a `mud-url` in the request line. In this build, multiple MUD URLs were transmitted, depending on the type of the device. Example alterations made to dhclient configuration files can be seen below:

```
send mud-url = "https://sensor.nist.local/nistmud1";
send mud-url = "https://otherman.nist.local/nistmud2";
```

```
send mud-url = "https://sensor.nist.local/nistmud1";

request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, domain-search, host-name, mud-url,
        dhcp6.name-servers, dhcp6.domain-search,
        netbios-name-servers, netbios-scope, interface-mtu,
        rfc3442-classless-static-routes, ntp-servers,
        dhcp6.fqdn, dhcp6.sntp-servers;
```

4. To control the time at which the MUD URL is emitted, we manually reacquire the DHCP address rather than have the device acquire the MUD URL on boot. Emit the MUD URL and attain an IP address by sending the altered dhclient configuration file manually with the following commands:

```
sudo rm /var/lib/dhcp/dhclient.leases
sudo ifconfig wlan0 0.0.0.0
sudo dhclient -v wlan0 -cf /etc/dhcp/dhclient.conf.toaster
```

```
 sensor ] sudo rm /var/lib/dhcp/dhclient.leases; sudo ifconfig wlan0 0.0.0.0;  sudo dhclient -v wlan0 -cf /etc/dhcp/dhclient.conf.toaster
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/wlan0/b8:27:eb:3d:65:78
Sending on   LPF/wlan0/b8:27:eb:3d:65:78
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 4
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 10
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 11
DHCPREQUEST of 10.0.41.190 on wlan0 to 255.255.255.255 port 67
DHCPOFFER of 10.0.41.190 from 10.0.41.1
DHCPACK of 10.0.41.190 from 10.0.41.1
bound to 10.0.41.190 -- renewal in 21068 seconds.
 sensor ]
```

5.6 Update Server

5.6.1 Update Server Overview

This section provides configuration details for the Linux-based IoT development kit used in the build, which acts as an update server. This update server will attempt to access and be accessed by the IoT device, which, in this case, is one of the development kits built in the lab. The update server is a web server that hosts mock software update files to be served as software updates to our IoT device devkits. When the server receives an http request, it sends the corresponding update file.

5.6.2 Configuration Overview

The devkit runs Raspbian 9, a Linux-based operating system, and is configured to act as an update server. This host was used to test approved internet interactions related to MUD capabilities.

5.6.2.1 Network Configuration

The web server host has a static public IP address configuration and is connected to the access point on the wired interface. It is given an address on the 203.0.113 network.

5.6.2.2 Software Configuration

The Raspberry Pi is configured on Raspbian. The devkit also utilized a simple Python script to run an http server to test MUD capabilities.

5.6.2.3 Hardware Configuration

The hardware used for this devkit includes a Raspberry Pi 3 Model B.

5.6.3 Setup

The primary configuration needed for the web server device is done with the DNS mapping on the Zodiac WX access point to be discussed in the section related to setup of the Northbound Networks Zodiac WX Access Point. The Raspberry Pi is required to run a simple http server.

1. Copy the example Python script below onto the Raspberry Pi:

   Example Python script (`httpserver.py`):

   ```
   import SimpleHTTPServer
   import SocketServer
   import argparse
   if __name__ == "__main__":
      parser = argparse.ArgumentParser()
      parser.add_argument("-H", help="Host address", default="0.0.0.0")
      parser.add_argument("-P", help="Port ", default="80")
      args = parser.parse_args()
      hostAddr = args.H
      PORT = int(args.P)
      Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
      httpd = SocketServer.TCPServer((hostAddr, PORT), Handler)
      print "serving at port", PORT
      httpd.serve_forever()
   ```

2. From the same directory as the script copied in the previous step, execute the command below to start the http server:

```
sudo python httpserver.py -P 443
```

```
www.nist.local ] sudo python httpserver.py -P 443
serving at port 443
```

5.7 Unapproved Server

5.7.1 Unapproved Server Overview

This section provides configuration details for the Linux-based IoT development kit used in the build, which acts as an unapproved internet host. This host will attempt to access and to be accessed by an IoT device, which, in this case, is one of the MUD-capable devices on the network.

The unapproved server is an internet host that is not explicitly authorized in the MUD file to communicate with the IoT device. When the IoT device attempts to connect to this server, the switch should not allow this traffic because it is not an approved internet service per the corresponding MUD file. Likewise, when the server attempts to connect to the IoT device, this traffic should be denied at the switch.

5.7.2 Configuration Overview

The devkit runs Raspbian 9, a Linux-based operating system, and is configured to act as an unapproved internet host. This host was used to test unapproved internet interactions related to MUD capabilities.

5.7.2.1 Network Configuration

The web host has a static public IP address configuration and is connected to the access point on the wired interface. It is given an address on the 203.0.113 network.

5.7.2.2 Software Configuration

The Raspberry Pi is configured on Raspbian. The devkit also utilized a simple Python script to run an http server to test MUD capabilities.

5.7.2.3 Hardware Configuration

The hardware used for this devkit includes a Raspberry Pi 3 Model B.

5.7.3 Setup

The primary configuration needed for the web server device is accomplished by the DNS mapping on the Zodiac WX access point to be discussed in the section related to setup of the Northbound Networks Zodiac WX Access Point. The Raspberry Pi is required to run a simple http server.

1. Copy the example Python script below onto the Raspberry Pi:

Example Python script (`httpserver.py`):

```
import SimpleHTTPServer
import SocketServer
import argparse
if __name__ == "__main__":
   parser = argparse.ArgumentParser()
   parser.add_argument("-H", help="Host address", default="0.0.0.0")
   parser.add_argument("-P", help="Port ", default="80")
   args = parser.parse_args()
   hostAddr = args.H
   PORT = int(args.P)
   Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
   httpd = SocketServer.TCPServer((hostAddr, PORT), Handler)
   print "serving at port", PORT
   httpd.serve_forever()
```

2. From the same directory as the script copied in the previous step, execute the command below to start the http server:

```
sudo python httpserver.py -P 443
```

```
www.nist.local ] sudo python httpserver.py -P 443
serving at port 443
```

Appendix A List of Acronyms

AAA	Authentication, Authorization, and Accounting
ACL	Access Control List
API	Application Programming Interface
CMS	Cryptographic Message Syntax
COA	Change of Authorization
CRADA	Cooperative Research and Development Agreement
DB	Database
DDoS	Distributed Denial of Service
Devkit	Development Kit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
GCA	Global Cyber Alliance
http	Hypertext Transfer Protocol
https	Hypertext Transfer Protocol Secure
IOS	Cisco's Internetwork Operating System
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IT	Information Technology
ITL	NIST's Information Technology Laboratory
JSON	JavaScript Object Notation
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LED	Light-Emitting Diode

LLDP	Link Layer Discovery Protocol (**Institute of Electrical and Electronics Engineers** 802.1AB)
MAB	MAC Authentication Bypass
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
MUD	Manufacturer Usage Description
NAS	Network Access Server
NAT	Network Address Translation
NCCoE	National Cybersecurity Center of Excellence
NIST	National Institute of Standards and Technology
OS	Operating System
PoE	Power over Ethernet
RADIUS	Remote Authentication Dial-In User Service
REST	Representational State Transfer
RFC	Request for Comments
SDN	Software-Defined Networking
SP	Special Publication
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator
Vi	Visual
VLAN	Virtual Local Area Network

VNC	Virtual Network Computing
WAN	Wide Area Network

Appendix B Glossary

Audit Independent review and examination of records and activities to assess the adequacy of system controls to ensure compliance with established policies and operational procedures (National Institute of Standards and Technology [NIST] Special Publication [SP] 800-12 Rev. 1)

Best Practice A procedure that has been shown by research and experience to produce optimal results and that is established or proposed as a standard suitable for widespread adoption (Merriam-Webster)

Botnet The word “botnet” is formed from the words “robot” and “network.” Cybercriminals use special Trojan viruses to breach the security of several users’ computers, take control of each computer, and organise all of the infected machines into a network of “bots” that the criminal can remotely manage. (https://usa.kaspersky.com/resource-center/threats/botnet-attacks)

Control A measure that is modifying risk (Note: Controls include any process, policy, device, practice, or other actions that modify risk.) (NIST Interagency or Internal Report 8053)

Denial of Service The prevention of authorized access to a system resource or the delaying of system operations and functions (NIST SP 800-82 Rev. 2)

Distributed Denial of Service (DDoS) A denial of service technique that uses numerous hosts to perform the attack (NIST Interagency or Internal Report 7711)

Managed Devices Personal computers, laptops, mobile devices, virtual machines, and infrastructure components require management agents, allowing information technology staff to discover, maintain, and control these devices. Those with broken or missing agents cannot be seen or managed by agent-based security products.

Manufacturer Usage Description (MUD) A component-based architecture specified in Request for Comments (RFC) 8250 that is designed to provide a means for end devices to signal to the network what sort of access and network functionality they require to properly function

Mapping Depiction of how data from one information source maps to data from another information source

Mitigate	To make less severe or painful or to cause to become less harsh or hostile (Merriam-Webster)
MUD-Capable	An IoT device that is capable of emitting a MUD uniform resource locator (URL) in compliance with the MUD specification
Network Address Translation (NAT)	A function by which internet protocol (IP) addresses within a packet are replaced with different IP addresses. This function is most commonly performed by either **routers** or firewalls. It enables private IP networks that **use** unregistered IP addresses to connect to the internet. **NAT** operates on a router, usually connecting two networks together, and translates the private (not globally unique) addresses in the internal network into legal addresses before packets are forwarded to another network.
Non-MUD-Capable	An IoT device that is not capable of emitting a MUD URL in compliance with the MUD specification (RFC 8250)
Policy	Statements, rules, or assertions that specify the correct or expected behavior of an entity. For example, an authorization policy might specify the correct access control rules for a software component. (NIST SP 800-95 and NIST Interagency or Internal Report 7621 Rev. 1)
Policy Enforcement Point	A network device on which policy decisions are carried out or enforced
Risk	The net negative impact of the exercise of a vulnerability, considering both the probability and the impact of occurrence. Risk management is the process of identifying risk, assessing risk, and taking steps to reduce risk to an acceptable level. (NIST SP 800-30)
Router	A computer that is a gateway between two networks at open systems interconnection layer 3 and that relays and directs data packets through that internetwork. The most common form of router operates on IP packets. (NIST SP 800-82 Rev. 2)
Security Control	A safeguard or countermeasure prescribed for an information system or an organization, which is designed to protect the confidentiality, integrity, and availability of its information and to meet a set of defined security requirements (NIST SP 800-53 Rev. 4)

Server	A computer or device on a network that manages network resources. Examples are file servers (to store files), print servers (to manage one or more printers), network servers (to manage network traffic), and database servers (to process database queries). (NIST SP 800-47)
Shall	A requirement that must be met unless a justification of why it cannot be met is given and accepted (NIST Interagency or Internal Report 5153)
Should	This term is used to indicate an important recommendation. Ignoring the recommendation could result in undesirable results. (NIST SP 800-108)
Threat	Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. Also, the potential for a threat source to successfully exploit a particular information system vulnerability (Federal Information Processing Standards 200)
Threat Signaling	Real-time signaling of DDoS-related telemetry and threat-handling requests and data between elements concerned with DDoS attack detection, classification, traceback, and mitigation (https://joinup.ec.europa.eu/collection/rolling-plan-ict-standardisation/cybersecurity-network-and-information-security)
Traffic Filter	An entry in an access control list that is installed on the router or switch to enforce access controls on the network
Uniform Resource Locator (URL)	A reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A typical URL could have the form http://www.example.com/index.html, which indicates a protocol (hypertext transfer protocol [http]), a host name (www.example.com), and a file name *(index.html)*. Also sometimes referred to as a *web address*
Update	New, improved, or fixed software, which replaces older versions of the same software. For example, updating an OS brings it up-to-date with the latest drivers, system utilities, and security software. Updates are often provided by the software publisher free of charge. (https://www.computerhope.com/jargon/u/update.htm)
Update Server	A server that provides patches and other software updates to Internet of Things devices

Virtual Local Area Network (VLAN)	A broadcast domain that is partitioned and isolated within a network at the data link layer. A single physical local area network (LAN) can be logically partitioned into multiple, independent VLANs; a group of devices on one or more physical LANs can be configured to communicate within the same VLAN as if they were attached to the same physical LAN.
Vulnerability	Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source (NIST SP 800-37 Rev. 2)

Appendix C Bibliography

Request for Comments (RFC) 8520. (2019, Mar.) “Manufacturer Usage Description Specification” [Online]. Available: https://tools.ietf.org/html/rfc8520.

Cisco’s developer MUD Manager GitHub page [Website]. Available: https://github.com/CiscoDevNet/MUD-Manager/tree/1.0#dependancies.

Apache HTTP Server Project documentation, Version 2.4. Compiling and Installing Apache [Website]. Available: https://httpd.apache.org/docs/current/install.html.

Apache HTTP Server Project documentation, Version 2.4. Apache SSL/TLS Encryption [Website]. Available: https://httpd.apache.org/docs/current/ssl/ssl_howto.html.

Welcome to MUD File maker! [Website]. Available: https://www.mudmaker.org/.

DigiCert. Advanced CertCentral Getting Started Guide, Version 9.2 [Website]. Available: https://www.digicert.com/certcentral-support/digicert-getting-started-guide.pdf.

DigiCert. SSL Certificate Support [Website]. Available: https://www.digicert.com/security-certificate-support/.

DigiCert. Order your SSL/TLS certificates [Website]. Available: https://docs.digicert.com/manage-certificates/order-your-ssltls-certificates/.

DigiCert. CertCentral Client Certificate Guide, Version 1.9 [Website]. Available: https://www.digicert.com/certcentral-support/client-certificate-guide.pdf.

Forescout. ForeScout CounterAct® Installation Guide, Version 8.0.1 [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/10/CounterACT_Installation_Guide_8.0.1.pdf.

Forescout. (2018, Feb.) ForeScout CounterAct Device Profile Library Configuration Guide [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_Device_Profile_Library.pdf.

Forescout. (2018, Feb.) ForeScout CounterAct IoT Posture Assessment Library Configuration Guide [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_IoT_Posture_Assessment_Library-1.pdf.

Forescout. ForeScout CounterAct Open Integration Module Overview Guide, Version 1.1 [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/08/CounterACT_Open_Integration_Module_Overview_1.1.pdf.

Forescout. (2018, Feb.) ForeScout CounterAct Windows Applications Configuration Guide [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_Windows_Applications.pdf.

Forescout. (2018, Feb.) ForeScout CounterAct Windows Vulnerability DB Configuration Guide [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/04/CounterACT_Windows_Vulnerability_DB_18.0.2.pdf.

Forescout. HPS NIC Vendor DB Configuration Guide, Version 1.2.4 [Website]. Available: https://www.Forescout.com/wp-content/uploads/2018/04/HPS_NIC_Vendor_DB_1.2.4.pdf.

DRAFT

Securing Small-Business and Home Internet of Things (IoT) Devices:

Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)

Functional Demonstration Results
Supplement to NIST Special Publication 1800-15B

Mudumbai Ranganathan
NIST

William C. Barker
Dakota Consulting

Drew Cohen
Kevin Yeich
MasterPeace Solutions, Ltd.

Steve Johnson
Ashwini Kadam
Craig Pratt
Darshak Thakore
CableLabs

Eliot Lear
Cisco

Adnan Baykal
Global Cyber Alliance

Yemi Fashina
Parisa Grayeli
Joshua Harrington
Joshua Klosterman
Blaine Mulugeta
Susan Symington
The MITRE Corporation

September 2020

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos

National Institute of Standards and Technology
U.S. Department of Commerce

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

Contents

List of Tables

1 Introduction

The National Institute of Standards and Technology (NIST) Cybersecurity Practice Guide explains how the [Manufacturer Usage Description (MUD) Specification (Internet Engineering Task Force [IETF] Request for Comments [RFC] 8520)] can be used to reduce the vulnerability of Internet of Things (IoT) devices to botnets and other network-based threats as well as reduce the potential for harm from exploited IoT devices. It describes the logical architecture of a standards-based reference design for using MUD, threat signaling, and employing software updates to significantly increase the effort required by malicious actors to compromise and exploit IoT devices on a home or small-business network. It provides users with the information they need to replicate deployment of the MUD protocol to mitigate IoT-based distributed denial of service (DDoS) threats. The guide contains three volumes and a supplement:

- NIST SP 1800-15A: *Executive Summary – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge*.
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics – what we built and why, including the risk analysis performed, and the security control map*.
- NIST SP 1800-15C: *How-To Guides – instructions for building the example implementations including all the security-relevant details that would allow you to replicate all or parts of this project*.

This document, *Functional Demonstration Results,* is a supplement to NIST SP 1800-15B, *Approach, Architecture, and Security Characteristics*. The document describes the functional demonstration results for four implementations of the reference design that were demonstrated as part of this National Cybersecurity Center of Excellence (NCCoE) project. These implementations are referred to as *builds*:

- Build 1 uses equipment from Cisco Systems and Forescout. The Cisco MUD Manager is used to provide support for MUD, and the Forescout Virtual Appliances and Enterprise Manager are used to perform non-MUD-related device discovery on the network.
- Build 2 uses equipment from MasterPeace Solutions Ltd., Global Cyber Alliance (GCA), and ThreatSTOP. The MasterPeace Solutions Yikes! router, cloud service, and mobile application are used to support MUD, as well as to perform device discovery on the network and to apply additional traffic rules to both MUD-capable and non-MUD-capable devices based on device manufacturer and model. The GCA Quad9 DNS Service and the ThreatSTOP Threat MUD File Server are used to support threat signaling.
- Build 3 uses equipment from CableLabs. CableLabs Micronets (e.g., Micronets Gateway, Micronets Manager, Micronets mobile phone application, and related service provider cloud-based infrastructure) supports MUD and implements the Wi-Fi Alliance's Wi-Fi Easy Connect protocol to securely onboard devices to the network. It also uses software-defined networking

to create separate trust zones (e.g., network segments) called "micronets" to which devices are assigned according to their intended network function.

- Build 4 uses software developed at the NIST Advanced Networking Technologies laboratory. This software serves as a working prototype for demonstrating the feasibility and scalability characteristics of the MUD RFC.

For a more comprehensive description of each build and a detailed explanation of each build's architecture and technologies, refer to NIST SP 1800-15B.

1.1 Objective

This document, *Functional Demonstration Results*, reports the results of the functional evaluation and demonstration of Builds 1, 2, 3, and 4. For each of these builds, we defined a list of requirements unique to that build and then developed a set of test cases to verify that the build meets those requirements. The requirements, test cases, and test results for each of these four builds are documented below.

1.2 Functional Demonstration Activities

All builds were tested to determine the extent to which they correctly implement basic functionality defined within the MUD RFC. Builds 1, 2, and 3 were also subjected to additional exercises that were designed to demonstrate non-MUD-related capabilities. These additional exercises were demonstrative rather than evaluative. They did not verify the build's behavior for conformance to a standard or specification; they were designed to demonstrate advertised capabilities of the builds related to their ability to increase device and network security in ways that are independent of the MUD RFC. These additional capabilities may provide security for both non-MUD-capable and MUD-capable devices. Examples of this type of capability are device discovery, identification and classification, support for threat signaling, and secure, automated onboarding of devices using the Wi-Fi Easy Connect protocol.

1.3 Assumptions

The physical architecture of each build as deployed in the NCCoE laboratory environment is depicted and described in NIST SP 1800-15B. Tests for each build were run on the lab architecture documented in NIST SP 1800-15B. Prior to testing each build, all communication paths to the IoT devices on the network were open and could potentially be used to attack systems on the internet. For traffic to be sent between IoT devices, it was required to pass through the router/switch that served as the policy enforcement point (PEP) for the MUD rules.

In the lab setup for each build, the following hosts and web servers were required to be set up and available to support the tests defined below. On the local network where the IoT devices are located, hosts with the following names must exist and be reachable from an IoT device that is plugged into the local network:

- *unnamed-host* (i.e., a local host that is not from the same manufacturer as the IoT device in question and whose MUD Uniform Resource Locator (URL) is not explicitly mentioned in the MUD file of the IoT device as denoting a class of devices with which the IoT device is permitted to communicate. For example, if device A's MUD file says that it may communicate locally with devices that have MUD URLs www.zzz.com and www.xxx.com, then a local host that has a MUD file of www.qqq.com could be *unnamed-host*.)
- *anyhost-to* (i.e., a local host to which the IoT device in question is permitted to initiate communications but not vice versa)
- *anyhost-from* (i.e., a local host that is permitted to initiate communication to the IoT device but not vice versa)
- *same-manufacturer-host* (i.e., a local host that is from the same manufacturer as the IoT device in question. For example, if device A's MUD file is found at URL www.aaa.com and device B's MUD file is also found at URL www.aaa.com, then device B could be *same-manufacturer-host*.)

On the internet (i.e., outside the local network), the following web servers must be set up and reachable from an IoT device that is plugged into the local network:

- https://yes-permit-to.com (i.e., an internet location to which the IoT device in question is permitted to initiate communications but not vice versa)
- https://yes-permit-from.com (i.e., an internet location that is permitted to initiate communications to the IoT device but not vice versa)
- https://unnamed.com (i.e., an internet location with which the IoT device is not permitted to communicate)

We also defined several MUD files for each build (provided in each build section below) that were used to evaluate specific capabilities.

1.4 Document Conventions

For each build, a set of requirements and a corresponding set of functional test cases were defined to verify that the build meets a specific set of requirements that are unique to that build. For evaluating MUD-related capabilities, these requirements are closely aligned to the order of operations in the Manufacturer Usage Description Specification (RFC 8520). However, even for MUD-specific tests, there are tests that are applicable to some builds but not to others, depending on how any given build is implemented.

For each build, the MUD-related requirements for that build are listed in a table. Each of these requirements is associated with two separate tests, one using Internet Protocol version 4 (IPv4) and one using IPv6. At the time of testing, however, IPv6 functionality was not fully supported by any of the builds and so was not evaluated. The names of the tests in which each requirement is tested are listed in the rightmost column of the requirements table for each build. Tests that end with the suffix “v4” are those in which IPv4 addressing is used; tests that end with the suffix “v6” are those in which IPv6 addressing is used. Only the IPv4 versions of each test are listed explicitly in this document. For each test that has both an IPv4 and an IPv6 version, the IPv4 version of the test, IoT-n-v4, is identical to the IPv6 version of the test, IoT-n-v6, except:

- IoT-n-v6 devices are configured to use IPv6, whereas IoT-n-v4 devices are configured to use IPv4.
- IoT-n-v6 devices are configured to use Dynamic Host Configuration Protocol version 6 (DHCPv6), whereas IoT-n-v4 devices are configured to use DHCPv4.
- The IoT-n-v6 DHCPv6 message that is emitted includes the MUD URL option that uses Internet Assigned Numbers Authority (IANA) code 112, whereas the IoT-n-v4 DHCPv4 message that is emitted includes the MUD URL option that uses IANA code 161.

Each test consists of multiple fields that collectively identify the goal of the test, the specifics required to implement the test, and how to assess the results of the test. Table 1-1 describes all test fields.

Table 1-1: Test Case Fields

Test Case Field	Description
Parent Requirement	Identifies the top-level requirement or the series of top-level requirements leading to the testable requirement
Testable Requirement	Guides the definition of the remainder of the test case fields, and specifies the capability to be evaluated
Description	Describes the objective of the test case
Associated Test Case(s)	In some instances, a test case may be based on the outcome of (an)other test case(s). For example, analysis-based test cases produce a result that is verifiable through various means (e.g., log entries, reports, and alerts).
Associated Cybersecurity Framework Subcategory(ies)	Lists the Cybersecurity Framework Subcategories addressed by the test case
IoT Device(s) Under Test	Text identifying which IoT device is being connected to the network in this test

Test Case Field	Description
MUD File(s) Used	Name of MUD file(s) used
Preconditions	Starting state of the test case. Preconditions indicate various starting-state items, such as a specific capability configuration required or specific protocol and content.
Procedure	Step-by-step actions required to implement the test case. A procedure may consist of a single sequence of steps or multiple sequences of steps (with delineation) to indicate variations in the test procedure.
Expected Results	Expected results for each variation in the test procedure
Actual Results	Observed results
Overall Results	Overall result of the test as pass/fail

Each test case is presented in the format described in Table 1-1.

1.5 Document Organization

The remainder of this document describes the evaluation and demonstration activities that were performed for Builds 1, 2, 3, and 4. Each build has a section devoted to it, with that section being divided into subsections that describe the evaluation of MUD-related capabilities and the demonstration of non-MUD-related capabilities (if applicable). The MUD files used for each build are also provided.

Acronyms used in this document can be found in the Acronyms Appendix in NIST SP 1800-15B.

1.6 Typographic Conventions

The following table presents typographic conventions used in this document.

Typeface/ Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide*.
Bold	names of menus, options, command buttons, and fields	Choose **File > Edit.**
`Monospace`	command-line input, onscreen computer output, sample code examples, status codes	`Mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

2 Build 1

Build 1 uses equipment from Cisco Systems and Forescout. The Cisco MUD Manager is used to support MUD and the Forescout Virtual Appliances, and Enterprise Manager is used to perform non-MUD-related device discovery on the network.

2.1 Evaluation of MUD-Related Capabilities

The functional evaluation that was conducted to verify that Build 1 conforms to the MUD specification was based on the Build 1-specific requirements defined in Table 2-1.

2.1.1 Requirements

Table 2-1: MUD Use Case Functional Requirements

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-1	The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled **IoT device emit a MUD file URL via DHCP, Link Layer Discovery Protocol [LLDP], or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL).**			IoT-1-v4, IoT-1-v6, IoT-11-v4, IoT-11-v6
CR-1.a		Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one **MUD URL, in hypertext transfer protocol secure**		IoT-1-v4, IoT-1-v6, IoT-11-v4, IoT-11-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		(https) scheme, within the DHCP transaction.		
CR-1.a.1			The DHCP server shall be able to receive **DHCPv4 DISCOVER and REQUEST with IANA code 161** (OPTION_MUD_URL_V4) from the MUD-enabled IoT device.	IoT-1-v4, IoT-11-v4
CR-1.a.2			The DHCP server shall be able to receive **DHCPv6 Solicit and Request with IANA code 112** (OPTION_MUD_URL_V6) from the MUD-enabled IoT device.	IoT-1-v6, IoT-11-v6
CR-1.b		Upon initialization, the MUD-enabled IoT device shall **emit the MUD URL as an LLDP extension.**		IoT-1-v4, IoT-1-v6, IoT-11-v4, IoT-11-v6
CR-1.b.1			The network service shall be able to **process** the MUD URL that is received as an **LLDP extension.**	IoT-1-v4, IoT-1-v6, IoT-11-v4, IoT-11-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-2	The IoT DDoS example implementation shall include the capability for the MUD URL **to be provided to a MUD manager.**			IoT-1-v4, IoT-1-v6
CR-2.a		The DHCP server shall **assign an IP address lease** to the MUD-enabled IoT device.		IoT-1-v4, IoT-1-v6
CR-2.a.1			The MUD-enabled IoT device shall **receive the IP address.**	IoT-1-v4, IoT-1-v6
CR-2.b		**The DHCP server shall** receive the DHCP message and **extract the MUD URL, which is then passed to the MUD manager.**		IoT-1-v4, IoT-1-v6
CR-2.b.1			**The MUD manager shall receive the MUD URL.**	IoT-1-v4, IoT-1-v6
CR-3	The IoT DDoS example implementation shall include a **MUD manager that can request a MUD file and signature from a MUD file server.**			IoT-1-v4, IoT-1-v6
CR-3.a		The MUD manager shall use the GET method (RFC 7231) to		IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		request MUD and signature files (per RFC 7230) from the MUD file server and can **validate the MUD file server's Transport Layer Security (TLS) certificate** by using the rules in RFC 2818.		
CR-3.a.1			**The MUD file server shall receive the https request from the MUD manager.**	IoT-1-v4, IoT-1-v6
CR-3.b		**The MUD manager** shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it **cannot validate the MUD file server's TLS certificate** by using the rules in RFC 2818.		IoT-2-v4, IoT-2-v6
CR-3.b.1			**The MUD manager shall drop the connection** to the MUD file server.	IoT-2-v4, IoT-2-v6
CR-3.b.2			**The MUD manager shall send locally defined policy to the**	IoT-2-v4, IoT-2-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	
CR-4	The IoT DDoS example implementation shall include a **MUD file server that can serve a MUD file and signature to the MUD manager**.			IoT-1-v4, IoT-1-v6
CR-4.a		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** (signed using distinguished encoding rules [DER]-encoded Cryptographic Message Syntax [CMS] [RFC 5652]) was valid at the time of signing, i.e., the **certificate had not expired.**		IoT-1-v4, IoT-1-v6
CR-4.b		**The MUD file server shall serve the file and signature to the**		IoT-3-v4, IoT-3-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file was valid at the time of signing, i.e., the **certificate had already expired when it was used to sign the MUD file.**		
CR-4.b.1			The MUD manager shall cease to process the MUD file.	IoT-3-v4, IoT-3-v6
CR-4.b.2			The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-3-v4, IoT-3-v6
CR-5	The IoT DDoS example implementation shall include a **MUD manager** that **can translate local network configurations based on the MUD file.**			IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-5.a		**The MUD manager shall successfully validate the signature of the MUD file.**		IoT-1-v4, IoT-1-v6
CR-5.a.1			The MUD manager, after validation of the MUD file signature, shall **check for an existing MUD file and translate abstractions in the MUD file to router or switch configurations.**	IoT-1-v4, IoT-1-v6
CR-5.a.2			The MUD manager shall **cache** this newly received MUD file.	IoT-10-v4, IoT-10-v6
CR-5.b		The MUD manager shall attempt to validate the signature of the **MUD file,** but the **signature validation fails** (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason).		IoT-4-v4, IoT-4-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-5.b.1			**The MUD manager shall cease processing the MUD file.**	IoT-4-v4, IoT-4-v6
CR-5.b.2			**The MUD manager shall send locally defined policy to the router or switch** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-4-v4, IoT-4-v6
CR-6	The IoT DDoS example implementation shall include a **MUD manager that can configure the MUD PEP,** i.e., the router or switch nearest the MUD-enabled IoT device that emitted the URL.			IoT-1-v4, IoT-1-v6
CR-6.a		**The MUD manager shall install a router configuration** on the router or switch nearest the MUD-enabled IoT device that emitted the URL.		IoT-1-v4, IoT-1-v6
CR-6.a.1			**The router or switch shall have been configured to enforce the route filter sent**	IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			by the MUD manager.	
CR-7	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file.**			IoT-5-v4, IoT-5-v6
CR-7.a		The MUD-enabled IoT device shall attempt to **initiate outbound traffic to approved internet services.**		IoT-5-v4, IoT-5-v6
CR-7.a.1			The router or switch shall receive the attempt and shall **allow it to pass** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-7.b		An approved **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4, IoT-5-v6
CR-7.b.1			The router or switch shall receive the attempt and shall **allow it to pass** based on	IoT-5-v4, IoT-5-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			the filters from the MUD file.	
CR-8	The IoT DDoS example implementation shall **deny communications from a MUD-enabled IoT device to unapproved internet services** (i.e., services that are denied by virtue of not being explicitly approved).			IoT-5-v4, IoT-5-v6
CR-8.a		The MUD-enabled IoT device shall **attempt to initiate outbound traffic to unapproved** (implicitly denied) **internet services.**		IoT-5-v4, IoT-5-v6
CR-8.a.1			**The router or switch shall receive the attempt and** shall **deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8.b		**An unapproved** (implicitly denied) **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4, IoT-5-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-8.b.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8.c		The MUD-enabled IoT device shall initiate communications to an internet service that is **approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device.**		IoT-5-v4, IoT-5-v6
CR-8.c.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8.d		An internet service shall initiate communications to a MUD-enabled device that is **approved to initiate communications with the internet service but that is not approved to receive**		IoT-5-v4, IoT-5-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		communications initiated by the internet service.		
CR-8.d.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-9	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate laterally with devices that are approved** in the MUD file.			IoT-6-v4, IoT-6-v6
CR-9.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to approved devices.**		IoT-6-v4, IoT-6-v6
CR-9.a.1			**The router or switch shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-9.b		An approved device **shall attempt to initiate a lateral connection to the MUD-enabled IoT device.**		IoT-6-v4, IoT-6-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-9.b.1			**The router or switch shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-10	The IoT DDoS example implementation shall **deny lateral communications from a MUD-enabled IoT device to devices that are not approved** in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).			IoT-6-v4, IoT-6-v6
CR-10.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to unapproved** (implicitly denied) **devices.**		IoT-6-v4, IoT-6-v6
CR-10.a.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-10.b		**An unapproved** (implicitly denied) **device shall attempt to initi-**		IoT-6-v4, IoT-6-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		ate a lateral connection to the MUD-enabled IoT device.		
CR-10.b.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-11	If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, **the DHCP server must notify the MUD manager of any corresponding change to the DHCP state** of the MUD-enabled IoT device, and the MUD manager should **remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.**			IoT-7-v4, IoT-7-v6
CR-11.a		The MUD-enabled IoT **device shall explicitly release the IP address lease** (i.e., it sends a DHCP release message to the DHCP server).		IoT-7-v4, IoT-7-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-11.a.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has been released.**	IoT-7-v4, IoT-7-v6
CR-11.a.2			**The MUD manager should remove all policies** associated with the disconnected IoT device that had been configured on the MUD PEP router/switch.	IoT-7-v4, IoT-7-v6
CR-11.b		The MUD-enabled IoT **device's IP address lease shall expire.**		IoT-8-v4, IoT-8-v6
CR-11.b.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has expired.**	IoT-8-v4, IoT-8-v6
CR-11.b.2			**The MUD manager should remove all policies** associated with the affected IoT device that had been configured on the MUD PEP router/switch.	IoT-8-v4, IoT-8-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-12	The IoT DDoS example implementation shall include a **MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed** for the MUD file indicated by the MUD URL**. The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.**			IoT-10-v4, IoT-10-v6
CR-12.a		The MUD manager shall check if the file associated with the **MUD URL is present in its cache** and shall determine that it is.		IoT-10-v4, IoT-10-v6
CR-12.a.1			The MUD manager shall **check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager shall apply the contents of the cached MUD file.	IoT-10-v4, IoT-10-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-12.a.2			The MUD manager **shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.	IoT-10-v4, IoT-10-v6
CR-13	The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the MUD PEP router/switch will get configured with **all possible instantiations of that rule,** insofar as **each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the MUD PEP router/switch.**			IoT-9-v4, IoT-9-v6
CR-13.a		The MUD file for a device shall contain a rule involving a **domain that can resolve**		IoT-9-v4, IoT-9-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		to multiple IP addresses when queried by the MUD PEP router/switch. **An Access Control List (ACL) for permitting access to each of those IP addresses will be inserted into the MUD PEP router/switch** for the device in question, and the device will be permitted to communicate with all of those IP addresses.		
CR-13.a.1			IPv4 addressing is used on the network.	IoT-9-v4
CR-13.a.2			IPv6 addressing is used on the network.	IoT-9-v6

2.1.2 Test Cases

This section contains the test cases that were used to verify that Build 1 met the requirements listed in Table 2-1.

2.1.2.1 Test Case IoT-1-v4

Table 2-2: Test Case IoT-1-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, Link Layer Discovery

Test Case Field	Description
	Protocol [LLDP], or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL). (CR-2) The IoT DDoS example implementation shall include the capability for the MUD URL to be provided to a MUD manager. (CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server. (CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager. (CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file. (CR-6) The IoT DDoS example implementation shall include a MUD manager that can configure the router or switch nearest the MUD-enabled IoT device that emitted the URL.
Testable Requirements	(CR-1.a) Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one MUD URL, in https scheme, within the DHCP transaction. (CR-1.a.1) The DHCP server shall be able to receive DHCPv4 DISCOVER and REQUEST with IANA code 161 (OPTION_MUD_URL_V4) from the MUD-enabled IoT device. (NOTE: Test IoT-1-v6 does not test this requirement; instead, it tests CR-1.a.2, which pertains to DHCPv6 rather than DHCPv4.) OR (CR-1.b) Upon initialization, the MUD-enabled IoT device shall emit the MUD URL as an LLDP extension. (CR-1.b.1) The network service shall be able to process the MUD URL that is received as an LLDP extension. (CR-2.a) The DHCP server shall assign an IP address lease to the MUD-enabled IoT device. (CR-2.a.1) The MUD-enabled IoT device shall receive the IP address. (CR-2.b) The DHCP server shall receive the DHCP message and extract the MUD URL, which is then passed to the MUD manager. (CR-2.b.1) The MUD manager shall receive the MUD URL.

Test Case Field	Description
	(CR-3.a) The MUD manager shall use the “GET” method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server and can validate the MUD file server’s TLS certificate by using the rules in RFC 2818. (CR-3.a.1) The MUD file server shall receive the https request from the MUD manager. (CR-4.a) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the certificate had not expired. (CR-5.a) The MUD manager shall successfully validate the signature of the MUD file. (CR-5.a.1) The MUD manager, after validation of the MUD file signature, shall check for an existing MUD file and translate abstractions in the MUD file to router or switch configurations. (CR-6.a) The MUD manager shall install a router configuration on the router or switch nearest the MUD-enabled IoT device that emitted the URL. (CR-6.a.1) The router or switch shall have been configured to enforce the route filter sent by the MUD manager.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device’s MUD file, assuming the MUD file has a valid signature and is served from a MUD file server that has a valid TLS certificate
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.PT-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi

Test Case Field	Description
MUD File(s) Used	*ciscopi2.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The device's MUD file has a valid signature that was signed by a certificate that had not yet expired, and it is being hosted on a MUD file server that has a valid TLS certificate. 4. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 5. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 2.1.3.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. IoT device automatically emits a MUD URL in one of the following methods: a. DHCPv4 message containing the device's MUD URL (IANA code 161) (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) b. LLDP message containing the device's MUD URL in its extension 2. Corresponding service is responsible for the following actions: a. The DHCP server receives a DHCP message containing the IoT device's MUD URL. b. The LLDP server receives an LLDP advertisement containing the IoT device's MUD URL. 3. The respective service (LLDP or DHCP) extracts the MUD URL. 4. The MUD URL is then provided to the MUD manager.

Test Case Field	Description
	5. The MUD manager automatically contacts the MUD file server that is located using the MUD URL, verifies that it has a valid TLS certificate, requests and receives the MUD file and signature from the MUD file server, validates the MUD file's signature, and translates the MUD file's contents into appropriate route filtering rules. It then installs these rules onto the MUD PEP for the IoT device in question so that this router/switch is now configured to enforce the policies specified in the MUD file. 6. The DHCP server offers an IP address lease to the newly connected IoT device. 7. The IoT device requests this IP address lease, which the DHCP server acknowledges.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. The expected configuration should resemble the following details: `Extended IP access list mud-81726-v4fr.in` `   10 permit tcp any host 192.168.4.7 eq www ack syn` `   20 permit tcp any host 192.168.10.104 eq www` `   30 permit tcp any host 192.168.10.105 eq www` `   50 permit tcp any 192.168.10.0 0.0.0.255 eq www` `   60 permit tcp any 192.168.13.0 0.0.0.255 eq www` `   70 permit tcp any 192.168.14.0 0.0.0.255 eq www` `   80 permit tcp any eq 22 any` `   81 permit udp any eq bootpc any eq bootps` `   82 permit udp any any eq domain` `   83 deny ip any any` All protocol exchanges described in steps 1–7 above are expected to occur and can be viewed via Wireshark if desired. If the router/switch does not get configured in accordance with the MUD file, each exchange of DHCP and MUD-related protocol traffic should be viewed on the network via Wireshark to determine which transactions did not proceed as expected, and the observed and absent protocol exchanges should be described here.
Actual Results	**Dynamic access-session on switch**:

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>Build1#sh access-session int g1/0/15 det
          Interface: GigabitEthernet1/0/15
             IIF-ID: 0x1B6BCEA5
        MAC Address: b827.ebeb.6c8b
       IPv6 Address: Unknown
       IPv4 Address: 192.168.13.9
          User-Name: b827ebeb6c8b
             Status: Authorized
             Domain: DATA
     Oper host mode: multi-auth
   Oper control dir: both
    Session timeout: N/A
  Common Session ID: C0A80A02000000A6A9828F06
    Acct Session ID: 0x0000003b
             Handle: 0x2200009c
     Current Policy: mud-mab-test

Server Policies:
            ACS ACL: mud-81726-v4fr.in
         Vlan Group: Vlan: 3

Method status list:
      Method   State
        mab   Authc Success
</pre>
<u>access-list on switch</u>:
<pre>Build1#sh access-list mud-81726-v4fr.in
Extended IP access list mud-81726-v4fr.in
    10 permit tcp any host 192.168.4.7 eq www ack syn
    20 permit tcp any host 192.168.10.104 eq www
    30 permit tcp any host 192.168.10.105 eq www
    50 permit tcp any 192.168.10.0 0.0.0.255 eq www
    60 permit tcp any 192.168.13.0 0.0.0.255 eq www
    70 permit tcp any 192.168.14.0 0.0.0.255 eq www
    80 permit tcp any eq 22 any
    81 permit udp any eq bootpc any eq bootps
    82 permit udp any any eq domain
    83 deny ip any any</pre></td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

Test case IoT-1-v6 is identical to test case IoT-1-v4 except that IoT-1-v6 tests requirement CR-1.a.2, whereas IoT-1-v4 tests requirement CR-1.a.1. Hence, as explained above, test case IoT-1-v6 uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

2.1.2.2 Test Case IoT-2-v4

Table 2-3: Test Case IoT-2-v4

Test Case Field	Description
Parent Requirement	(CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server.
Testable requirement	(CR-3.b) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it cannot validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.b.1) The MUD manager shall drop the connection to the MUD file server. (CR-3.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD manager is not able to validate the TLS certificate of a MUD file server when trying to retrieve the MUD file for a specific IoT device, the MUD manager will drop the connection to the MUD file server and configure the router/switch according to locally defined policy regarding whether to allow or block traffic to the IoT device in question
Associated Test Case(s)	IoT-11-v4 (for the v6 version of this test, IoT-11-v6)
Associated Cybersecurity Framework Subcategory(ies)	PR.AC-7
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*ciscopi2.json*

Test Case Field	Description
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The MUD file server that is hosting the MUD file of the device under test does not have a valid TLS certificate. 4. Local policy has been defined to ensure that if the MUD file for a device is located on a server with an invalid certificate, the router/switch will be configured to deny all communication to and from the device. 5. The MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 3. The DHCP server offers an IP address lease to the newly connected IoT device. 4. The IoT device requests this IP address lease, which the DHCP server acknowledges. 5. The DHCP server sends the MUD URL to the MUD manager. 6. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, determines that it does not have a valid TLS certificate, and drops the connection to the MUD file server.

Test Case Field	Description
	7. The MUD manager configures the router/switch that is closest to the IoT device so that it denies all communication to and from the IoT device.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to local policy for communication to/from the IoT device.
Actual Results	`***MUDC [STATUS][send_mudfs_request:2005]-->` `Request URI` *<https://mudfileserver/ciscopi2>* `</home/mudtester/ca.cert.pem>` `*   Trying 192.168.4.5...` `* TCP_NODELAY set` `* Connected to mudfileserver (192.168.4.5) port 443 (#0)` `* found 1 certificate in /home/mudtester/ca.cert.pem` `* found 400 certificates in /etc/ssl/certs` `* ALPN, offering http/1.1` `* SSL connection using TLS1.2 / ECDHE_RSA_AES_256_GCM_SHA384` `* server certificate verification failed. CAfile:` `/home/mudtester/ca.cert.pem CRLfile: none` `* stopped the pause stream!` `* Closing connection 0` `***MUDC [ERROR][fetch_file:182]--> curl_easy_perform()` `failed: Peer certificate cannot be authenticated with given` `CA certificates` `***MUDC [INFO][send_mudfs_request:2019]--> Unable to reach` `MUD fileserver to fetch MUD file.  Will try to append .json` `*   Trying 192.168.4.5...` `* TCP_NODELAY set` `* Connected to mudfileserver (192.168.4.5) port 443 (#0)` `* found 1 certificate in /home/mudtester/ca.cert.pem` `* found 400 certificates in /etc/ssl/certs` `* ALPN, offering http/1.1` `* SSL connection using TLS1.2 / ECDHE_RSA_AES_256_GCM_SHA384` `* server certificate verification failed. CAfile:` `/home/mudtester/ca.cert.pem CRLfile: none` `* stopped the pause stream!` `* Closing connection 0` `***MUDC [ERROR][fetch_file:182]--> curl_easy_perform()` `failed: Peer certificate cannot be authenticated with given` `CA certificates` `***MUDC [ERROR][send_mudfs_request:2027]--> Unable to reach` `MUD fileserver to fetch .json file` `***MUDC [INFO][mudc_construct_head:135]--> status_code: 204,` `content_len: 14, extra_headers: (null)`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>
***MUDC [INFO][mudc_construct_head:152]--> HTTP header:
HTTP/1.1 204 No Content
Content-Length: 14

***MUDC [INFO][send_error_result:176]--> error from FS

***MUDC [ERROR][send_mudfs_request:2170]--> mudfs_conn
failed
</pre>
<hr>
<pre>
Build1#sho access-session int g1018 det
            Interface GigabitEthernet1018
               IIF-ID 0x181835C2
          MAC Address b827.eba7.0533
         IPv6 Address Unknown
         IPv4 Address 192.168.10.106
            User-Name b827eba70533
               Status Authorized
               Domain DATA
       Oper host mode multi-auth
     Oper control dir both
      Session timeout NA
    Common Session ID C0A80A02000000CCBDB267F8
      Acct Session ID 0x00000046
               Handle 0x100000c2
       Current Policy mud-mab-test

Server Policies

Method status list
       Method   State
          mab   Authc Success
</pre></td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

As explained above, test IoT-2-v6 is identical to test IoT-2-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

2.1.2.3 Test Case IoT-3-v4

Table 2-4: Test Case IoT-3-v4

Test Case Field	Description
Parent Requirement	(CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager.
Testable Requirement	(CR-4.b) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file was valid at the time of signing, i.e., the certificate had already expired when it was used to sign the MUD file. (CR-4.b.1) The MUD manager shall cease to process the MUD file. (CR-4.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD file server serves a MUD file with a signature that was created with an expired certificate, the MUD manager will cease processing the MUD file
Associated Test Case(s)	IoT-11-v4 (for the v6 version of this test, IoT-11-v6)
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*expiredcerttest.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The IoT device's MUD file is being hosted on a MUD file server that has a valid TLS certificate, but the MUD file signature was signed by a certificate that had already expired at the time of signature.

Test Case Field	Description
	4. Local policy has been defined to ensure that if the MUD file for a device has a signature that was signed by a certificate that had already expired at the time of signature, the device's MUD PEP router/switch will be configured to deny all communication to/from the device. 5. The MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 3. The DHCP server offers an IP address lease to the newly connected IoT device. 4. The IoT device requests this IP address lease, which the DHCP server acknowledges. 5. The DHCP server sends the MUD URL to the MUD manager. 6. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. 7. The MUD file server serves the MUD file and signature to the MUD manager, and the MUD manager detects that the MUD file's signature was created by using a certificate that had already expired at the time of signing.

Test Case Field	Description
	8. The MUD manager configures the router/switch that is closest to the IoT device so that it denies all communication to and from the IoT device.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to deny all communication to and from the IoT device. The expected configuration should resemble the details below. Expecting a show access session without a MUD file as seen below: `Build1#show access-session int g1018 det` `Interface  GigabitEthernet1018` `IIF-ID  0x181835C2` `MAC Address  b827.eba7.0533` `IPv6 Address  Unknown` `IPv4 Address  192.168.10.106` `User-Name  b827eba70533` `Status  Authorized` `Domain  DATA` `Oper host mode  multi-auth` `Oper control dir  both` `Session timeout  NA` `Common Session ID  C0A80A02000000CCBDB267F8` `Acct Session ID  0x00000046` `Handle  0x100000c2` `Current Policy  mud-mab-test` `Server Policies` `Method status list` `Method          State` `mab          Authc Success`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td>Actual Results</td><td><pre>
***MUDC [INFO][verify_mud_content:1594]--> BIO_reset <1>

***MUDC [ERROR][verify_mud_content:1604]--> Verification
Failure

139713269933824:error:2E099064:CMS routines:cms_sign-
erinfo_verify_cert:certificate verify er-
ror:../crypto/cms/cms_smime.c:253:Verify error:certificate
has expired
***MUDC [INFO][send_mudfs_request:2092]--> Verification
failed. Manufacturer Index <0>

***MUDC [INFO][mudc_construct_head:135]--> status_code: 401,
content_len: 19, extra_headers: (null)
***MUDC [INFO][mudc_construct_head:152]--> HTTP header:
HTTP/1.1 401 Unauthorized
Content-Length: 19

***MUDC [INFO][send_error_result:176]--> Verification failed
***MUDC [ERROR][send_mudfs_request:2170]--> mudfs_conn
failed
</pre>
<hr>
<pre>
Build1#sho access-session int g1018 det
           Interface GigabitEthernet1018
              IIF-ID 0x181835C2
         MAC Address b827.eba7.0533
        IPv6 Address Unknown
        IPv4 Address 192.168.10.106
           User-Name b827eba70533
              Status Authorized
              Domain DATA
      Oper host mode multi-auth
    Oper control dir both
     Session timeout NA
   Common Session ID C0A80A02000000CCBDB267F8
     Acct Session ID 0x00000046
              Handle 0x100000c2
      Current Policy mud-mab-test

Server Policies

Method status list
      Method   State
         mab   Authc Success
</pre></td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

As explained above, test IoT-3-v6 is identical to test IoT-3-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

2.1.2.4 Test Case IoT-4-v4

Table 2-5: Test Case IoT-4-v4

Test Case Field	Description
Parent Requirement	(CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file.
Testable Requirement	(CR-5.b) The MUD manager shall attempt to validate the signature of the MUD file, but the signature validation fails (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason). (CR-5.b.1) The MUD manager shall cease processing the MUD file. (CR-5.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if the MUD manager determines that the signature on the MUD file it receives from the MUD file server is invalid, it will cease processing the MUD file and configure the router/switch according to locally defined policy regarding whether to allow or block traffic to the IoT device in question
Associated Test Case(s)	IoT-11-v4 (for the v6 version of this test, IoT-11-v6)
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*ciscop2.json*

Test Case Field	Description
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The MUD file that is served from the MUD file server to the MUD manager has a signature that is invalid, even though it was signed by a certificate that had not expired at the time of signing. 4. Local policy has been defined to ensure that if the MUD file for a device has an invalid signature, the device's MUD PEP router/switch will be configured to deny all communication to and from the device. 5. The MUD PEP router/switch does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 3. The DHCP server offers an IP address lease to the newly connected IoT device. 4. The IoT device requests this IP address lease, which the DHCP server acknowledges. 5. The DHCP server sends the MUD URL to the MUD manager. 6. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. 7. The MUD file server sends the MUD file, and the MUD manager detects that the MUD file's signature is invalid.

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>8. The MUD manager configures the router/switch that is closest to the IoT device so that it denies all communication to and from the IoT device.</td></tr>
<tr><td>Expected Results</td><td>The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to deny all communication to/from the IoT device. The expected configuration should resemble the following details.
Expecting a show access session without a MUD file as seen below:
<pre>
Build1#sho access-session int g1018 det
            Interface GigabitEthernet1018
               IIF-ID 0x181835C2
          MAC Address b827.eba7.0533
         IPv6 Address Unknown
         IPv4 Address 192.168.10.106
            User-Name b827eba70533
               Status Authorized
               Domain DATA
       Oper host mode multi-auth
     Oper control dir both
      Session timeout NA
    Common Session ID C0A80A02000000CCBDB267F8
      Acct Session ID 0x00000046
               Handle 0x100000c2
       Current Policy mud-mab-test

Server Policies

Method status list
       Method   State
          mab   Authc Success
</pre></td></tr>
<tr><td>Actual Results</td><td><pre>
> GET /ciscopi2.json HTTP/1.1
Host: mudfileserver
Accept: */*
</pre>
[Omitted for brevity]
<pre>
***MUDC [STATUS][send_mudfs_request:2060]-->
Request signature URI <https://mudfileserver/ciscopi2.p7s>
</home/mudtester/mud-intermediate.pem>
</pre></td></tr>
</table>

Test Case Field	Description
	`*   Trying 192.168.4.5...` `* TCP_NODELAY set` `* Connected to mudfileserver (192.168.4.5) port 443 (#0)` `* found 1 certificate in /home/mudtester/mud-intermedi-ate.pem` `* found 400 certificates in /etc/ssl/certs` `* ALPN, offering http/1.1` `* SSL connection using TLS1.2 / ECDHE_RSA_AES_256_GCM_SHA384` `*       server certificate verification OK` `*       server certificate status verification SKIPPED` `*       common name: mudfileserver (matched)` `*       server certificate expiration date OK` `*       server certificate activation date OK` `*       certificate public key: RSA` `*       certificate version: #3` `*       subject: C=US,ST=Maryland,L=Rockville,O=National Cy-bersecurity Center of Excellence - NIST,CN=mudfileserver` `*       start date: Fri, 05 Oct 2018 00:00:00 GMT` `*       expire date: Wed, 13 Oct 2021 12:00:00 GMT` `*       issuer: C=US,O=DigiCert Inc,CN=DigiCert Test SHA2 Intermediate CA-1` `*       compression: NULL` `* ALPN, server did not agree to a protocol` `> GET /ciscopi2.p7s HTTP/1.1` `Host: mudfileserver` `Accept: */*` **[Omitted for brevity]** `***MUDC [INFO][send_mudfs_request:2080]--> MUD signature file successfully retrieved` `***MUDC [DEBUG][verify_mud_content:1543]--> MUD signature file (length 4680)` `[shortened logs]` `***MUDC [INFO][verify_mud_content:1594]--> BIO_reset <1>` `***MUDC [ERROR][verify_mud_content:1604]-->` **`Verification Failure`** `140561528563456:error:2E09A09E:CMS routines:CMS_Sign-erInfo_verify_content:verification fail-ure:../crypto/cms/cms_sd.c:819:` `140561528563456:error:2E09D06D:CMS routines:CMS_verify:con-tent verify error:../crypto/cms/cms_smime.c:393:`

Test Case Field	Description
	***MUDC [INFO][send_mudfs_request:2092]--> **Verification failed. Manufacturer Index <0>** ***MUDC [INFO][mudc_construct_head:135]--> status_code: 401, content_len: 19, extra_headers: (null) ***MUDC [INFO][mudc_construct_head:152]--> HTTP header: HTTP/1.1 401 Unauthorized Content-Length: 19 ***MUDC [INFO][send_error_result:176]--> **Verification failed** ***MUDC [ERROR][send_mudfs_request:2170]--> **mudfs_conn failed**
	Switch access-session: Build1#sho access-session int g1/0/18 det Interface: GigabitEthernet1/0/18 IIF-ID: 0x11C404C6 MAC Address: b827.eba7.0533 IPv6 Address: Unknown IPv4 Address: 192.168.10.106 User-Name: b827eba70533 Status: Authorized Domain: DATA Oper host mode: multi-auth Oper control dir: both Session timeout: N/A Common Session ID: C0A80A02000000CDBDB68A30 Acct Session ID: 0x00000047 Handle: 0x690000c3 Current Policy: mud-mab-test Server Policies: Method status list: Method State mab Authc Success
Overall Results	Pass

As explained above, test IoT-4-v6 is identical to test IoT-4-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

2.1.2.5 Test Case IoT-5-v4

Table 2-6: Test Case IoT-5-v4

Test Case Field	Description
Parent Requirement	(CR-7) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file. (CR-8) The IoT DDoS example implementation shall deny communications from a MUD-enabled IoT device to unapproved internet services (i.e., services that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-7.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to approved internet services. (CR-7.a.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-7.b) An approved internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-7.b.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-8.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to unapproved (implicitly denied) internet services. (CR-8.a.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.b) An unapproved (implicitly denied) internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-8.b.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.c) The MUD-enabled IoT device shall initiate communications to an internet service that is approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device. (CR-8.c.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.d) An internet service shall initiate communications to a MUD-enabled device that is approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service.

Test Case Field	Description
	(CR-8.d.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with internet services. Further shows that the policies that are configured on the MUD PEP router/switch with respect to communication with internet services will be enforced as expected, with communications that are configured as denied being blocked, and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4 (for the v6 version of this test, IoT-1-v6)
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.IP-1, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*ciscopi2.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the following policies for the IoT device in question (as defined in the MUD file in Section 2.1.3): a) Explicitly permit *https://yes-permit-from.com* to initiate communication with the IoT device. b) Explicitly permit the IoT device to initiate communication with *https://yes-permit-to.com*. c) Implicitly deny all other communications with the internet, including denying i) the IoT device to initiate communication with *https://yes-permit-from.com*

Test Case Field	Description
	ii) *https://yes-permit-to.com* to initiate communication with the IoT device iii) communication between the IoT device and all other internet locations, such as *https://unnamed-to.com* (by not mentioning this or any other URLs in the MUD file)
Procedure	Note: Procedure steps with strike-through were not tested in this phase because ingress Dynamic Access Control Lists (DACLs) are not supported in this implementation. 1. As stipulated in the preconditions, right before this test, test IoT-1-v4 (or IoT-1-v6) must have been run successfully. 2. Initiate communications from the IoT device to *https://yes-permit-to.com* and verify that this traffic is received at *https://yes-permit-to.com*. (egress) 3. Initiate communications to the IoT device from *https://yes-permit-to.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress) ~~4. Initiate communications to the IoT device from *https://yes-permit-from.com* and verify that this traffic is received at the IoT device. (ingress)~~ ~~5. Initiate communications from the IoT device to *https://yes-permit-from.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at *https://yes-permit-from.com*. (ingress)~~ 6. Initiate communications from the IoT device to *https://unnamed.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at *https://unnamed.com*. (egress) 7. Initiate communications to the IoT device from *https://unnamed.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress)

Test Case Field	Description
Expected Results	Each of the results that is listed as needing to be verified in procedure steps above occurs as expected.
Actual Results	Procedure 2: Connection to update server successfully initiated by IoT device: `pi@raspberrypi:~ $` **`wget`** ***`http://www.updateserver.com/`*** `--2018-12-13 21:28:00--  `*`http://www.updateserver.com/`* `Resolving `*`www.updateserver.com`*` (`*`www.updateserver.com`*`)... 192.168.4.7` `Connecting to `*`www.updateserver.com`*` (`*`www.up-dateserver.com`*`)\|192.168.4.7\|:80... connected.` `HTTP request sent, awaiting response... 200 OK` `Length: 10918 (11K) [text/html]` `Saving to: ‘index.html.2’` `index.html.2      100%[===================>]  10.66K  --.-KB/s    in 0s` `2018-12-13 21:28:00 (30.6 MB/s) - ‘index.html.2’ saved [10918/10918]` Procedure 3: Update server failed to connect to IoT device: `iot@update-server:~$` **`wget`** ***`http://192.168.13.9`*** `--2018-12-13 21:49:36--  `*`http://192.168.13.9/`* `Connecting to 192.168.13.9:80... failed: Connection timed out.` `Retrying.` Procedure 6: IoT device failed to connect to unapproved server: `pi@raspberrypi:~ $` **`wget`** ***`http://192.168.4.105`*** `--2018-12-14 16:42:36--  `*`http://192.168.4.105/`* `Connecting to 192.168.4.105:80... failed: Connection timed out.` `Retrying.` Procedure 7: Unapproved server attempts to connect to IoT device:

Test Case Field	Description
	`[mud@unapprovedserver ~]$ wget http://192.168.13.14` `--2018-12-14 13:03:32--  http://192.168.13.14/` `Connecting to 192.168.13.14:80... failed: Connection timed out.` `Retrying.`
Overall Results	Pass (for testable procedures—as stated, ingress cannot be tested)

259 As explained above, test IoT-5-v6 is identical to test IoT-5-v4 except that it uses IPv6, DHCPv6, and IANA
260 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

261 2.1.2.6 Test Case IoT-6-v4

262 **Table 2-7: Test Case IoT-6-v4**

Test Case Field	Description
Parent Requirement	(CR-9) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate laterally with devices that are approved in the MUD file. (CR-10) The IoT DDoS example implementation shall deny latterly communications from a MUD-enabled IoT device to devices that are not approved in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-9.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to approved devices. (CR-9.a.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-9.b) An approved device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-9.b.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-10.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to unapproved (implicitly denied) devices. (CR-10.a.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.

Test Case Field	Description
	(CR-10.b) An unapproved (implicitly denied) device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-10.b.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with lateral devices. Further shows that the policies that are configured on the MUD PEP router/switch with respect to communication with lateral devices will be enforced as expected, with communications that are configured as denied being blocked, and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4 (for the v6 version of this test, IoT-1-v6)
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.AC-5, PR.IP-1, PR.PT-3, PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*ciscopi2.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the following policies for the IoT device in question with respect to local communications (as defined in the MUD files in Section 2.1.3): a) Local-network class—Explicitly permit **local communication to and from the IoT device and any local hosts** (including the specific local hosts *anyhost-to* and *anyhost-from*) **for specific services,** as specified in the MUD file by source port: any; destination port: 80; and protocol: TCP, and which party initiates the connection. b) Manufacturer class—Explicitly permit **local communication to and from the IoT device and other classes of IoT devices, as**

Test Case Field	Description
	identified by their MUD URL (*www.devicetype.com*), and further constrained by source port: any; destination port: 80; and protocol: TCP. c) Same-manufacturer class—Explicitly permit **local communication to and from IoT devices of the same manufacturer as the IoT device in question (the domain in the MUD URLs [mud-fileserver] of the other IoT devices is the same as the domain in the MUD URL [mudfileserver] of the IoT device in question),** and further constrained by source port: any; destination port: 80; and protocol: TCP. d) Implicitly deny all other local communication that is not explicitly permitted in the MUD file, including denying i) ***anyhost-to* to initiate communications** with the IoT device ii) the **IoT device to initiate communications** with *anyhost-to* by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** iii) the **IoT device to initiate communications with *anyhost-from*** iv) ***anyhost-from* to initiate communications** with the IoT device by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** v) communications between the IoT device and all lateral hosts (including *unnamed-host*) whose **MUD URLs are not explicitly mentioned** as being permissible in the MUD file vi) communications between the IoT device and all lateral hosts whose **MUD URLS are explicitly mentioned** as being permissible, **but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** vii) communications between the IoT device and all lateral hosts that are **not from the same manufacturer** as the IoT device in question viii) communications between the IoT device and a lateral host that **is from the same manufacturer, but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**

Test Case Field	Description
Procedure	Note: Procedure steps with strike-through were not tested in this phase because ingress DACLs are not supported in this implementation. 1. As stipulated in the preconditions, right before this test, test IoT-1-v4 (or IoT-1-v6) must have been run successfully. ~~2. Local-network (ingress): Initiate communications to the IoT device from *anyhost-from* **for specific permitted service,** and verify that this traffic is received at the IoT device.~~ 3. Local-network (egress): **Initiate communications from the IoT device to *anyhost-from*** for specific permitted service, and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *anyhost-from*. 4. Local-network, controller, my-controller, manufacturer class (egress): Initiate communications from the IoT device to *anyhost-to* **for specific permitted service,** and verify that this traffic **is received** at *anyhost-to.* ~~5. Local-network, controller, my-controller, manufacturer class (ingress): **Initiate communications to the IoT device from *anyhost-to*** for specific permitted service, and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at the IoT device.~~ 6. No associated class (egress): Initiate communications from the IoT device to *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted),** and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *unnamed-host*. ~~7. No associated class (ingress): Initiate communications to the IoT device from *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted),** and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device.~~

Test Case Field	Description
	8. Same-manufacturer class (egress): Initiate communications from the IoT device to *same-manufacturer-host* (where *same-manufacturer-host* is **a host that is from the same manufacturer as the IoT device** in question) and verify that this traffic **is received** at *same-manufacturer-host*. 9. Same-manufacturer class (egress): Initiate communications from the IoT device to *same-manufacturer-host* (where *same-manufacturer-host* is **a host that is from the same manufacturer as the IoT device** in question) **but using a port or protocol that is not specified,** and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *same-manufacturer-host*.
Expected Results	Each of the results that is listed as needing to be verified in the procedure steps above occurs as expected.
Actual Results	The numbering in this section correlates with the procedure steps above: 3. Local_network (egress)—blocked: `pi@raspberrypi:~ $ wget https://192.168.10.106/` `--2019-01-31 19:59:23--  https://192.168.10.106/` `Connecting to 192.168.10.106:443... failed: Connection timed out.` `Retrying.` 4. Local-network, controller, my-controller, manufacturer class (egress)—allowed: Local_Network: `pi@raspberrypi:~ $ wget http://192.168.10.175` `--2018-12-14 15:11:50--  http://192.168.10.175/` `Connecting to 192.168.10.175:80... connected.` `HTTP request sent, awaiting response... 200 OK` `Length: 10701 (10K) [text/html]` `Saving to: ‘index.html.4’` `index.html.4      100%[==================>]  10.45K  --.-KB/s    in 0s`

Test Case Field	Description
	2018-12-14 15:11:50 (41.4 MB/s) - 'index.html.4' saved [10701/10701] Controller: pi@raspberrypi:~ $ **wget *http://192.168.10.105/*** --2019-01-31 21:03:45-- *http://192.168.10.105/* Connecting to 192.168.10.105:80... connected. HTTP request sent, awaiting response... 200 OK Length: 277 Saving to: 'index.html.10' in- dex.html.10 100%[===================>] 277 --.-KB/s in 0s 2019-01-31 21:03:45 (18.8 MB/s) - 'index.html.10' saved [277/277] My-controller: pi@raspberrypi:~ $ **wget *http://192.168.10.104/*** --2019-01-31 21:06:39-- *http://192.168.10.104/* Connecting to 192.168.10.104:80... connected. HTTP request sent, awaiting response... 200 OK Length: 10701 (10K) [text/html] Saving to: 'index.html.11' in- dex.html.11 100%[===================>] 10.45K --.-KB/s in 0s 2019-01-31 21:06:39 (32.5 MB/s) - 'index.html.11' saved [10701/10701] Manufacturer: pi@raspberrypi:~ $ **wget *http://192.168.14.2/*** --2019-01-31 21:13:47-- *http://192.168.14.2/* Connecting to 192.168.14.2:80... connected. HTTP request sent, awaiting response... 200 OK Length: 10701 (10K) [text/html] Saving to: 'index.html.12'

Test Case Field	Description
	in- dex.html.12 100%[===================>] 10.45K --.-KB/s in 0s 2019-01-31 21:13:47 (39.6 MB/s) - ‘index.html.12’ saved [10701/10701]
	6. No associated class (egress)—blocked: pi@raspberrypi:~ $ **wget** ***http://192.168.15.105*** --2018-12-14 17:15:36-- *http://192.168.15.105/* Connecting to 192.168.15.105:80... failed: Connection timed out. Retrying.
	8. Same-manufacturer class (egress)—allowed: pi@raspberrypi:~ $ **wget** ***http://192.168.13.8/*** --2019-01-31 21:16:41-- *http://192.168.13.8/* Connecting to 192.168.13.8:80... connected. HTTP request sent, awaiting response... 200 OK Length: 10701 (10K) [text/html] Saving to: ‘index.html.13’ index.html.13 100%[===================>] 10.45K --.-KB/s in 0s 2019-01-31 21:16:41 (37.9 MB/s) - ‘index.html.13’ saved [10701/10701]
	9. Same-manufacturer class (egress)—blocked: pi@raspberrypi:~ $ **wget** ***https://192.168.13.8/*** --2019-01-31 21:17:15-- *https://192.168.13.8/* Connecting to 192.168.13.8:443... failed: Connection timed out. Retrying.
Overall Results	Pass (for testable procedures—as stated, ingress cannot be tested)

As explained above, test IoT-6-v6 is identical to test IoT-6-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

265 ### 2.1.2.7 Test Case IoT-7-v4

266 **Table 2-8: Test Case IoT-7-v4**

Test Case Field	Description
Parent Requirement	(CR-11) If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, the DHCP server must notify the MUD manager of any corresponding change to the DHCP state of the MUD-enabled IoT device, and the MUD manager should remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.
Testable Requirement	(CR-11.a) The MUD-enabled IoT device shall explicitly release the IP address lease (i.e., it sends a DHCP release message to the DHCP server). (CR-11.a.1) The DHCP server shall notify the MUD manager that the device's IP address lease has been released. (CR-11.a.2) The MUD manager should remove all policies associated with the disconnected IoT device that had been configured on the MUD PEP router/switch.
Description	Shows that when a MUD-enabled IoT device explicitly releases its IP address lease, the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch
Associated Test Case(s)	IoT-1-v4 (or IoT-1-v6 when IPv6 addressing is used)
Associated Cybersecurity Framework Subcategory(ies)	PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*ciscopi2.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the policies defined in the MUD file in section 2.1.3 for the IoT device in question.

Test Case Field	Description
Procedure	1. As stipulated in the preconditions, right before this test, test IoT-1-v4 (or IoT-1-v6) must have been run successfully. Verify that the MUD PEP router/switch for the IoT device has been configured to enforce the policies listed in the preconditions section above for the IoT device in question. 2. Cause a DHCP release of the IoT device in question. 3. Verify that all the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Expected Results	All of the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Actual Results	Procedure 1: `Build1#`**`sh access-session int g1/0/15 det`** `Interface:  GigabitEthernet1/0/15` `IIF-ID:  0x1B6BCEA5` `MAC Address:  b827.ebeb.6c8b` `IPv6 Address:  Unknown` `IPv4 Address:  192.168.13.17` `User-Name:  b827ebeb6c8b` `Status:  Authorized` `Domain:  DATA` `Oper host mode:  multi-auth` `Oper control dir:  both` `Session timeout:  N/A` `Common Session ID:  C0A80A0200000A6A9828F06` `Acct Session ID:  0x0000003b` `Handle:  0x2200009c` `Current Policy:  mud-mab-test` `Server Policies:` `ACS ACL: mud-81726-v4fr.in` `Vlan Group:  Vlan: 3` `Method status list:` `Method          State` `mab          Authc Success`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
Procedure 2:

<pre>
pi@raspberrypi:~ $ sudo dhclient -v -r
</pre>
<hr>
<pre>
Build1#sh access-session int g1/0/15 det
           Interface: GigabitEthernet1/0/15
              IIF-ID: 0x1B6BCEA5
         MAC Address: b827.ebeb.6c8b
        IPv6 Address: Unknown
        IPv4 Address: Unknown
           User-Name: b827ebeb6c8b
              Status: Authorized
              Domain: DATA
      Oper host mode: multi-auth
    Oper control dir: both
     Session timeout: N/A
   Common Session ID: C0A80A0200000A6A9828F06
     Acct Session ID: 0x0000003b
              Handle: 0x2200009c
      Current Policy: mud-mab-test

Server Policies:
             ACS ACL: mud-81726-v4fr.in
          Vlan Group: Vlan: 3

Method status list:
      Method   State
         mab   Authc Success
</pre>
</td></tr>
<tr><td>Overall Results</td><td>Failed</td></tr>
</table>

As explained above, test IoT-7-v6 is identical to test IoT-7-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

2.1.2.8 Test Case IoT-8-v4

Table 2-9: Test Case IoT-8-v4

Test Case Field	Description
Parent Requirement	(CR-11) If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, the DHCP server must notify the MUD manager of any corresponding change to the DHCP state of the MUD-enabled IoT device, and the MUD manager should remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.
Testable Requirement	(CR-11.b) The MUD-enabled IoT device's IP address lease shall expire. (CR-11.b.1) The DHCP server shall notify the MUD manager that the device's IP address lease has expired. (CR-11.b.2) The MUD manager should remove all policies associated with the affected IoT device that had been configured on the MUD PEP router/switch.
Description	Shows that when a MUD-enabled IoT device's IP address lease expires, the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch
Associated Test Case(s)	IoT-1-v4 (or IoT-1-v6 when IPv6 addressing is used)
Associated Cybersecurity Framework Subcategory(ies)	PR.IP-3, PR.DS-3
IoT Device(s) Under Test	TBD (Not testable in Build 1)
MUD File(s) Used	TBD (Not testable in Build 1)
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the policies defined in the MUD file in Section 2.1.3 for the IoT device in question.
Procedure	1. Configure the DHCP server to have a DHCP lease time of 10 minutes. 2. Run test IoT-1-v4 (or IoT-1-v6).

Test Case Field	Description
	3. Verify that the MUD PEP router/switch for the IoT device has been configured to enforce the policies listed above for the IoT device in question. 4. Disconnect the IoT device in question from the network. 5. After 10 minutes have elapsed, verify that all of the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Expected Results	Once 10 minutes have elapsed after disconnecting the IoT device from the network, all of the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Actual Results	TBD (Not testable in Build 1)
Overall Results	TBD (Not testable in Build 1)

As explained above, test IoT-8-v6 is identical to test IoT-8-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

2.1.2.9 Test Case IoT-9-v4

Table 2-10: Test Case IoT-9-v4

Test Case Field	Description
Parent Requirements	(CR-13) The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the MUD PEP router/switch will get configured with all possible instantiations of that rule, insofar as each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the MUD PEP router/switch.
Testable Requirements	(CR-13.a) The MUD file for a device shall contain a rule involving an external domain that can resolve to multiple IP addresses when queried by the MUD PEP router/switch. An ACL for permitting access to each of those IP addresses will be inserted into the MUD PEP router/switch for

Test Case Field	Description
	the device in question, and the device will be permitted to communicate with all of those IP addresses.
Description	Shows that if a domain in a MUD file rule resolves to multiple IP addresses when the address resolution is queried by the network gateway, then 1. ACLs instantiating that MUD file rule corresponding to each of these IP addresses will be configured in the gateway for the IoT device associated with the MUD file, and 2. the IoT device associated with the MUD file will be permitted to communicate with all of the IP addresses to which that domain resolves
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*dnstest.json*
Preconditions	1. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 2. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 2.1.3. (Therefore, the MUD file used in the test permits the device to send data to *www.updateserver.com.*) 3. The tester has access to a domain name system (DNS) server that will be used by the MUD PEP router/switch and can configure it such that it will resolve the domain *www.updateserver.com* to any of these addresses when queried by the MUD PEP router/switch: x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.

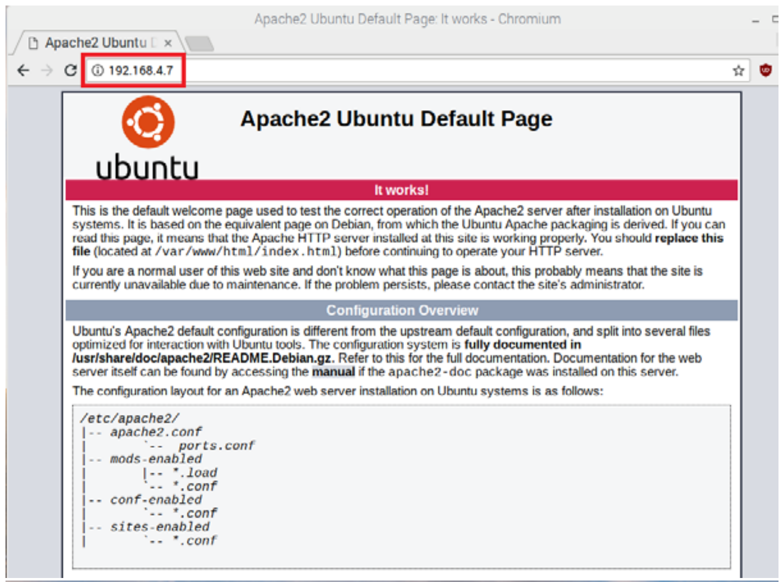
Test Case Field	Description
	4. There is an update server running at each of these three IP addresses.
Procedure	1. Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 2. Run test IoT-1-v4 (or IoT-1-v6). The result should be that the MUD PEP router/switch has been configured to explicitly permit the IoT device to initiate communication with *www.updateserver.com*. 3. Verify that the MUD PEP router/switch has been configured with ACLs that permit the IoT device to send data to IP addresses x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. 4. Have the device in question attempt to connect to x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to permit the IoT device to send data to IP addresses x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. The IoT device is permitted to send data to each of the update servers at these addresses.
Actual Results	**Procedures 1–2:** **Completed; excluded for brevity** **Procedure 3:** **MUD MANAGER:** `***MUDC [INFO][fetch_uri_from_macaddr:2166]--> ============= Returning URI:`*`https://mudfileserver/dnstest.json`* `***MUDC [INFO][handle_get_aclname:3149]--> Found URI `*`https://mudfileserver/dnstest.json`*` for MAC address b827ebcf7b81` `***MUDC [INFO][validate_muduri:3009]--> uri: `*`https://mudfileserver/dnstest.jsonhttps://mudfileserver/dnstest.json`* `***MUDC [INFO][validate_muduri:3035]--> ip: mudfileserver, filename: dnstest.json`

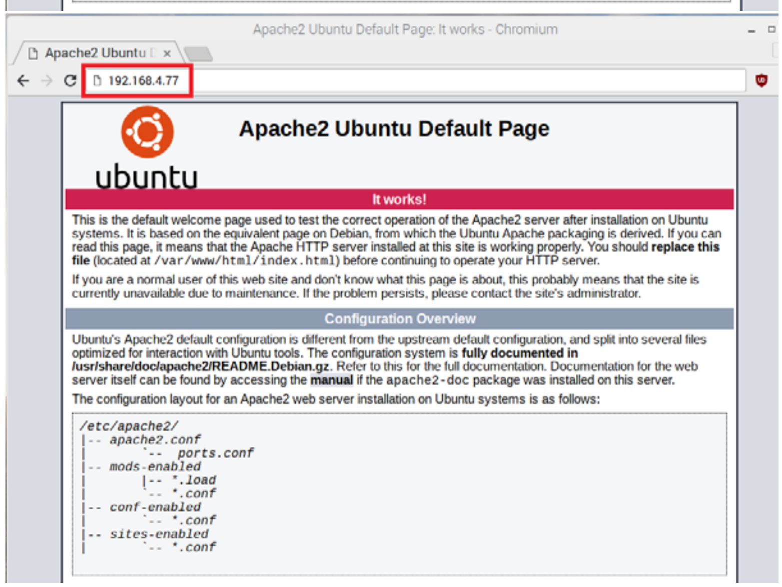
Test Case Field	Description
	***MUDC [INFO][handle_get_aclname:3194]--> Got URL from message <*https://mudfileserver/dnstest.json*> ***MUDC [INFO][query_policies_by_uri:1873]--> found the record <{ "_id" : { "$oid" : "5d51d0eb0ff2eb76576ee38b" }, "DACL_Name" : "ACS:CiscoSecure-Defined-ACL=mud-77797-v4fr.in", "DACL" : "[\"ip:inacl#10=permit tcp any host **192.168.4.7** range 80 80 syn ack\", \"ip:inacl#20=permit tcp any host **192.168.4.78** range 80 80 syn ack\", \"ip:inacl#30=permit tcp any host **192.168.4.77** range 80 80 syn ack\", \"ip:inacl#40=permit tcp any eq 22 any\", \"ip:inacl#41=permit udp any eq 68 any eq 67\", \"ip:inacl#42=permit udp any any eq 53\", \"ip:inacl#43=deny ip any any\"]", "URI" : "*https://mudfileserver/dnstest.json*" }> ***MUDC [INFO][query_policies_by_uri:1915]--> Response <{ "Cisco-AVPair": ["ACS:CiscoSecure-Defined-ACL=mud-77797-v4fr.in"] }> ***MUDC [INFO][mudc_construct_head:63]--> status_code: 200, content_len: 70, extra_headers: Content-Type: application/aclname ***MUDC [INFO][mudc_construct_head:80]--> HTTP header: HTTP/1.1 200 OK Content-Type: application/aclname Content-Length: 70 ***MUDC [INFO][query_policies_by_uri:1918]--> { "Cisco-AVPair": ["ACS:CiscoSecure-Defined-ACL=mud-77797-v4fr.in"] } ***MUDC [INFO][handle_get_aclname:3204]--> Got ACLs from the MUD URL **Switch/PEP:** Build1#**show access-lists** Extended IP access list mud-77797-v4fr.in 10 permit tcp any host **192.168.4.7** eq www ack syn 20 permit tcp any host **192.168.4.78** eq www ack syn 30 permit tcp any host **192.168.4.77** eq www ack syn

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>

```
40 permit tcp any eq 22 any
41 permit udp any eq bootpc any eq bootps
42 permit udp any any eq domain
43 deny ip any any
```

Procedure 4:





</td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

Test Case IoT-9-v6 is identical to test case IoT-9-v4 except that IoT-9-v6 uses IPv6 addresses rather than IPv4 addresses.

2.1.2.10 Test Case IoT-10-v4

Table 2-11: Test Case IoT-10-v4

Test Case Field	Description
Parent Requirements	(CR-12) The IoT DDoS example implementation shall include a MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed for the MUD file indicated by the MUD URL**.** The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.
Testable Requirements	(CR-12.a) The MUD manager shall check if the file associated with the MUD URL is present in its cache and shall determine that it is. (CR-12.a.1) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If so, the MUD manager shall apply the contents of the cached MUD file. (CR-12.a.2) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file. If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the cached MUD file for that device's MUD URL, assuming that the amount of time that has elapsed since the cached MUD file was retrieved is less than or equal to the number of hours in the file's cache-validity value. If the cache validity has expired for the respective file, the MUD manager should fetch a new MUD file from the MUD file server.
Associated Test Case(s)	N/A

Test Case Field	Description
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Ciscopi2.json*
Preconditions	1. All devices have been configured to use IPv4. 2. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 3. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 2.1.3.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Run test IoT-1-v4 (or IoT-1-v6). 2. Within 24 hours (i.e., within the cache-validity period for the MUD file) of running test IoT-1-v4 (or IoT-1-v6), remove the IoT device that was connected during test IoT-1-v4 (or IoT-1-v6) from the network. Ensure all traffic filters associated to IoT device have been removed, and reconnect it to the test network. This should set in motion the following series of steps, which should occur automatically. 3. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 4. The DHCP server receives the DHCPv4 message containing the IoT device's MUD URL. 5. The DHCP server offers an IP address lease to the newly connected IoT device. 6. The IoT device requests this IP address lease, which the DHCP server acknowledges. 7. The DHCP server sends the MUD URL to the MUD manager.

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>8. The MUD manager determines that it has this MUD file cached and checks that the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If the cache validity has been exceeded, the MUD manager will fetch a new MUD file. (Run the test both ways—with a cache-validity period that has expired and with one that has not.)
9. The MUD manager translates the MUD file’s contents into appropriate route filtering rules and installs these rules onto the MUD PEP for the IoT device in question so that this router/switch is now configured to enforce the policies specified in the MUD file.</td></tr>
<tr><td>Expected Results</td><td>The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device’s MUD file. The expected configuration should resemble the following.
Cache is valid (the MUD manager does NOT retrieve the MUD file from the MUD file server):

<code>Extended IP access list mud-81726-v4fr.in</code>
<code>   10 permit tcp any host 192.168.4.7 eq www ack syn</code>
<code>   20 permit tcp any host 192.168.10.104 eq www</code>
<code>   30 permit tcp any host 192.168.10.105 eq www</code>
<code>   50 permit tcp any 192.168.10.0 0.0.0.255 eq www</code>
<code>   60 permit tcp any 192.168.13.0 0.0.0.255 eq www</code>
<code>   70 permit tcp any 192.168.14.0 0.0.0.255 eq www</code>
<code>   80 permit tcp any eq 22 any</code>
<code>   81 permit udp any eq bootpc any eq bootps</code>
<code>   82 permit udp any any eq domain</code>
<code>   83 deny ip any any</code>

Cache is valid (the MUD manager does NOT retrieve the MUD file from the MUD file server):

<code>Extended IP access list mud-81726-v4fr.in</code>
<code>   10 permit tcp any host 192.168.4.7 eq www ack syn</code>
<code>   20 permit tcp any host 192.168.10.104 eq www</code>
<code>   30 permit tcp any host 192.168.10.105 eq www</code>
<code>   50 permit tcp any 192.168.10.0 0.0.0.255 eq www</code>
<code>   60 permit tcp any 192.168.13.0 0.0.0.255 eq www</code></td></tr>
</table>

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
   70 permit tcp any 192.168.14.0 0.0.0.255 eq www
   80 permit tcp any eq 22 any
   81 permit udp any eq bootpc any eq bootps
   82 permit udp any any eq domain
   83 deny ip any any
</pre>
Cache is not valid (the MUD manager does retrieve the MUD file from the MUD file server):
<pre>
Extended IP access list mud-81726-v4fr.in
   10 permit tcp any host 192.168.4.7 eq www ack syn
   20 permit tcp any host 192.168.10.104 eq www
   30 permit tcp any host 192.168.10.105 eq www
   50 permit tcp any 192.168.10.0 0.0.0.255 eq www
   60 permit tcp any 192.168.13.0 0.0.0.255 eq www
   70 permit tcp any 192.168.14.0 0.0.0.255 eq www
   80 permit tcp any eq 22 any
   81 permit udp any eq bootpc any eq bootps
   82 permit udp any any eq domain
   83 deny ip any any
</pre>
All protocol exchanges described in steps 1–9 above are expected to occur and can be viewed via Wireshark if desired. If the router/switch does not get configured in accordance with the MUD file, each exchange of DHCP and MUD-related protocol traffic should be viewed on the network via Wireshark to determine which transactions did not proceed as expected, and the observed and absent protocol exchanges should be described here.
</td></tr>
<tr><td>Actual Results</td><td>
MUD manager logs for valid cache:
<pre>
**MUDC [INFO][mudc_print_request_info:2185]--> print parsed
HTTP request header info
***MUDC [INFO][mudc_print_request_info:2186]--> request
method: POST
***MUDC [INFO][mudc_print_request_info:2187]--> request uri:
/getaclname
***MUDC [INFO][mudc_print_request_info:2188]--> local uri:
/getaclname
***MUDC [INFO][mudc_print_request_info:2189]--> http ver-
sion: 1.1
***MUDC [INFO][mudc_print_request_info:2190]--> query
string: (null)
***MUDC [INFO][mudc_print_request_info:2191]--> con-
tent_length: 27
</pre>
</td></tr>
</table>

Test Case Field	Description
	***MUDC [INFO][mudc_print_request_info:2192]--> remote ip addr: 0xe7719c38 ***MUDC [INFO][mudc_print_request_info:2193]--> remote port: 49344 ***MUDC [INFO][mudc_print_request_info:2194]--> remote_user: (null) ***MUDC [INFO][mudc_print_request_info:2195]--> is ssl: 0 ***MUDC [INFO][mudc_print_request_info:2199]--> header(0): name: <Host>, value: <127.0.0.1:8000> ***MUDC [INFO][mudc_print_request_info:2199]--> header(1): name: <User-Agent>, value: <FreeRADIUS 3.0.17> ***MUDC [INFO][mudc_print_request_info:2199]--> header(2): name: <Accept>, value: <*/*> ***MUDC [INFO][mudc_print_request_info:2199]--> header(3): name: <Content-Type>, value: <application/json> ***MUDC [INFO][mudc_print_request_info:2199]--> header(4): name: <X-FreeRADIUS-Section>, value: <authorize> ***MUDC [INFO][mudc_print_request_info:2199]--> header(5): name: <X-FreeRADIUS-Server>, value: <default> ***MUDC [INFO][mudc_print_request_info:2199]--> header(6): name: <Content-Length>, value: <27> ***MUDC [INFO][handle_get_aclname:2506]--> Mac address <b827ebeb6c8b> ***MUDC [INFO][fetch_uri_from_macaddr:1702]--> found the fields <{ "_id" : { "$oid" : "5c182c7edb40218cde918776" }, "URI" : "https://mudfileserver/ciscopi2" }> ***MUDC [INFO][fetch_uri_from_macaddr:1711]--> ============= Returning URI:https://mudfileserver/ciscopi2 ***MUDC [INFO][handle_get_aclname:2513]--> Found URI https://mudfileserver/ciscopi2 for MAC address b827ebeb6c8b ***MUDC [INFO][validate_muduri:2373]--> uri: https://mudfileserver/ciscopi2 ***MUDC [INFO][validate_muduri:2399]--> ip: mudfileserver, filename: ciscopi2 ***MUDC [INFO][handle_get_aclname:2558]--> Got URL from message <https://mudfileserver/ciscopi2> ***MUDC [INFO][query_policies_by_uri:1419]--> found the record <{ "_id" : { "$oid" : "5c182d9cdb40218cde91884a" }, "DACL_Name" : "ACS:CiscoSecure-Defined-ACL=mud-81726-v4fr.in", "DACL" : "[\"ip:inacl#10=permit tcp any host 192.168.4.7 range 80 80 syn ack\", \"ip:inacl#20=permit tcp any host 192.168.10.104 range 80 80\", \"ip:inacl#30=permit tcp any host 192.168.10.105 range 80 80\", \"ip:inacl#40=permit tcp any host 192.168.10.104 range 80 80\", \"ip:inacl#50=permit tcp any 192.168.10.0 0.0.0.255 range 80 80\", \"ip:inacl#60=permit tcp any 192.168.13.0 0.0.0.255 range 80 80\", \"ip:inacl#70=permit tcp any 192.168.14.0

Test Case Field	Description
	0.0.0.255 range 80 80\", \"ip:inacl#80=permit tcp any eq 22 any\", \"ip:inacl#81=permit udp any eq 68 any eq 67\", \"ip:inacl#82=permit udp any any eq 53\", \"ip:inacl#83=deny ip any any\"]", "URI" : "*https://mudfileserver/ciscopi2*", "VLAN" : 3 }> ***MUDC [INFO][query_policies_by_uri:1461]--> Response <{ "Cisco-AVPair": ["ACS:CiscoSecure-Defined-ACL=mud-81726-v4fr.in"], "Tunnel-Type": "VLAN", "Tunnel-Medium-Type": "IEEE-802", "Tunnel-Private-Group-Id": 3 }> ***MUDC [INFO][mudc_construct_head:135]--> status_code: 200, content_len: 160, extra_headers: Content-Type: application/aclname ***MUDC [INFO][mudc_construct_head:152]--> HTTP header: HTTP/1.1 200 OK Content-Type: application/aclname Content-Length: 160 ***MUDC [INFO][query_policies_by_uri:1464]--> { "Cisco-AVPair": ["ACS:CiscoSecure-Defined-ACL=mud-81726-v4fr.in"], "Tunnel-Type": "VLAN", "Tunnel-Medium-Type": "IEEE-802", "Tunnel-Private-Group-Id": 3 } ***MUDC [INFO][handle_get_aclname:2568]--> Got ACLs from the MUD URL **MUD manager logs for expired cache:** ***MUDC [INFO][mudc_print_request_info:2185]--> print parsed HTTP request header info ***MUDC [INFO][mudc_print_request_info:2186]--> request method: POST ***MUDC [INFO][mudc_print_request_info:2187]--> request uri: /getaclname ***MUDC [INFO][mudc_print_request_info:2188]--> local uri: /getaclname ***MUDC [INFO][mudc_print_request_info:2189]--> http version: 1.1 ***MUDC [INFO][mudc_print_request_info:2190]--> query string: (null) ***MUDC [INFO][handle_get_aclname:2506]--> Mac address <**b827ebeb6c8b**>

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
***MUDC [INFO][fetch_uri_from_macaddr:1702]--> found the fields <{ "_id" : { "$oid" : "5c182c7edb40218cde918776" }, "URI" : "<i>https://mudfileserver/ciscopi2</i>" }>

***MUDC [INFO][fetch_uri_from_macaddr:1711]--> ============= Returning URI:https://mudfileserver/ciscopi2

***MUDC [INFO][handle_get_aclname:2513]--> Found URI https://mudfileserver/ciscopi2 for MAC address b827ebeb6c8b

***MUDC [INFO][validate_muduri:2373]--> uri: https://mud-fileserver/ciscopi2
***MUDC [INFO][validate_muduri:2399]--> ip: mudfileserver, filename: ciscopi2
***MUDC [INFO][handle_get_aclname:2558]--> Got URL from message <https://mudfileserver/ciscopi2>

***MUDC [INFO][query_policies_by_uri:1399]--> Cache has expired

[Omitted for brevity]

***MUDC [STATUS][send_mudfs_request:2005]-->
Request URI <https://mudfileserver/ciscopi2>
</home/mudtester/mud-intermediate.pem>

*   Trying 192.168.4.5...
* TCP_NODELAY set
* Connected to mudfileserver (192.168.4.5) port 443 (#0)
* found 1 certificate in /home/mudtester/mud-intermediate.pem
* found 400 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_RSA_AES_256_GCM_SHA384
*   server certificate verification OK
*   server certificate status verification SKIPPED
*   common name: mudfileserver (matched)
*   server certificate expiration date OK
*   server certificate activation date OK
*   certificate public key: RSA
*   certificate version: #3
*   subject: C=US,ST=Maryland,L=Rockville,O=National Cybersecurity Center of Excellence - NIST,CN=mudfileserver
*   start date: Fri, 05 Oct 2018 00:00:00 GMT
*   expire date: Wed, 13 Oct 2021 12:00:00 GMT
*   issuer: C=US,O=DigiCert Inc,CN=DigiCert Test SHA2 Intermediate CA-1
*   compression: NULL
* ALPN, server did not agree to a protocol
> GET /ciscopi2 HTTP/1.1
</td></tr>
</table>

Test Case Field	Description
	`Host: mudfileserver` `Accept: */*` **[Omitted for brevity]**
Overall Results	Pass

Test case IoT-10-v6 is identical to test case IoT-10-v4 except that IoT-10-v6 tests requirement CR-1.a.2, whereas IoT-10-v4 tests requirement CR-1.a.1. Hence, as explained above, test IoT-10-v6 uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

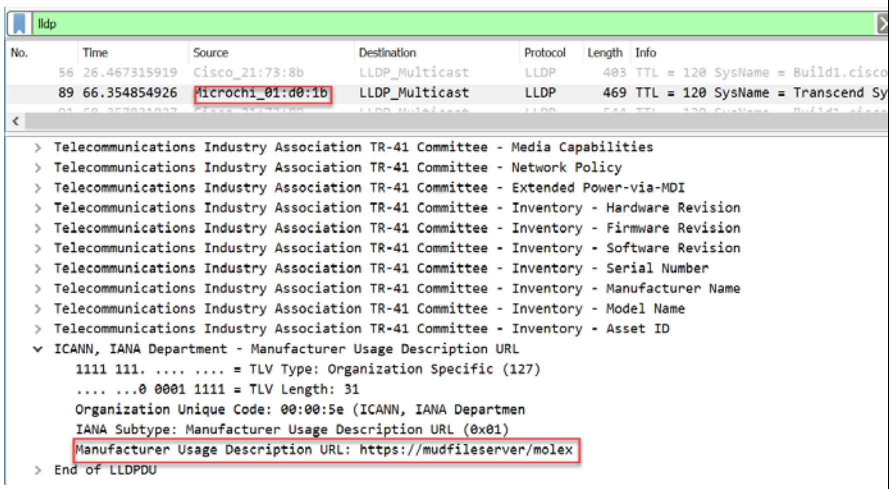
2.1.2.11 Test Case IoT-11-v4

Table 2-12: Test Case IoT-11-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, Link Layer Discovery Protocol [LLDP], or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL).
Testable Requirements	(CR-1.a) Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one MUD URL, in https scheme, within the DHCP transaction. (CR-1.a.1) The DHCP server shall be able to receive DHCPv4 DISCOVER and REQUEST with IANA code 161 (OPTION_MUD_URL_V4) from the MUD-enabled IoT device. OR (CR-1.b) Upon initialization, the MUD-enabled IoT device shall emit the MUD URL as an LLDP extension. (CR-1.b.1) The network service shall be able to process the MUD URL that is received as an LLDP extension.

Test Case Field	Description
Description	Shows that the IoT DDoS example implementation includes IoT devices that can emit a MUD URL via DHCP or LLDP
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1
IoT Device(s) Under Test	Raspberry Pi, Molex light engine, u-blox C027-G35
MUD File(s) Used	*Ciscopi2.json, molex.json, ublox.json*
Preconditions	Device has been developed to emit a MUD URL in a DHCP transaction
Procedure	1. Power on a device and connect it to the network. 2. Verify that the device emits a MUD URL in a DHCP transaction or LLDP message. a. Use Wireshark to capture a DHCP transaction with options present. b. Use Wireshark to capture an LLDP message with a MUD URL present in the LLDP frame.
Expected Results	DHCP transaction with MUD option 161 or LLDP TLV MUD extension enabled and MUD URL included

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td>Actual Results</td><td>Raspberry Pi (using DHCPv4):
2875 2931.939031… 0.0.0.0 255.255.255.255 DHCP 350 DHCP Discover - Transaction I
2877 2933.217946… 0.0.0.0 255.255.255.255 DHCP 350 DHCP Request - Transaction I
3174 3005.512734… 0.0.0.0 255.255.255.255 DHCP 350 DHCP Discover - Transaction I
3175 3005.513333… 0.0.0.0 255.255.255.255 DHCP 350 DHCP Request - Transaction I
Frame 2875: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits) on interface 0
Ethernet II, Src: Raspberr_eb:6c:8b (b8:27:eb:eb:6c:8b), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
User Datagram Protocol, Src Port: 68, Dst Port: 67
Dynamic Host Configuration Protocol (Discover)
Message type: Boot Request (1)
Hardware type: Ethernet (0x01)
Hardware address length: 6
Hops: 0
Transaction ID: 0x02523497
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0
Your (client) IP address: 0.0.0.0
Next server IP address: 0.0.0.0
Relay agent IP address: 0.0.0.0
Client MAC address: Raspberr_eb:6c:8b (b8:27:eb:eb:6c:8b)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
Option: (53) DHCP Message Type (Discover)
Option: (57) Maximum DHCP Message Size
Option: (55) Parameter Request List
Option: (161) Manufacturer Usage Description
Length: 30
MUDURL: https://mudfileserver/ciscopi2
Option: (255) End
Padding: 000000000000000000000000000000000000000000
u-blox C027-G35 (using DHCPv4):
bootp
No. Time Source Destination Protocol Length Info
149 88.162148385 0.0.0.0 255.255.255.255 DHCP 350 DHCP Discover - Transaction ID 0x53ef0545
151 88.167569034 0.0.0.0 255.255.255.255 DHCP 350 DHCP Request - Transaction ID 0x53ef0545
168 91.166593028 10.42.0.1 10.42.0.186 DHCP 342 DHCP Offer - Transaction ID 0x53ef0545
Hops: 0
Transaction ID: 0x53ef0545
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0
Your (client) IP address: 0.0.0.0
Next server IP address: 0.0.0.0
Relay agent IP address: 0.0.0.0
Client MAC address: Arm_f0:00:00 (00:02:f7:f0:00:00)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
Option: (53) DHCP Message Type (Discover)
Option: (57) Maximum DHCP Message Size
Option: (55) Parameter Request List
Option: (161) Manufacturer Usage Description
Length: 27
MUDURL: https://mudfileserver/ublox
Option: (255) End
Padding: 000000000000000000000000000000000000000000000000
Molex light engine (using LLDP):</td></tr>
</table>

Test Case Field	Description
	
Overall Results	Pass

2.1.3 MUD Files

This section contains the MUD files that were used in the Build 1 functional demonstration.

2.1.3.1 Ciscopi2.json

The complete Ciscopi2.json MUD file has been linked to this document. To access this MUD file please click the link below.

Ciscopi2.json

2.1.3.2 expiredcerttest.json

The complete expiredcerttest.json MUD file has been linked to this document. To access this MUD file please click the link below.

expiredcerttest.json

2.1.3.3 molex.json

The complete molex.json MUD file has been linked to this document. To access this MUD file please click the link below.

molex.json

2.1.3.4 ublox.json

The complete ublox.json MUD file has been linked to this document. To access this MUD file please click the link below.

ublox.json

2.1.3.5 dnstest.json

The complete dnstest.json MUD file has been linked to this document. To access this MUD file please click the link below.

dnstest.json

2.2 Demonstration of Non-MUD-Related Capabilities

In addition to supporting MUD, Build 1 supports capabilities with respect to device discovery, attribute identification, and monitoring. Table 2-13 lists the non-MUD-related capabilities that were demonstrated for Build 1. We use the letter "C" as a prefix for these functional capability identifiers in the table below because these capabilities are specific to Build 1, which uses Cisco equipment.

312 ### 2.2.1 Non-MUD-Related Functional Capabilities

313 **Table 2-13: Non-MUD-Related Functional Capabilities Demonstrated**

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
C-1	The IoT DDoS example implementation shall include a visibility component that can **detect, identify, categorize, and monitor the status of IoT devices** that are on the network.			CnMUD-13-v4, CnMUD-13-v6
C-1.a		The visibility component shall **detect and identify** the attributes and category of a newly connected IoT device.		CnMUD-13-v4, IoT-13-v6
C-1.a.1			The visibility component shall **monitor the status** of the IoT device (e.g., notice if the device goes offline).	CnMUD-13-v4, IoT-13-v6

314 ### 2.2.2 Exercises to Demonstrate the Above Non-MUD-Related Capabilities

315 This section contains the exercises that were performed to verify that Build 1 supports the non-MUD-
316 related capabilities listed in Table 2-13.

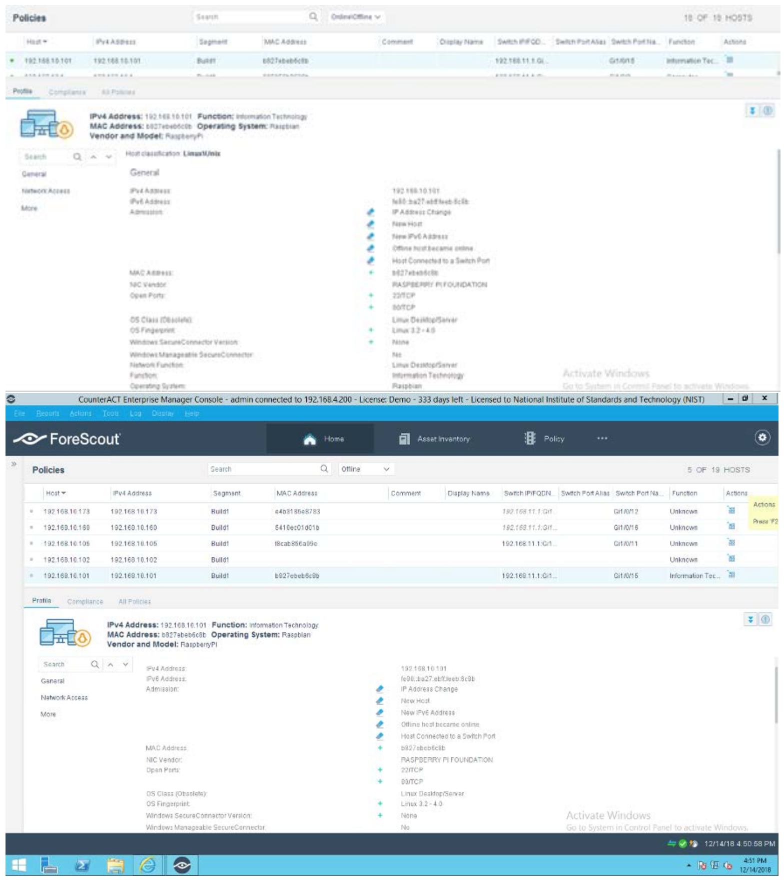
317 #### 2.2.2.1 Exercise CnMUD-13-v4

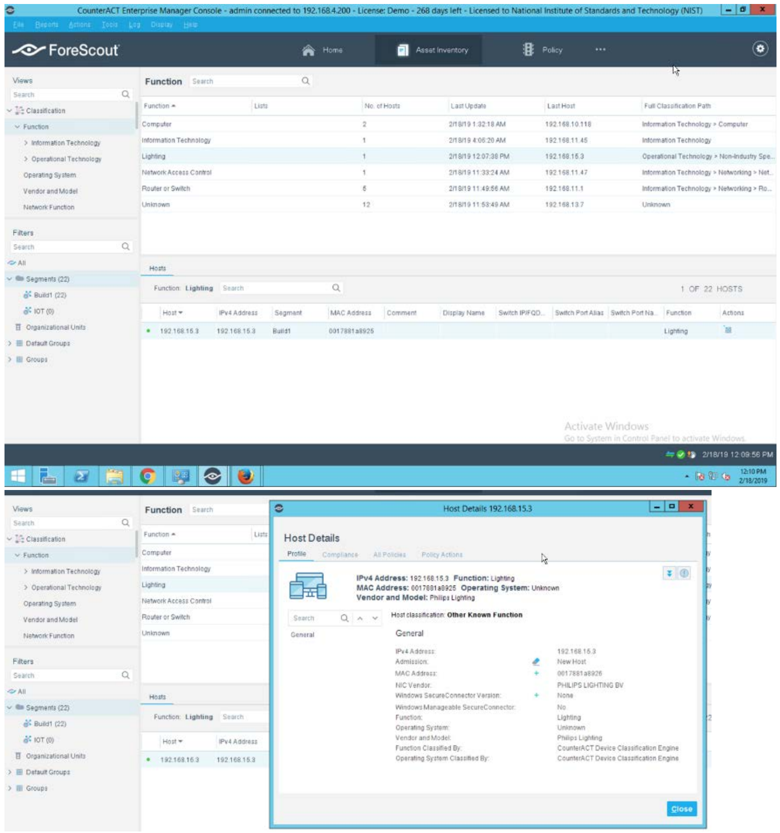
318 **Table 2-14: Exercise CnMUD-13-v4**

Test Case Field	Description
Parent Requirements	(C-1) The IoT DDoS example implementation shall include a visibility component that can detect, identify, categorize, and monitor the status of IoT devices that are on the network.
Testable Requirements	(C-1.a) The visibility component shall detect and identify the attributes and category of a newly connected IoT device. (C-1.a.1) The visibility component shall monitor the status of the IoT device (e.g., notice if the device goes offline).
Description	Shows that the IoT DDoS example implementation includes a visibility component that can perform the following actions. Upon connection of a live IoT device to the network, the device will be detected; identified in terms of attributes such as its IP address, operating system (OS), and device type; and continuously monitored as long as it remains live on the network. If the device becomes disconnected or turns off, this change of status will also be detected.
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, DE.AE-1, DE.CM-1
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	Not applicable for this test
Preconditions	The visibility component is up and running and attached to the network.
Procedure	1. Power on a device and connect it to the network. 2. Verify that the device is detected by the visibility component and that its type, address, OS, and other features are identified, and the device is categorized correctly.

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>3. Turn off the device.
4. Verify that its absence from the network is detected.
5. Power the device back on.
6. Verify that its presence is detected and its features are identified correctly.
7. Disconnect the device from the network.
8. Verify that its absence from the network is detected.</td></tr>
<tr><td>Expected Results</td><td>All expectations as enumerated in items 2, 4, 6, and 8 above are observed.</td></tr>
<tr><td>Actual Results</td><td>At Power-On:
<pre>pi@raspberrypi:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.10.101 netmask 255.255.255.0 broadcast
192.168.10.255
        ether b8:27:eb:eb:6c:8b txqueuelen 1000 (Ethernet)
        RX packets 9193 bytes 8208593 (7.8 MiB)
        RX errors 0 dropped 5 overruns 0 frame 0
        TX packets 7210 bytes 822414 (803.1 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 colli-
sions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 16 bytes 1467 (1.4 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 16 bytes 1467 (1.4 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 colli-
sions 0</pre>
Screenshot from Forescout:
IoT device status is indicated by green or gray light shown in the screen capture</td></tr>
</table>

Test Case Field	Description
	 **Categorizing IoT Device:** We tested this function with a connected light bulb. See the example screenshots below.

Test Case Field	Description
	
Overall Results	Pass

Test case CnMUD-13-v6 is identical to test case CnMUD-13-v4 except that test case CnMUD-13-v6 uses IPv6 and DHCPv6 instead of using IPv4 and DHCPv4.

3 Build 2

Build 2 uses equipment from MasterPeace Solutions Ltd., GCA, and ThreatSTOP. The MasterPeace Solutions Yikes! router, cloud service, and mobile application are used to support MUD as well as to perform device discovery on the network and to apply additional traffic rules to both MUD-capable and non-MUD-capable devices based on device manufacturer and model. The GCA Quad9 DNS Service and the ThreatSTOP Threat MUD File Server are used to support threat signaling.

3.1 Evaluation of MUD-Related Capabilities

The functional evaluation that was conducted to verify that Build 2 conforms to the MUD specification was based on the Build 2-specific requirements listed in Table 3-1.

3.1.1 Requirements

Table 3-1: MUD Use Case Functional Requirements

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-1	The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled **IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL).**			IoT-1-v4, IoT-1-v6, IoT-11-v4, IoT-11-v6
CR-1.a		Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one **MUD URL, in https scheme,**		IoT-1-v4, IoT-1-v6, IoT-11-v4, IoT-11-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		within the DHCP transaction.		
CR-1.a.1			The DHCP server shall be able to receive **DHCPv4 DISCOVER and REQUEST with IANA code 161** (OPTION_MUD_URL_V4) from the MUD-enabled IoT device.	IoT-1-v4, IoT-11-v4
CR-1.a.2			The DHCP server shall be able to receive **DHCPv6 Solicit and Request with IANA code 112** (OPTION_MUD_URL_V6) from the MUD-enabled IoT device.	IoT-1-v6, IoT-11-v6
CR-2	The IoT DDoS example implementation shall include the capability for the MUD URL **to be provided to a MUD manager.**			IoT-1-v4, IoT-1-v6
CR-2.a		The DHCP server shall **assign an IP address lease** to the MUD-enabled IoT device.		IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-2.a.1			The MUD-enabled IoT device shall **receive the IP address.**	IoT-1-v4, IoT-1-v6
CR-2.b		**The DHCP server shall** receive the DHCP message and **extract the MUD URL, which is then passed to the MUD manager.**		IoT-1-v4, IoT-1-v6
CR-2.b.1			**The MUD manager shall receive the MUD URL.**	IoT-1-v4, IoT-1-v6
CR-3	The IoT DDoS example implementation shall include a **MUD manager that can request a MUD file and signature from a MUD file server.**			IoT-1-v4, IoT-1-v6
CR-3.a		The MUD manager shall use the GET method (RFC 7231) to **request MUD and signature files** (per RFC 7230) from the MUD file server and can **validate the MUD file server's TLS certificate** by using the rules in RFC 2818.		IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-3.a.1			**The MUD file server shall receive the https request from the MUD manager.**	IoT-1-v4, IoT-1-v6
CR-3.b		**The MUD manager** shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it **cannot validate the MUD file server's TLS certificate** by using the rules in RFC 2818.		IoT-2-v4, IoT-2-v6
CR-3.b.1			**The MUD manager shall drop the connection** to the MUD file server.	IoT-2-v4, IoT-2-v6
CR-3.b.2			**The MUD manager shall send locally defined policy to the router or switch** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-2-v4, IoT-2-v6
CR-4	The IoT DDoS example implementation shall include a **MUD file server that can**			IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	serve a MUD file and signature to the MUD manager.			
CR-4.a		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the **certificate had not expired.**		IoT-1-v4, IoT-1-v6
CR-4.b		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** was valid at the time of signing, i.e., the **certificate had already expired when it was used to sign the MUD file.**		IoT-3-v4, IoT-3-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-4.b.1			The MUD manager shall cease to process the MUD file.	IoT-3-v4, IoT-3-v6
CR-4.b.2			The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-3-v4, IoT-3-v6
CR-5	The IoT DDoS example implementation shall include a **MUD manager** that **can translate local network configurations based on the MUD file.**			IoT-1-v4, IoT-1-v6
CR-5.a		**The MUD manager shall successfully validate the signature of the MUD file.**		IoT-1-v4, IoT-1-v6
CR-5.a.1			The MUD manager, after validation of the MUD file signature, shall **check for an existing MUD file and translate abstractions in the MUD file to router or switch configurations.**	IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-5.a.2			The MUD manager shall **cache** this newly received MUD file.	IoT-10-v4, IoT-10-v6
CR-5.b		The MUD manager shall attempt to validate the signature of the **MUD file,** but the **signature validation fails** (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason).		IoT-4-v4, IoT-4-v6
CR-5.b.1			**The MUD manager shall cease processing the MUD file.**	IoT-4-v4, IoT-4-v6
CR-5.b.2			**The MUD manager shall send locally defined policy to the router or switch** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-4-v4, IoT-4-v6
CR-6	The IoT DDoS example implementation shall include a			IoT-1-v4, IoT-1-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	MUD manager that can configure the MUD PEP, i.e., the router or switch nearest the MUD-enabled IoT device that emitted the URL.			
CR-6.a		**The MUD manager shall install a router configuration** on the router or switch nearest the MUD-enabled IoT device that emitted the URL.		IoT-1-v4, IoT-1-v6
CR-6.a.1			**The router or switch shall have been configured to enforce the route filter sent by the MUD manager.**	IoT-1-v4, IoT-1-v6
CR-7	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file.**			IoT-5-v4, IoT-5-v6
CR-7.a		The MUD-enabled IoT device shall attempt to **initiate outbound traffic to approved internet services.**		IoT-5-v4, IoT-5-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-7.a.1			The router or switch shall receive the attempt and shall **allow it to pass** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-7.b		An approved **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4, IoT-5-v6
CR-7.b.1			The router or switch shall receive the attempt and shall **allow it to pass** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8	The IoT DDoS example implementation shall **deny communications from a MUD-enabled IoT device to unapproved internet services** (i.e., services that are denied by virtue of not being explicitly approved).			IoT-5-v4, IoT-5-v6
CR-8.a		The MUD-enabled IoT device shall **attempt to initiate outbound traffic to unapproved** (implicitly denied) **internet services.**		IoT-5-v4, IoT-5-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-8.a.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8.b		**An unapproved** (implicitly denied) **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4, IoT-5-v6
CR-8.b.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8.c		The MUD-enabled IoT device shall initiate communications to an internet service that is **approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device.**		IoT-5-v4, IoT-5-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-8.c.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-8.d		An internet service shall initiate communications to a MUD-enabled device that is **approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service.**		IoT-5-v4, IoT-5-v6
CR-8.d.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4, IoT-5-v6
CR-9	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate laterally with devices that are approved** in the MUD file.			IoT-6-v4, IoT-6-v6
CR-9.a		The MUD-enabled IoT device shall **attempt**		IoT-6-v4, IoT-6-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		to initiate lateral traffic to approved devices.		
CR-9.a.1			**The router or switch shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-9.b		An approved device **shall attempt to initiate a lateral connection to the MUD-enabled IoT device.**		IoT-6-v4, IoT-6-v6
CR-9.b.1			**The router or switch shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-10	The IoT DDoS example implementation shall **deny lateral communications from a MUD-enabled IoT device to devices that are not approved** in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).			IoT-6-v4, IoT-6-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-10.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to unapproved** (implicitly denied) **devices.**		IoT-6-v4, IoT-6-v6
CR-10.a.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-10.b		**An unapproved** (implicitly denied) **device shall attempt to initiate a lateral connection** to the MUD-enabled IoT device.		IoT-6-v4, IoT-6-v6
CR-10.b.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4, IoT-6-v6
CR-11	If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, **the DHCP server must notify the MUD manager of any corresponding change to the DHCP state** of			IoT-7-v4, IoT-7-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	the MUD-enabled IoT device, and the MUD manager should **remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.**			
CR-11.a		The MUD-enabled IoT **device shall explicitly release the IP address lease** (i.e., it sends a DHCP release message to the DHCP server).		IoT-7-v4, IoT-7-v6
CR-11.a.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has been released.**	IoT-7-v4, IoT-7-v6
CR-11.a.2			**The MUD manager should remove all policies** associated with the disconnected IoT device that had been configured on the MUD PEP router/switch.	IoT-7-v4, IoT-7-v6
CR-11.b		The MUD-enabled IoT **device's IP address lease shall expire.**		IoT-8-v4, IoT-8-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-11.b.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has expired.**	IoT-8-v4, IoT-8-v6
CR-11.b.2			**The MUD manager should remove all policies** associated with the affected IoT device that had been configured on the MUD PEP router/switch.	IoT-8-v4, IoT-8-v6
CR-12	The IoT DDoS example implementation shall include a **MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed** for the MUD file indicated by the MUD URL**. The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.**			IoT-10-v4, IoT-10-v6
CR-12.a		The MUD manager shall check if the file associated with the **MUD URL is present in its cache** and shall determine that it is.		IoT-10-v4, IoT-10-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-12.a.1			The MUD manager shall **check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager shall apply the contents of the cached MUD file.	IoT-10-v4, IoT-10-v6
CR-12.a.2			The MUD manager **shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.	IoT-10-v4, IoT-10-v6
CR-13	The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the MUD PEP			IoT-9-v4, IoT-9-v6

Capability Requirement (CR-ID)	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	router/switch will get configured with **all possible instantiations of that rule,** insofar as **each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the MUD PEP router/switch.**			
CR-13.a		The MUD file for a device shall contain a rule involving a **domain that can resolve to multiple IP addresses** when queried by the MUD PEP router/switch. **An ACL for permitting access to each of those IP addresses will be inserted into the MUD PEP router/switch** for the device in question, and the device will be permitted to communicate with all of those IP addresses.		IoT-9-v4, IoT-9-v6
CR-13.a.1			IPv4 addressing is used on the network.	IoT-9-v4
CR-13.a.2			IPv6 addressing is used on the network.	IoT-9-v6

3.1.2 Test Cases

3.1.2.1 Test Case IoT-1-v4

This section contains the test cases that were used to verify that Build 2 met the requirements listed in Table 3-1.

Table 3-2: Test Case IoT-1-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL). (CR-2) The IoT DDoS example implementation shall include the capability for the MUD URL to be provided to a MUD manager. (CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server. (CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager. (CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file. (CR-6) The IoT DDoS example implementation shall include a MUD manager that can configure the router or switch nearest the MUD-enabled IoT device that emitted the URL.
Testable Requirements	(CR-1.a) Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one MUD URL, in https scheme, within the DHCP transaction. (CR-1.a.1) The DHCP server shall be able to receive DHCPv4 DISCOVER and/or REQUEST with IANA code 161 (OPTION_MUD_URL_V4) from the MUD-enabled IoT device. (NOTE: Test IoT-1-v6 does not test this requirement; instead, it tests CR-1.a.2, which pertains to DHCPv6 rather than DHCPv4.) (CR-2.a) The DHCP server shall assign an IP address lease to the MUD-enabled IoT device. (CR-2.a.1) The MUD-enabled IoT device shall receive the IP address.

Test Case Field	Description
	(CR-2.b) The DHCP server shall receive the DHCP message and extract the MUD URL, which is then passed to the MUD manager. (CR-2.b.1) The MUD manager shall receive the MUD URL. (CR-3.a) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server and can validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.a.1) The MUD file server shall receive the https request from the MUD manager. (CR-4.a) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the certificate had not expired. (CR-5.a) The MUD manager shall successfully validate the signature of the MUD file. (CR-5.a.1) The MUD manager, after validation of the MUD file signature, shall check for an existing MUD file and translate abstractions in the MUD file to router or switch configurations. (CR-6.a) The MUD manager shall install a router configuration on the router or switch nearest the MUD-enabled IoT device that emitted the URL. (CR-6.a.1) The router or switch shall have been configured to enforce the route filter sent by the MUD manager.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file, assuming the MUD file has a valid signature and is served from a MUD file server that has a valid TLS certificate
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.PT-3, PR.DS-2

Test Case Field	Description
IoT Device(s) Under Test	Raspberry Pi (1)
MUD File(s) Used	*Yikesmain.json*
Preconditions	1. This MUD file is not currently cached at the MUD manager. 2. The device's MUD file has a valid signature that was signed by a certificate that had not yet expired, and it is being hosted on a MUD file server that has a valid TLS certificate. 3. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 4. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 3.1.3.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a MUD URL in a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server offers an IP address lease to the newly connected IoT device. 3. The IoT device requests this IP address lease, which the DHCP server acknowledges. 4. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 5. The DHCP service extracts the MUD URL. 6. The MUD URL is then provided to the MUD manager.

Test Case Field	Description
	7. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, requests and receives the MUD file and signature from the MUD file server, validates the MUD file's signature, and translates the MUD file's contents into appropriate route filtering rules. The MUD manager installs these rules onto the MUD PEP for the IoT device in question so that this router/switch is now configured to enforce the policies specified in the MUD file.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. The expected configuration should resemble the following: `config rule` `        option enabled   '1'` `        option name      'mud_192.168.20.222_main-pi-` `Build2_cl0-frdev'` `        option target    ACCEPT` `        option src       lan` `        option dest      wan` `        option proto     tcp` `        option family    ipv4` `        option src_ip     192.168.20.222` `        option dest_ip    198.71.233.87` `        option dest_port  443:443` `config rule` `        option enabled   '1'` `        option name      'mud_192.168.20.222_main-pi-` `Build2_cl0-todev'` `        option target    ACCEPT` `        option src       wan` `        option dest      lan` `        option proto     tcp` `        option family    ipv4` `        option src_ip     198.71.233.87` `        option dest_ip    192.168.20.222` `        option dest_port  443:443` `config rule` `        option enabled   '1'` `        option name      'mud_192.168.20.222_main-pi-` `Build2_cl1-frdev'`

Test Case Field	Description
	`option target ACCEPT` `option src lan` `option dest wan` `option proto tcp` `option family ipv4` `option src_ip 192.168.20.222` `option dest_ip 192.168.4.7` `option dest_port 80:80` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_cl1-todev'` `option target ACCEPT` `option src wan` `option dest lan` `option proto tcp` `option family ipv4` `option src_ip 192.168.4.7` `option dest_ip 192.168.20.222` `option dest_port 80:80` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev'` `option target ACCEPT` `option src lan` `option dest wan` `option proto tcp` `option family ipv4` `option src_ip 192.168.20.222` `option dest_ip 99.84.216.69` `option dest_port 443:443` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev'` `option target ACCEPT` `option src lan` `option dest wan` `option proto tcp` `option family ipv4` `option src_ip 192.168.20.222` `option dest_ip 99.84.216.65` `option dest_port 443:443`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
config rule
       option enabled   '1'
       option name   'mud_192.168.20.222_main-pi-
Build2_cl2-frdev'
       option target   ACCEPT
       option src   lan
       option dest   wan
       option proto   tcp
       option family   ipv4
       option src_ip   192.168.20.222
       option dest_ip   99.84.216.79
       option dest_port 443:443

config rule
       option enabled   '1'
       option name   'mud_192.168.20.222_main-pi-
Build2_cl2-frdev'
       option target   ACCEPT
       option src   lan
       option dest   wan
       option proto   tcp
       option family   ipv4
       option src_ip   192.168.20.222
       option dest_ip   99.84.216.27
       option dest_port 443:443

config rule
       option enabled   '1'
       option name   'mud_192.168.20.222_main-pi-
Build2_cl2-todev'
       option target   ACCEPT
       option src   wan
       option dest   lan
       option proto   tcp
       option family   ipv4
       option src_ip   99.84.216.27
       option dest_ip   192.168.20.222
       option dest_port 443:443

config rule
       option enabled   '1'
       option name   'mud_192.168.20.222_main-pi-
Build2_cl2-todev'
       option target   ACCEPT
       option src   wan
       option dest   lan
       option proto   tcp
       option family   ipv4
</pre>
</td></tr>
</table>

Test Case Field	Description
	`option src_ip 99.84.216.79` `option dest_ip 192.168.20.222` `option dest_port 443:443` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_cl2-todev'` `option target ACCEPT` `option src wan` `option dest lan` `option proto tcp` `option family ipv4` `option src_ip 99.84.216.65` `option dest_ip 192.168.20.222` `option dest_port 443:443` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_cl2-todev'` `option target ACCEPT` `option src wan` `option dest lan` `option proto tcp` `option family ipv4` `option src_ip 99.84.216.69` `option dest_ip 192.168.20.222` `option dest_port 443:443` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_ent0-frdev'` `option target ACCEPT` `option src lan` `option dest wan` `option proto tcp` `option family ipv4` `option src_ip 192.168.20.222` `option dest_ip 172.217.164.132` `option dest_port 443:443` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_ent0-frdev'` `option target ACCEPT`

Test Case Field	Description
	`` option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 0.0.0.0 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_ent0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 172.217.164.132 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_ent0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 0.0.0.0 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_loc0-frdev' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.222 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- ```

Test Case Field	Description
	Build2_loc0-todev' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip any option dest_ip 192.168.20.222 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_man0-frdev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.222 option ipset www_gmail_com-SMTD option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_man0-todev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option ipset www_gmail_com-SMFD option dest_ip 192.168.20.222 option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_myctl0-frdev' option target ACCEPT option src lan option dest wan option proto all option family ipv4 option src_ip 192.168.20.222 option dest_ip 192.168.20.101 config rule

Test Case Field	Description
	`option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_myctl0-todev'` `option target ACCEPT` `option src wan` `option dest lan` `option proto all` `option family ipv4` `option src_ip 192.168.20.101` `option dest_ip 192.168.20.222` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_myman0-frdev-SM'` `option target ACCEPT` `option src lan` `option dest lan` `option proto udp` `option family ipv4` `option src_ip 192.168.20.222` `option ipset mudfiles_nist_getyikes_com-SMTD` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_myman0-todev-SM'` `option target ACCEPT` `option src lan` `option dest lan` `option proto udp` `option family ipv4` `option ipset mudfiles_nist_getyikes_com-SMFD` `option dest_ip 192.168.20.222` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_REJECT-ALL-LOCAL-FROM'` `option target REJECT` `option src lan` `option dest lan` `option proto all` `option family ipv4` `option src_ip 192.168.20.222` `config rule` `option enabled '1'`

Test Case Field	Description
	<pre> option name 'mud_192.168.20.222_main-pi- Build2_REJECT-ALL-LOCAL-TO' option target REJECT option src lan option dest lan option proto all option family ipv4 option src_ip any option dest_ip 192.168.20.222 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_REJECT-ALL' option target REJECT option src lan option dest wan option proto all option family ipv4 option src_ip 192.168.20.222 # OSMUD end</pre> All protocol exchanges described in steps 1–7 above are expected to occur and can be viewed via Wireshark if desired. If the router/switch does not get configured in accordance with the MUD file, each exchange of DHCP and MUD-related protocol traffic should be viewed on the network via Wireshark to determine which transactions did not proceed as expected, and the observed and absent protocol exchanges should be described here.
Actual Results	**Procedures 1–3:** `pi@main-pi-Build2:~$ sudo dhclient -v -i eth0` `sudo: unable to resolve host main-pi-Build2: Connection refused` `Internet Systems Consortium DHCP Client 4.3.5` `Copyright 2004-2016 Internet Systems Consortium.` `All rights reserved.` `For info, please visit https://www.isc.org/software/dhcp/` `RTNETLINK answers: Operation not possible due to RF-kill` `Listening on LPF/wlan0/b8:27:eb:be:39:de` `Sending on   LPF/wlan0/b8:27:eb:be:39:de` `Listening on LPF/eth0/b8:27:eb:eb:6c:8b` `Sending on   LPF/eth0/b8:27:eb:eb:6c:8b`

Test Case Field	Description
	Sending on Socket/fallback **DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 4** **DHCPREQUEST of 192.168.20.222 on eth0 to 255.255.255.255 port 67** **DHCPOFFER of 192.168.20.222 from 192.168.20.1** **DHCPACK of 192.168.20.222 from 192.168.20.1** Too few arguments. Too few arguments. **bound to 192.168.20.222** -- renewal in 1800 seconds. **Procedures 4–5:** **dhcpmasq.txt** 2019-07-15T20:27:57Z|OLD|Wired|DHCP|-|MUD|-|-|ba:47:a1:7d:60:44|192.168.20.148|| 2019-07-15T20:28:01Z|OLD|NIST 5|DHCP|-|MUD|-|-|18:b4:30:50:98:38|192.168.20.203|| 2019-07-15T20:28:08Z|OLD|NIST 2.4|DHCP|-|MUD|-|-|d0:73:d5:28:08:2a|192.168.20.202|| 2019-07-15T20:28:11Z|OLD|Wired|DHCP|-|MUD|-|-|b8:27:eb:95:55:fe|192.168.20.232|raspberrypi| **2019-07-15T20:28:31Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|**https://mudfiles.nist.getyikes.com/yikesmain.json|-|b8:27:eb:eb:6c:8b|192.168.20.222|main-pi-Build2| 2019-07-15T20:28:42Z|NEW|NIST 5|DHCP|1,28,2,121,15,6,12,40,41,42,26,119,3,121,249,33,252,42|MUD|-|-|80:00:0b:ef:81:70|192.168.20.238|| **Procedure 6:** **MUD MANAGER:** **2019-07-15 20:28:32 DEBUG::GENERAL::2019-07-15T20:28:31Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|**https://mudfiles.nist.getyikes.com/yikesmain.json|-|b8:27:eb:eb:6c:8b|192.168.20.222|main-pi-Build2| 2019-07-15 20:28:32 DEBUG::GENERAL::Executing on dhcpmasq info 2019-07-15 20:28:32 INFO::GENERAL::NEW Device Action: IP: 192.168.20.222, MAC: b8:27:eb:eb:6c:8b 2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() doing it now.... 2019-07-15 20:28:32 DEBUG::COMMUNICATION::https://mudfiles.nist.getyikes.com/yikesmain.

Test Case Field	Description
	json 2019-07-15 20:28:32 DEBUG::COMMUNICATION::Found HTTPS 2019-07-15 20:28:32 DEBUG::COMMUNICATION::in write data 2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() success 2019-07-15 20:28:32 DEBUG::COMMUNICATION::MUD File Server returned success state. 2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() doing it now.... 2019-07-15 20:28:32 DEBUG::COMMUNICATION::https://mudfiles.nist.getyikes.com/yikesmain.p7s 2019-07-15 20:28:32 DEBUG::COMMUNICATION::Found HTTPS 2019-07-15 20:28:32 DEBUG::COMMUNICATION::in write data **2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() success** **2019-07-15 20:28:32 DEBUG::COMMUNICATION::MUD File Server returned success state.** **2019-07-15 20:28:32 DEBUG::MUD_FILE_OPERATIONS::IN ****NEW**** MUD and SIG FILE RETRIEVED!!!** **2019-07-15 20:28:32 DEBUG::GENERAL::IN ****NEW**** validateMudFileWithSig()** **2019-07-15 20:28:32 DEBUG::GENERAL::openssl cms -verify -in /etc/osmud/state/mudfiles/yikesmain.p7s -inform DER -content /etc/osmud/state/mudfiles/yikesmain.json -purpose any > /dev/null** 2019-07-15 20:28:32 DEBUG::GENERAL::IN ****NEW**** executeMudWithDhcpContext() 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_mud_db_entry.sh -d /etc/osmud/state/mudfiles/mudStateFile.txt -i 192.168.20.222 -m b8:27:eb:eb:6c:8b -c main-pi-Build2 -u https://mudfiles.nist.getyikes.com/yikesmain.json -f /etc/osmud/state/mudfiles/yikesmain.json 2019-07-15 20:28:32 DEBUG::GENERAL::rm -f /tmp/osmud/* 2019-07-15 20:28:32 DEBUG::GENERAL::cp /etc/osmud/state/ipSets/* /tmp/osmud 2019-07-15 20:28:32 WARNING::DEVICE_INTERFACE::The URL in the MUD file does not match the URL used to download the MUD FILE 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/remove_ip_fw_rule.sh -i 192.168.20.222 -m b8:27:eb:eb:6c:8b -d /tmp/osmud 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/remove_from_ipset.sh -d /tmp/osmud -i 192.168.20.222 2019-07-15 20:28:32

Test Case Field	Description
	DEBUG::GENERAL::/etc/osmud/add_to_ipset.sh -d /tmp/osmud -a mudfiles.nist.getyikes.com -n SM -i 192.168.20.222 -c main-pi-Build2 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing ACL-DNS *from* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::www.osmud.org 2019-07-15 20:28:32 DEBUG::GENERAL::198.71.233.87 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 198.71.233.87 -b 443:443 -p tcp -n cl0-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing ACL-DNS *from* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::us.dlink.com 2019-07-15 20:28:32 DEBUG::GENERAL::192.168.4.7 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 192.168.4.7 -b 80:80 -p tcp -n cl1-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing ACL-DNS *from* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::www.trytechy.com 2019-07-15 20:28:32 DEBUG::GENERAL::99.84.216.69 2019-07-15 20:28:32 DEBUG::GENERAL::99.84.216.65 2019-07-15 20:28:32 DEBUG::GENERAL::99.84.216.79 2019-07-15 20:28:32 DEBUG::GENERAL::99.84.216.27 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 99.84.216.69 -b 443:443 -p tcp -n cl2-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 99.84.216.65 -b 443:443 -p tcp -n cl2-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 99.84.216.79 -b 443:443 -p tcp -n cl2-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32

Test Case Field	Description
	DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 99.84.216.27 -b 443:443 -p tcp -n cl2-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 WARNING::DEVICE_INTERFACE::Processing CONTROLLER *from* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::www.google.com 2019-07-15 20:28:32 DEBUG::GENERAL::172.217.164.132 2019-07-15 20:28:32 DEBUG::GENERAL::0.0.0.0 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 172.217.164.132 -b 443:443 -p tcp -n ent0-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 0.0.0.0 -b 443:443 -p tcp -n ent0-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 WARNING::DEVICE_INTERFACE::Processing MY_CONTROLLER *from* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::yikes.example.com 2019-07-15 20:28:32 DEBUG::GENERAL::192.168.20.101 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 192.168.20.101 -b any -p all -n myctl0-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing LOCAL_NETWORK *to* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i 192.168.20.222 -a any -j any -b any -p tcp -n loc0-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing MANUFACTURER *from* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i 192.168.20.222 -a any -e www.gmail.com-SMTD -b 80:80 -p tcp -n man0-frdev-SM -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing SAME_MANUFACTURER *from* THING ace rule. 2019-07-15 20:28:32

Test Case Field	Description
	DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i 192.168.20.222 -a any -e mudfiles.nist.getyikes.com-SMTD -b any -p udp -n myman0-frdev-SM -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Successfully installed fromAccess rule. 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing DNS-ACL *to* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::www.osmud.org 2019-07-15 20:28:32 DEBUG::GENERAL::198.71.233.87 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 198.71.233.87 -a any -j 192.168.20.222 -b 443:443 -p tcp -n cl0-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing DNS-ACL *to* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::us.dlink.com 2019-07-15 20:28:32 DEBUG::GENERAL::192.168.4.7 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 192.168.4.7 -a any -j 192.168.20.222 -b 80:80 -p tcp -n cl1-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:32 INFO::DEVICE_INTERFACE::Processing DNS-ACL *to* ace rule. 2019-07-15 20:28:32 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:32 DEBUG::GENERAL::www.trytechy.com 2019-07-15 20:28:33 DEBUG::GENERAL::99.84.216.27 2019-07-15 20:28:33 DEBUG::GENERAL::99.84.216.79 2019-07-15 20:28:33 DEBUG::GENERAL::99.84.216.65 2019-07-15 20:28:33 DEBUG::GENERAL::99.84.216.69 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 99.84.216.27 -a any -j 192.168.20.222 -b 443:443 -p tcp -n cl2-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 99.84.216.79 -a any -j 192.168.20.222 -b 443:443 -p tcp -n cl2-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 99.84.216.65 -a any -j 192.168.20.222 -b 443:443 -p

Test Case Field	Description
	tcp -n cl2-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 99.84.216.69 -a any -j 192.168.20.222 -b 443:443 -p tcp -n cl2-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 WARNING::DEVICE_INTERFACE::Processing CONTROLLER *to* ace rule. 2019-07-15 20:28:33 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:33 DEBUG::GENERAL::www.google.com 2019-07-15 20:28:33 DEBUG::GENERAL::172.217.164.132 2019-07-15 20:28:33 DEBUG::GENERAL::0.0.0.0 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 172.217.164.132 -a any -j 192.168.20.222 -b 443:443 -p tcp -n ent0-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 0.0.0.0 -a any -j 192.168.20.222 -b 443:443 -p tcp -n ent0-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 WARNING::DEVICE_INTERFACE::Processing MY_CONTROLLER *to* ace rule. 2019-07-15 20:28:33 DEBUG::GENERAL::Starting DNS lookup 2019-07-15 20:28:33 DEBUG::GENERAL::yikes.example.com 2019-07-15 20:28:33 DEBUG::GENERAL::192.168.20.101 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s wan -d lan -i 192.168.20.101 -a any -j 192.168.20.222 -b any -p all -n myctl0-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 INFO::DEVICE_INTERFACE::Processing LOCAL_NETWORK *to* ace rule. 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i any -a any -j 192.168.20.222 -b any -p tcp -n loc0-todev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 INFO::DEVICE_INTERFACE::Processing (TBD) MANUFACTURER *to* ace rule. 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -j 192.168.20.222 -a any -e www.gmail.com-SMFD -b 80:80 -p tcp -n man0-todev-SM -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222

Test Case Field	Description
	2019-07-15 20:28:33 INFO::DEVICE_INTERFACE::Processing SAME_MANUFACTURER *to* THING ace rule. 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -j 192.168.20.222 -a any -e mudfiles.nist.getyikes.com-SMFD -b any -p udp -n myman0-todev-SM -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 INFO::DEVICE_INTERFACE::Successfully installed toAccess rule. 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j any -b any -p all -n REJECT-ALL -t REJECT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i 192.168.20.222 -a any -j any -b any -p all -n REJECT-ALL-LOCAL-FROM -t REJECT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i any -a any -j 192.168.20.222 -b any -p all -n REJECT-ALL-LOCAL-TO -t REJECT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/commit_ip_fw_rules.sh -d /etc/osmud/state/ipSets -t /tmp/osmud 2019-07-15 20:28:33 DEBUG::GENERAL::**Success returned from for transaction** **Procedure 7:** **Router/PEP:** config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl0-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 198.71.233.87 option dest_port 443:443 config rule option enabled '1'

Test Case Field	Description
	<pre> option name 'mud_192.168.20.222_main-pi- Build2_cl0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 198.71.233.87 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_cl1-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 192.168.4.7 option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_cl1-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 192.168.4.7 option dest_ip 192.168.20.222 option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_cl2-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 99.84.216.69 option dest_port 443:443</pre>

Test Case Field	Description
	config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 99.84.216.65 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 99.84.216.79 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 99.84.216.27 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4

Test Case Field	Description
	`` option src_ip 99.84.216.27 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_cl2-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 99.84.216.79 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_cl2-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 99.84.216.65 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_cl2-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 99.84.216.69 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_ent0-frdev' option target ACCEPT option src lan ```

Test Case Field	Description
	`` option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 172.217.164.132 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_ent0-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 option dest_ip 0.0.0.0 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_ent0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 172.217.164.132 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_ent0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 0.0.0.0 option dest_ip 192.168.20.222 option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- ```

Test Case Field	Description
	Build2_loc0-frdev' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.222 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_loc0-todev' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip any option dest_ip 192.168.20.222 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_man0-frdev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.222 option ipset www_gmail_com-SMTD option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi- Build2_man0-todev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option ipset www_gmail_com-SMFD option dest_ip 192.168.20.222 option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-

Test Case Field	Description
| | ```
Build2_myctl0-frdev'
        option target    ACCEPT
        option src       lan
        option dest      wan
        option proto     all
        option family    ipv4
        option src_ip     192.168.20.222
        option dest_ip    192.168.20.101

config rule
        option enabled   '1'
        option name      'mud_192.168.20.222_main-pi-
Build2_myctl0-todev'
        option target    ACCEPT
        option src       wan
        option dest      lan
        option proto     all
        option family    ipv4
        option src_ip     192.168.20.101
        option dest_ip    192.168.20.222

config rule
        option enabled   '1'
        option name      'mud_192.168.20.222_main-pi-
Build2_myman0-frdev-SM'
        option target    ACCEPT
        option src       lan
        option dest      lan
        option proto     udp
        option family    ipv4
        option src_ip     192.168.20.222
        option ipset      mudfiles_nist_getyikes_com-SMTD

config rule
        option enabled   '1'
        option name      'mud_192.168.20.222_main-pi-
Build2_myman0-todev-SM'
        option target    ACCEPT
        option src       lan
        option dest      lan
        option proto     udp
        option family    ipv4
        option ipset      mudfiles_nist_getyikes_com-SMFD
        option dest_ip    192.168.20.222

config rule
        option enabled   '1'
        option name      'mud_192.168.20.222_main-pi-
Build2_REJECT-ALL-LOCAL-FROM'
``` |
```

Test Case Field	Description
	`option target REJECT` `option src lan` `option dest lan` `option proto all` `option family ipv4` `option src_ip 192.168.20.222` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_REJECT-ALL-LOCAL-TO'` `option target REJECT` `option src lan` `option dest lan` `option proto all` `option family ipv4` `option src_ip any` `option dest_ip 192.168.20.222` `config rule` `option enabled '1'` `option name 'mud_192.168.20.222_main-pi-Build2_REJECT-ALL'` `option target REJECT` `option src lan` `option dest wan` `option proto all` `option family ipv4` `option src_ip 192.168.20.222` `# OSMUD end`
Overall Results	Pass

337 Test case IoT-1-v6 is identical to test case IoT-1-v4 except that IoT-1-v6 tests requirement CR-1.a.2,
338 whereas IoT-1-v4 tests requirement CR-1.a.1. Hence, as explained above, test IoT-1-v6 uses IPv6,
339 DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

340 3.1.2.2 Test Case IoT-2-v4

341 **Table 3-3: Test Case IoT-2-v4**

Test Case Field	Description
Parent Requirement	(CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server.
Testable Requirement	(CR-3.b) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it cannot validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.b.1) The MUD manager shall drop the connection to the MUD file server. (CR-3.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD manager cannot validate the TLS certificate of a MUD file server when trying to retrieve the MUD file for a specific IoT device, the MUD manager will drop the connection to the MUD file server and configure the router/switch according to locally defined policy regarding whether to allow or block traffic to the IoT device in question
Associated Test Case(s)	IoT-11-v4 (for the v6 version of this test, IoT-11-v6)
Associated Cybersecurity Framework Subcategory(ies)	PR.AC-7
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Yikesmain.json, yikesmantest.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The MUD file server that is hosting the MUD file of the device under test does not have a valid TLS certificate. 4. Local policy has been defined to ensure that if the MUD file for a device is located on a server with an invalid certificate, the

Test Case Field	Description
	router/switch will be configured to deny all communication to and from the device. 5. The MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 3. The DHCP server offers an IP address lease to the newly connected IoT device. 4. The IoT device requests this IP address lease, which the DHCP server acknowledges. 5. The DHCP server sends the MUD URL to the MUD manager. 6. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, determines that it does not have a valid TLS certificate, and drops the connection to the MUD file server. 7. The MUD manager configures the router/switch that is closest to the IoT device according to locally defined policy, which in this case allows traffic to the IoT device in question.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to local policy for communication to/from the IoT device.

Test Case Field	Description
Actual Results	**Procedures 1–4:** `pi@main-pi-Build2:~$ sudo dhclient -v -i eth0` `sudo: unable to resolve host main-pi-Build2: Connection re-fused` `Internet Systems Consortium DHCP Client 4.3.5` `Copyright 2004-2016 Internet Systems Consortium.` `All rights reserved.` `For info, please visit https://www.isc.org/software/dhcp/` `RTNETLINK answers: Operation not possible due to RF-kill` `Listening on LPF/wlan0/b8:27:eb:be:39:de` `Sending on   LPF/wlan0/b8:27:eb:be:39:de` `Listening on LPF/eth0/b8:27:eb:eb:6c:8b` `Sending on   LPF/eth0/b8:27:eb:eb:6c:8b` `Sending on   Socket/fallback` **`DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 4`** **`DHCPREQUEST of 192.168.20.224 on eth0 to 255.255.255.255 port 67`** **`DHCPOFFER of 192.168.20.224 from 192.168.20.1`** **`DHCPACK of 192.168.20.224 from 192.168.20.1`** `Too few arguments.` `Too few arguments.` **`bound to 192.168.20.224`** `-- renewal in 1800 seconds`**.** **Procedure 5:** **dhcpmasq.txt** `2019-07-15T20:27:57Z|OLD|Wired|DHCP|-|MUD|-|-|ba:47:a1:7d:60:44|192.168.20.148||` `2019-07-15T20:28:01Z|OLD|NIST 5|DHCP|-|MUD|-|-|18:b4:30:50:98:38|192.168.20.203||` `2019-07-15T20:28:08Z|OLD|NIST 2.4|DHCP|-|MUD|-|-|d0:73:d5:28:08:2a|192.168.20.202||` `2019-07-15T20:28:11Z|OLD|Wired|DHCP|-|MUD|-|-|b8:27:eb:95:55:fe|192.168.20.232|raspberrypi|` **`2019-07-15T20:28:31Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|`**https://mudfiles.nist.getyikes.com/yikesmain.json|-|b8:27:eb:eb:6c:8b|192.168.20.224|main-pi-Build2| `2019-07-15T20:28:42Z|NEW|NIST 5|DHCP|1,28,2,121,15,6,12,40,41,42,26,119,3,121,249,33,252,42|MUD|-|-|80:00:0b:ef:81:70|192.168.20.238||`

Test Case Field	Description
	Procedure 6: **MUD Manager:** `2019-06-18 13:59:50 INFO::GENERAL::NEW Device Action: IP: 192.168.20.224, MAC: b8:27:eb:eb:6c:8b` `2019-06-18 13:59:50 ERROR::COMMUNICATION::curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE -- http-code: 0` `2019-06-18 13:59:50 WARNING::COMMUNICATION::`**`Comm error with a mud-file-server. Retrying transaction`**`...` `2019-06-18 13:59:50 INFO::GENERAL::NEW Device Action: IP: 192.168.20.224, MAC: b8:27:eb:eb:6c:8b` `2019-06-18 13:59:51 ERROR::COMMUNICATION::curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE -- http-code: 0` `2019-06-18 13:59:51 ERROR::GENERAL::Comm error with mud-file-server. Aborting transaction after second attempt and quarantine device.` **Procedure 7:** **Router/PEP:** `# OSMUD start` `#` `# DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-FIGURATION` `#` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM`

Test Case Field	Description
	config ipset option enabled 1 option name mudfileserver-SMTD option match dest_ip option storage hash option family ipv4 option external mudfileserver-SM config ipset option enabled 1 option name mudfileserver-SMFD option match src_ip option storage hash option family ipv4 option external mudfileserver-SM config ipset option enabled 1 option name www_facebook_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_facebook_com-SMFD option match src_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_gmail_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_gmail_com-SM config ipset option enabled 1 option name www_gmail_com-SMFD option match src_ip option storage hash option family ipv4 option external www_gmail_com-SM config rule

Test Case Field	Description
	`` option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.197 option dest_ip 198.71.233.87 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 198.71.233.87 option dest_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_myman0-frdev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.197 option ipset www_facebook_com-SMTD option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_myman0-todev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option ipset www_facebook_com-SMFD option dest_ip 192.168.20.197 option dest_port 80:80 ```

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>
config rule
        option enabled   '1'
        option name   'mud_192.168.20.197_same-manufac-
ture-pi_REJECT-ALL-LOCAL-FROM'
        option target   REJECT
        option src   lan
        option dest   lan
        option proto   all
        option family   ipv4
        option src_ip   192.168.20.197

config rule
        option enabled   '1'
        option name   'mud_192.168.20.197_same-manufac-
ture-pi_REJECT-ALL-LOCAL-TO'
        option target   REJECT
        option src   lan
        option dest   lan
        option proto   all
        option family   ipv4
        option src_ip   any
        option dest_ip   192.168.20.197

config rule
        option enabled   '1'
        option name   'mud_192.168.20.197_same-manufac-
ture-pi_REJECT-ALL'
        option target   REJECT
        option src   lan
        option dest   wan
        option proto   all
        option family   ipv4
        option src_ip   192.168.20.197
OSMUD end
</pre></td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

As explained above, test IoT-2-v6 is identical to test IoT-2-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

3.1.2.3 Test Case IoT-3-v4

Table 3-4: Test Case IoT-3-v4

Test Case Field	Description
Parent Requirement	(CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager.
Testable Requirement	(CR-4.b) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file was valid at the time of signing, i.e., the certificate had already expired when it was used to sign the MUD file. (CR-4.b.1) The MUD manager shall cease to process the MUD file. (CR-4.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD file server serves a MUD file with a signature that was created with an expired certificate, the MUD manager will cease processing the MUD file
Associated Test Case(s)	IoT-11-v4 (for the v6 version of this test, IoT-11-v6)
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*ExpiredCertTest.json*
Preconditions	1. This MUD file is not currently cached at the MUD manager. 2. The IoT device's MUD file is being hosted on a MUD file server that has a valid TLS certificate, but the MUD file signature was signed by a certificate that had already expired at the time of signature. 3. Local policy has been defined to ensure that if the MUD file for a device has a signature that was signed by a certificate that had already expired at the time of signature, the device's MUD PEP router/switch will be configured to deny all communication to/from the device.

Test Case Field	Description
	4. The MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 3. The DHCP server offers an IP address lease to the newly connected IoT device. 4. The IoT device requests this IP address lease, which the DHCP server acknowledges. 5. The DHCP server sends the MUD URL to the MUD manager. 6. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. 7. The MUD file server serves the MUD file and signature to the MUD manager, and the MUD manager detects that the MUD file's signature was created by using a certificate that had already expired at the time of signing. 8. The MUD manager configures the router/switch that is closest to the IoT device so that it allows all communications to and from the IoT device.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to deny all communication to and

Test Case Field	Description
	from the IoT device. The expected configuration should resemble the following. Expecting a show access session without a MUD file as seen below: `# OSMUD start` `#` `# DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-FIGURATION` `#` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfileserver-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfileserver-SM` `config ipset` `    option enabled 1` `    option name mudfileserver-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfileserver-SM` `config ipset` `    option enabled 1` `    option name www_facebook_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4`

Test Case Field	Description
	option external www_facebook_com-SM config ipset option enabled 1 option name www_facebook_com-SMFD option match src_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_gmail_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_gmail_com-SM config ipset option enabled 1 option name www_gmail_com-SMFD option match src_ip option storage hash option family ipv4 option external www_gmail_com-SM # OSMUD end
Actual Results	**Procedures 1–4:** pi@main-pi-Build2:~$ sudo dhclient -v -i eth0 sudo: unable to resolve host main-pi-Build2: Connection re-fused Internet Systems Consortium DHCP Client 4.3.5 Copyright 2004-2016 Internet Systems Consortium. All rights reserved. For info, please visit https://www.isc.org/software/dhcp/ RTNETLINK answers: Operation not possible due to RF-kill Listening on LPF/wlan0/b8:27:eb:be:39:de Sending on LPF/wlan0/b8:27:eb:be:39:de Listening on LPF/eth0/b8:27:eb:eb:6c:8b Sending on LPF/eth0/b8:27:eb:eb:6c:8b Sending on Socket/fallback **DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 4**

Test Case Field	Description
	DHCPREQUEST of 192.168.20.226 on eth0 to 255.255.255.255 port 67 **DHCPOFFER of 192.168.20.226 from 192.168.20.1** **DHCPACK of 192.168.20.226 from 192.168.20.1** Too few arguments. Too few arguments. **bound to 192.168.20.226** -- renewal in 1800 seconds**.** **Procedure 5:** **dhcpmasq.txt** 2019-07-11T18:03:00Z|OLD|Wired|DHCP|-|MUD|-|-|ba:47:a1:7d:41:bb|192.168.20.160|| 2019-07-11T18:03:05Z|OLD|NIST 5|DHCP|-|MUD|-|-|18:b4:30:50:E2:01|192.168.20.143|| 2019-07-11T18:03:12Z|DEL|Wired|DHCP|-|MUD|-||b8:27:eb:95:55:fe|192.168.20.233|raspberrypi| 2019-07-**11T18:03:25Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|https://mudfiles.nist.getyikes.com/ExpiredCert-Test.json|-|b8:27:eb:eb:6c:8b|192.168.20.226|main-pi-Build2|** **Procedure 7:** **MUD Manager:** 2019-07-11 18:03:26 DEBUG::GENERAL::**2019-07-11T18:03:25Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|https://mudfiles.nist.getyikes.com/ExpiredCert-Test.json|-|b8:27:eb:eb:6c:8b|192.168.20.226|main-pi-Build2|** 2019-07-11 18:03:26 DEBUG::GENERAL::Executing on dhcpmasq info 2019-07-11 18:03:26 INFO::GENERAL::NEW Device Action: IP: 192.168.20.226, MAC: b8:27:eb:eb:6c:8b 2019-07-11 18:03:26 DEBUG::COMMUNICATION::curl_easy_per-form() doing it now.... 2019-07-11 18:03:26 DEBUG::COMMUNICATION::https://mud-files.nist.getyikes.com/ExpiredCertTest.json 2019-07-11 18:03:26 DEBUG::COMMUNICATION::Found HTTPS 2019-07-11 18:03:26 DEBUG::COMMUNICATION::in write data 2019-07-11 18:03:26 DEBUG::COMMUNICATION::curl_easy_per-form() success 2019-07-11 18:03:26 DEBUG::COMMUNICATION::MUD File Server returned success state. 2019-07-11 18:03:26 DEBUG::COMMUNICATION::curl_easy_per-form() doing it now.... 2019-07-11 18:03:26 DEBUG::COMMUNICATION::https://mud-files.nist.getyikes.com/ExpiredCertTest.p7s 2019-07-11 18:03:26 DEBUG::COMMUNICATION::Found HTTPS 2019-07-11 18:03:27 DEBUG::COMMUNICATION::in write data

Test Case Field	Description
	2019-07-11 18:03:27 DEBUG::COMMUNICATION::curl_easy_per-form() success 2019-07-11 18:03:27 DEBUG::COMMUNICATION::MUD File Server returned success state. 2019-07-11 18:03:27 DEBUG::MUD_FILE_OPERATIONS::IN ****NEW**** MUD and SIG FILE RETRIEVED!!! 2019-07-11 18:03:27 DEBUG::GENERAL::IN ****NEW**** vali-dateMudFileWithSig() 2019-07-11 18:03:27 DEBUG::GENERAL::openssl cms -verify -in /etc/osmud/state/mudfiles/ExpiredCertTest.p7s -inform DER -content /etc/osmud/state/mudfiles/ExpiredCertTest.json -pur-pose any > /dev/null 2019-07-11 18:03:27 ERROR::DEVICE_INTERFACE::openssl cms -verify -in /etc/osmud/state/mudfiles/ExpiredCertTest.p7s -inform DER -content /etc/osmud/state/mudfiles/ExpiredCert-Test.json -purpose any > /dev/null **2019-07-11 18:03:27 ERROR::MUD_FILE_OPERATIONS::Could not validate the MUD File signature using openssl cms verify. Abort mud file processing and quarantine device.** 2019-07-11 18:03:27 DEBUG::GENERAL::/etc/osmud/cre-ate_ip_fw_rule.sh -s lan -d wan -i 192.168.20.226 -a any -j any -b any -p all -n REJECT-ALL -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.226 2019-07-11 18:03:27 DEBUG::GENERAL::/etc/osmud/cre-ate_ip_fw_rule.sh -s lan -d lan -i 192.168.20.226 -a any -j any -b any -p all -n REJECT-ALL-LOCAL-FROM -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.226 2019-07-11 18:03:27 DEBUG::GENERAL::/etc/osmud/cre-ate_ip_fw_rule.sh -s lan -d lan -i any -a any -j 192.168.20.226 -b any -p all -n REJECT-ALL-LOCAL-TO -t AC-CEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.226 **Router/PEP:** # OSMUD start # # DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-FIGURATION # config ipset option enabled 1 option name mudfiles_nist_getyikes_com-SMTD option match dest_ip option storage hash option family ipv4 option external mudfiles_nist_getyikes_com-SM config ipset

Test Case Field	Description
	`` option enabled 1 option name mudfiles_nist_getyikes_com-SMFD option match src_ip option storage hash option family ipv4 option external mudfiles_nist_getyikes_com-SM config ipset option enabled 1 option name mudfileserver-SMTD option match dest_ip option storage hash option family ipv4 option external mudfileserver-SM config ipset option enabled 1 option name mudfileserver-SMFD option match src_ip option storage hash option family ipv4 option external mudfileserver-SM config ipset option enabled 1 option name www_facebook_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_facebook_com-SMFD option match src_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_gmail_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_gmail_com-SM config ipset option enabled 1 ```

Test Case Field	Description
	`` option name www_gmail_com-SMFD option match src_ip option storage hash option family ipv4 option external www_gmail_com-SM config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.197 option dest_ip 198.71.233.87 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 198.71.233.87 option dest_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_myman0-frdev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.197 option ipset www_facebook_com-SMTD option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_myman0-todev-SM' option target ACCEPT option src lan ```

Test Case Field	Description
	option dest lan option proto tcp option family ipv4 option ipset www_facebook_com-SMFD option dest_ip 192.168.20.197 option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac-ture-pi_REJECT-ALL-LOCAL-FROM' option target REJECT option src lan option dest lan option proto all option family ipv4 option src_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac-ture-pi_REJECT-ALL-LOCAL-TO' option target REJECT option src lan option dest lan option proto all option family ipv4 option src_ip any option dest_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac-ture-pi_REJECT-ALL' option target REJECT option src lan option dest wan option proto all option family ipv4 option src_ip 192.168.20.197 # OSMUD end
Overall Results	Pass

As explained above, test IoT-3-v6 is identical to test IoT-3-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

3.1.2.4 Test Case IoT-4-v4

Table 3-5: Test Case IoT-4-v4

Test Case Field	Description
Parent Requirement	(CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file.
Testable Requirement	(CR-5.b) The MUD manager shall attempt to validate the signature of the MUD file, but the signature validation fails (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason). (CR-5.b.1) The MUD manager shall cease processing the MUD file. (CR-5.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if the MUD manager determines that the signature on the MUD file it receives from the MUD file server is invalid, it will cease processing the MUD file and configure the router/switch according to locally defined policy regarding whether to allow or block traffic to the IoT device in question
Associated Test Case(s)	IoT-11-v4 (for the v6 version of this test, IoT-11-v6)
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*cr-5b.json*
Preconditions	1. This MUD file is not currently cached at the MUD manager. 2. The MUD file that is served from the MUD file server to the MUD manager has a signature that is invalid, even though it was signed by a certificate that had not expired at the time of signing.

Test Case Field	Description
	3. Local policy has been defined to ensure that if the MUD file for a device has an invalid signature, the device's MUD PEP router/switch will be configured to deny all communication to/from the device. 4. The MUD PEP router/switch does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Power on the IoT device and connect it to the test network. This should set in motion the following series of steps, which should occur automatically: 1. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 2. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 3. The DHCP server offers an IP address lease to the newly connected IoT device. 4. The IoT device requests this IP address lease, which the DHCP server acknowledges. 5. The DHCP server sends the MUD URL to the MUD manager. 6. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. 7. The MUD file server sends the MUD file, and the MUD manager detects that the MUD file's signature is invalid. 8. The MUD manager configures the router/switch that is closest to the IoT device so that it allows all communications to and from the IoT device.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to deny all communication to/from

Test Case Field	Description
	the IoT device. The expected configuration should resemble the following: Expecting a show access session without a MUD file as seen below: `# OSMUD start` `#` `# DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CONFIGURATION` `#` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfileserver-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfileserver-SM` `config ipset` `    option enabled 1` `    option name mudfileserver-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfileserver-SM` `config ipset` `    option enabled 1` `    option name www_facebook_com-SMTD` `    option match dest_ip` `    option storage hash`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>
    option family ipv4
    option external www_facebook_com-SM

config ipset
    option enabled 1
    option name www_facebook_com-SMFD
    option match src_ip
    option storage hash
    option family ipv4
    option external www_facebook_com-SM

config ipset
    option enabled 1
    option name www_gmail_com-SMTD
    option match dest_ip
    option storage hash
    option family ipv4
    option external www_gmail_com-SM

config ipset
    option enabled 1
    option name www_gmail_com-SMFD
    option match src_ip
    option storage hash
    option family ipv4
    option external www_gmail_com-SM

OSMUD end
</pre></td></tr>
<tr><td>Actual Results</td><td>Procedures 1-5:
Excluded for sake of length.

Procedure 6:
MUD MANAGER:

<pre>
2019-07-11 18:10:30 DEBUG::GENERAL::2019-07-11T18:10:24Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|https://mudfiles.nist.getyikes.com/cr-5b.json|-|b8:27:eb:eb:6c:8b|192.168.20.226|main-pi-Build2|
2019-07-11 18:10:30 DEBUG::GENERAL::Executing on dhcpmasq info
2019-07-11 18:10:30 INFO::GENERAL::NEW Device Action: IP: 192.168.20.226, MAC: b8:27:eb:eb:6c:8b
2019-07-11 18:10:30 DEBUG::COMMUNICATION::curl_easy_perform() doing it now....
</pre></td></tr>
</table>

Test Case Field	Description
	2019-07-11 18:10:30 DEBUG::COMMUNICATION::https://mud-files.nist.getyikes.com/cr-5b.json 2019-07-11 18:10:30 DEBUG::COMMUNICATION::Found HTTPS 2019-07-11 18:10:31 DEBUG::COMMUNICATION::in write data 2019-07-11 18:10:31 DEBUG::COMMUNICATION::curl_easy_perform() success 2019-07-11 18:10:31 DEBUG::COMMUNICATION::MUD File Server returned success state. 2019-07-11 18:10:31 DEBUG::COMMUNICATION::curl_easy_perform() doing it now.... 2019-07-11 18:10:31 DEBUG::COMMUNICATION::https://mud-files.nist.getyikes.com/cr-5b.p7s 2019-07-11 18:10:31 DEBUG::COMMUNICATION::Found HTTPS 2019-07-11 18:10:31 DEBUG::COMMUNICATION::in write data 2019-07-11 18:10:31 DEBUG::COMMUNICATION::curl_easy_perform() success 2019-07-11 18:10:31 DEBUG::COMMUNICATION::MUD File Server returned success state. 2019-07-11 18:10:31 DEBUG::MUD_FILE_OPERATIONS::IN ****NEW**** MUD and SIG FILE RETRIEVED!!! 2019-07-11 18:10:31 DEBUG::GENERAL::IN ****NEW**** validateMudFileWithSig() 2019-07-11 18:10:31 DEBUG::GENERAL::openssl cms -verify -in /etc/osmud/state/mudfiles/cr-5b.p7s -inform DER -content /etc/osmud/state/mudfiles/cr-5b.json -purpose any > /dev/null 2019-07-11 18:10:31 ERROR::DEVICE_INTERFACE::openssl cms -verify -in /etc/osmud/state/mudfiles/cr-5b.p7s -inform DER -content /etc/osmud/state/mudfiles/cr-5b.json -purpose any > /dev/null **2019-07-11 18:10:31 ERROR::MUD_FILE_OPERATIONS::Could not validate the MUD File signature using openssl cms verify. Abort mud file processing and quarantine device.** 2019-07-11 18:10:31 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.226 -a any -j any -b any -p all -n REJECT-ALL -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.226 2019-07-11 18:10:31 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i 192.168.20.226 -a any -j any -b any -p all -n REJECT-ALL-LOCAL-FROM -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.226 2019-07-11 18:10:31 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d lan -i any -a any -j

Test Case Field	Description
	`192.168.20.226 -b any -p all -n REJECT-ALL-LOCAL-TO -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.226`
	Procedure 7: **Router/PEP:**

```
# OSMUD start
#
# DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-
FIGURATION
#

config ipset
    option enabled 1
    option name mudfiles_nist_getyikes_com-SMTD
    option match dest_ip
    option storage hash
    option family ipv4
    option external mudfiles_nist_getyikes_com-SM

config ipset
    option enabled 1
    option name mudfiles_nist_getyikes_com-SMFD
    option match src_ip
    option storage hash
    option family ipv4
    option external mudfiles_nist_getyikes_com-SM

config ipset
    option enabled 1
    option name mudfileserver-SMTD
    option match dest_ip
    option storage hash
    option family ipv4
    option external mudfileserver-SM

config ipset
    option enabled 1
    option name mudfileserver-SMFD
    option match src_ip
    option storage hash
    option family ipv4
    option external mudfileserver-SM

config ipset
    option enabled 1
    option name www_facebook_com-SMTD
```

Test Case Field	Description
	`` option match dest_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_facebook_com-SMFD option match src_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_gmail_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_gmail_com-SM config ipset option enabled 1 option name www_gmail_com-SMFD option match src_ip option storage hash option family ipv4 option external www_gmail_com-SM config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.197 option dest_ip 198.71.233.87 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-todev' option target ACCEPT option src wan option dest lan option proto tcp ```

Test Case Field	Description
	` option family ipv4` ` option src_ip 198.71.233.87` ` option dest_ip 192.168.20.197` `config rule` ` option enabled '1'` ` option name 'mud_192.168.20.197_same-manufac-` `ture-pi_myman0-frdev-SM'` ` option target ACCEPT` ` option src lan` ` option dest lan` ` option proto tcp` ` option family ipv4` ` option src_ip 192.168.20.197` ` option ipset www_facebook_com-SMTD` ` option dest_port 80:80` `config rule` ` option enabled '1'` ` option name 'mud_192.168.20.197_same-manufac-` `ture-pi_myman0-todev-SM'` ` option target ACCEPT` ` option src lan` ` option dest lan` ` option proto tcp` ` option family ipv4` ` option ipset www_facebook_com-SMFD` ` option dest_ip 192.168.20.197` ` option dest_port 80:80` `config rule` ` option enabled '1'` ` option name 'mud_192.168.20.197_same-manufac-` `ture-pi_REJECT-ALL-LOCAL-FROM'` ` option target REJECT` ` option src lan` ` option dest lan` ` option proto all` ` option family ipv4` ` option src_ip 192.168.20.197` `config rule` ` option enabled '1'` ` option name 'mud_192.168.20.197_same-manufac-` `ture-pi_REJECT-ALL-LOCAL-TO'` ` option target REJECT` ` option src lan` ` option dest lan` ` option proto all`

Test Case Field	Description
	`option family ipv4` `option src_ip any` `option dest_ip 192.168.20.197` `config rule` `option enabled '1'` `option name 'mud_192.168.20.197_same-manufac-ture-pi_REJECT-ALL'` `option target REJECT` `option src lan` `option dest wan` `option proto all` `option family ipv4` `option src_ip 192.168.20.197` `# OSMUD end`
Overall Results	Pass

350 As explained above, test IoT-4-v6 is identical to test IoT-4-v4 except that it uses IPv6, DHCPv6, and IANA
351 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

352 ### *3.1.2.5 Test Case IoT-5-v4*

353 **Table 3-6: Test Case IoT-5-v4**

Test Case Field	Description
Parent Requirement	(CR-7) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file. (CR-8) The IoT DDoS example implementation shall deny communications from a MUD-enabled IoT device to unapproved internet services (i.e., services that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-7.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to approved internet services. (CR-7.a.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file.

Test Case Field	Description
	(CR-7.b) An approved internet service shall attempt to initiate connection to the MUD-enabled IoT device. (CR-7.b.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-8.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to unapproved (implicitly denied) internet services. (CR-8.a.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.b) An unapproved (implicitly denied) internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-8.b.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.c) The MUD-enabled IoT device shall initiate communications to an internet service that is approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device. (CR-8.c.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.d) An internet service shall initiate communications to a MUD-enabled device that is approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service. (CR-8.d.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with internet services. Further shows that the policies that are configured on the MUD PEP router/switch with respect to communication with internet services will be enforced as expected, with communications that are configured as denied being blocked and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4 (for the v6 version of this test, IoT-1-v6)

Test Case Field	Description
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.IP-1, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Yikesmain.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the following policies for the IoT device in question (as defined in the MUD file in Section 3.1.3): Note: Procedure steps with strike-through were not tested due to network address translation (NAT). ~~a) Explicitly permit *https://yes-permit-from.com* to initiate communications with the IoT device.~~ b) Explicitly permit the IoT device to initiate communications with *https://yes-permit-to.com*. c) Implicitly deny all other communications with the internet, including denying i) the IoT device to initiate communications with *https://yes-permit-from.com* ~~ii) *https://yes-permit-to.com* to initiate communications with the IoT device~~ iii) communication between the IoT device and all other internet locations, such as *https://unnamed-to.com* (by not mentioning this or any other URLs in the MUD file)
Procedure	Note: Procedure steps with strike-through were not tested due to NAT. 1. As stipulated in the preconditions, right before this test, test IoT-1-v4 (or IoT-1-v6) must have been run successfully. 2. Initiate communications from the IoT device to *https://yes-permit-to.com* and verify that this traffic is received at *https://yes-permit-to.com*. (egress) ~~3. Initiate communications to the IoT device from *https://yes-permit-to.com* and verify that this traffic is received at the MUD PEP, but it~~

Test Case Field	Description
	~~is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress)~~ ~~4. Initiate communications to the IoT device from *https://yes-permit-from.com* and verify that this traffic is received at the IoT device. (ingress)~~ ~~5. Initiate communications from the IoT device to *https://yes-permit-from.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at *https://yes-permit-from.com*. (ingress)~~ 6. Initiate communications from the IoT device to *https://unnamed.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at *https://unnamed.com*. (egress) ~~7. Initiate communications to the IoT device from *https://unnamed.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress)~~
Expected Results	Each of the results that is listed as needing to be verified in procedure steps above occurs as expected.
Actual Results	**Procedure 1:** Excluded for length's sake **Procedure 2:** https://www.google.com (approved): `--2019-07-11 18:23:38--  https://www.google.com/` `Resolving www.google.com (www.google.com)... 172.217.164.132, 2607:f8b0:4004:814::2004` **`Connecting to www.google.com (www.google.com)|172.217.164.132|:443... connected.`** `HTTP request sent, awaiting response... 200 OK` `Length: unspecified [text/html]`

Test Case Field	Description
	Saving to: ‘index.html.6’ 0K .... 15.7M=0.001s 2019-07-11 18:23:38 (15.7 MB/s) - ‘index.html.6’ saved [11449] https://www.osmud.org (approved): --2019-07-11 18:23:04-- https://www.osmud.org/ Resolving www.osmud.org (www.osmud.org)... 198.71.233.87 **Connecting to www.osmud.org (www.osmud.org)|198.71.233.87|:443... connected.** HTTP request sent, awaiting response... 301 Moved Permanently Location: https://osmud.org/ [following] --2019-07-11 18:23:04-- https://osmud.org/ Resolving osmud.org (osmud.org)... 198.71.233.87 **Connecting to osmud.org (osmud.org)|198.71.233.87|:443... connected.** **HTTP request sent, awaiting response... 200 OK** Length: unspecified [text/html] Saving to: ‘index.html.4’ 0K .... 3.40M=0.007s 2019-07-11 18:23:05 (3.40 MB/s) - ‘index.html.4’ saved [24697] https://www.trytechy.com (approved): --2019-07-11 18:23:24-- https://www.trytechy.com/

Test Case Field	Description
	`Resolving www.trytechy.com (www.trytechy.com)... 99.84.181.77, 99.84.181.123, 99.84.181.11, ...` **`Connecting to www.trytechy.com (www.trytechy.com)|99.84.181.77|:443... connected.`** **`HTTP request sent, awaiting response... 200 OK`** `Length: unspecified [text/html]` `Saving to: 'index.html.5'` `    0K .......... ...... 13.1M=0.001s` `2019-07-11 18:23:24 (13.1 MB/s) - 'index.html.5' saved [16529]` --- **Procedure 6:** https://www.facebook.com (unapproved): `--2019-07-11 18:23:55--  https://www.facebook.com/` `Resolving www.facebook.com (www.facebook.com)... 31.13.71.36, 2a03:2880:f103:83:face:b00c:0:25de` **`Connecting to www.facebook.com (www.facebook.com)|31.13.71.36|:443... failed: Connection refused.`** `Connecting to www.facebook.com (www.facebook.com)|2a03:2880:f103:83:face:b00c:0:25de|:443... failed: Network is unreachable.` --- https://www.twitter.com (unapproved): `--2019-07-11 18:24:07--  https://www.twitter.com/` `Resolving www.twitter.com (www.twitter.com)... 104.244.42.1, 104.244.42.65`

Test Case Field	Description
	Connecting to www.twitter.com (www.twitter.com)|104.244.42.1|:443... failed: Connection refused. Connecting to www.twitter.com (www.twitter.com)|104.244.42.65|:443... failed: Connection refused.
Overall Results	Pass (for testable procedures, ingress cannot be tested due to NAT)

354 As explained above, test IoT-5-v6 is identical to test IoT-5-v4 except that it uses IPv6, DHCPv6, and IANA
355 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

356 3.1.2.6 Test Case IoT-6-v4

357 **Table 3-7: Test Case IoT-6-v4**

Test Case Field	Description
Parent Requirement	(CR-9) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate laterally with devices that are approved in the MUD file. (CR-10) The IoT DDoS example implementation shall deny lateral communications from a MUD-enabled IoT device to devices that are not approved in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-9.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to approved devices. (CR-9.a.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-9.b) An approved device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-9.b.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-10.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to unapproved (implicitly denied) devices.

Test Case Field	Description
	(CR-10.a.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-10.b) An unapproved (implicitly denied) device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-10.b.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with lateral devices. Further shows that the policies that are configured on the MUD PEP router/switch with respect to communication with lateral devices will be enforced as expected, with communications that are configured as denied being blocked and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4 (for the v6 version of this test, IoT-1-v6)
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.AC-5, PR.IP-1, PR.PT-3, PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi (3)
MUD File(s) Used	*Fe-localnetwork.json, Fe-my-controller.json, Fe-controller.json, Fe-manufacturer1.json, Fe-manufacturer2.json, Fe-samemanufacturer.json, Fe-localnetwork-to2.json, Fe-localnetwork-from2.json, Fe-samemanufacturer-from2.json, Fe-samemanufacturer-to2.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the following policies for the IoT device in question with respect to local communications (as defined in the MUD files in Section 3.1.3): a) Local-network class—Explicitly permit **local communication to and from the IoT device and any local hosts** (including the spe-

Test Case Field	Description
	cific local hosts *anyhost-to* and *anyhost-from*) **for specific services,** as specified in the MUD file by source port: any; destination port: 80; and protocol: TCP, and which party initiates the connection. b) Manufacturer class—Explicitly permit **local communication to and from the IoT device and other classes of IoT devices, as identified by their MUD URL (*www.devicetype.com*), and further constrained** by source port: any; destination port: 80; and protocol: TCP. c) Same-manufacturer class—Explicitly permit **local communication to and from IoT devices of the same manufacturer as the IoT device in question (the domain in the MUD URLs (mudfileserver) of the other IoT devices is the same as the domain in the MUD URL (mudfileserver) of the IoT device in question),** and further constrained by source port: any; destination port: 80; and protocol: TCP. d) Implicitly deny all other local communication that is not explicitly permitted in the MUD file, including denying i) ***anyhost-to* to initiate communications** with the IoT device ii) the **IoT device to initiate communications** with *anyhost-to* by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** iii) the **IoT device to initiate communications with *anyhost-from*** iv) ***anyhost-from* to initiate communications** with the IoT device by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** v) communications between the IoT device and all lateral hosts (including *unnamed-host*) whose **MUD URLs are not explicitly mentioned** as being permissible in the MUD file vi) communications between the IoT device and all lateral hosts whose **MUD URLs are explicitly mentioned** as being permissible **but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**

Test Case Field	Description
	vii) communications between the IoT device and all lateral hosts that are **not from the same manufacturer** as the IoT device in question viii) communications between the IoT device and a lateral host that **is from the same manufacturer but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**
Procedure	1. As stipulated in the preconditions, right before this test, test IoT-1-v4 (or IoT-1-v6) must have been run successfully. 2. Local-network (ingress): Initiate communications to the IoT device from *anyhost-from* **for specific permitted service,** and verify that this traffic is received at the IoT device. 3. Local-network (egress): **Initiate communications from the IoT device to *anyhost-from*** for specific permitted service, and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *anyhost-from*. 4. Local-network, controller, my-controller, manufacturer class (egress): Initiate communications from the IoT device to *anyhost-to* **for specific permitted service,** and verify that this traffic **is received** at *anyhost-to.* 5. Local-network, controller, my-controller, manufacturer class (ingress): **Initiate communications to the IoT device from *anyhost-to*** for specific permitted service, and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at the IoT device. 6. No associated class (egress): Initiate communications **from** the IoT device to *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted),** and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *unnamed-host*. 7. No associated class (ingress): Initiate communications **to** the IoT device from *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>whose MUD URL is not explicitly mentioned in the MUD file as being permitted), and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device.
8. Same-manufacturer class (egress): Initiate communications from the IoT device to same-manufacturer-host (where same-manufacturer-host is a host that is from the same manufacturer as the IoT device in question) and verify that this traffic is received at same-manufacturer-host.
9. Same-manufacturer class (egress): Initiate communications from the IoT device to same-manufacturer-host (where same-manufacturer-host is a host that is from the same manufacturer as the IoT device in question) but using a port or protocol that is not specified, and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at same-manufacturer-host.</td></tr>
<tr><td>Expected Results</td><td>Each of the results that is listed as needing to be verified in the procedure steps above occurs as expected.</td></tr>
<tr><td>Actual Results</td><td>Local-Network:

Procedure 2 (from laptop to pi):

http://192.168.20.222

<pre>[mud@localhost ~]$ wget 192.168.20.222
--2019-07-24 15:30:01-- http://192.168.20.222/
Connecting to 192.168.20.222:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10701 (10K) [text/html]
Saving to: ‘index.html’

100%[=====================================>]
10,701   --.-K/s   in 0s

2019-07-24 15:30:01 (139 MB/s) - ‘index.html’ saved
[10701/10701]</pre></td></tr>
</table>

Test Case Field	Description
	Procedure 3 (from pi to laptop): *http://192.168.20.238/* (unapproved): `--2019-07-10 17:37:09--  http://192.168.20.238/` **`Connecting to 192.168.20.238:80... failed: Connection refused.`** Procedure 4 (from pi to local hosts): *http://192.168.20.110:443/* (approved): `--2019-07-10 19:02:34--  http://192.168.20.110:443/` **`Connecting to 192.168.20.110:443... connected.`** **`HTTP request sent, awaiting response... 200 OK`** `Length: 10701 (10K) [text/html]` `Saving to: 'index.html.28'` `0K .......... 100% 11.2M=0.001s` `2019-07-10 19:02:34 (11.2 MB/s) - 'index.html.28' saved [10701/10701]` *http://192.168.20.232/* (approved): `--2019-07-10 19:00:10--  http://192.168.20.232/` **`Connecting to 192.168.20.232:80... connected.`** **`HTTP request sent, awaiting response... 200 OK`** `Length: 277` `Saving to: 'index.html.14'` `0K 100% 10.9M=0s`

Test Case Field	Description
	2019-07-10 19:00:10 (10.9 MB/s) - ‘index.html.14’ saved [277/277] *http://192.168.20.117/* (approved): --2019-07-10 18:59:40-- *http://192.168.20.117/* **Connecting to 192.168.20.117:80... connected.** **HTTP request sent, awaiting response... 200 OK** Length: 10701 (10K) [text/html] Saving to: ‘index.html.12’ 0K .... 100% 6.05M=0.002s 2019-07-10 18:59:40 (6.05 MB/s) - ‘index.html.12’ saved [10701/10701] *http://192.168.20.197/* (approved): --2019-07-10 18:55:39-- *http://192.168.20.197/* **Connecting to 192.168.20.197:80... connected.** **HTTP request sent, awaiting response... 200 OK** Length: 10701 (10K) [text/html] Saving to: ‘index.html.8’ 0K .... 100% 2.03M=0.005s 2019-07-10 18:55:40 (2.03 MB/s) - ‘index.html.8’ saved [10701/10701] *http://192.168.20.183/* (approved): --2019-07-10 18:59:21-- *http://192.168.20.183/* **Connecting to 192.168.20.183:80... connected.** **HTTP request sent, awaiting response... 200 OK**

Test Case Field	Description
	`Length: 10701 (10K) [text/html]` `Saving to: 'index.html.10'` `    0K ..........` `100% 17.6M=0.001s` `2019-07-10 18:59:21 (17.6 MB/s) - 'index.html.10' saved [10701/10701]` Procedure 5 (from laptop to pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-10 19:03:17--  `*`http://192.168.20.222/`* **`Connecting to 192.168.20.222:80... failed: Connection refused.`** Procedure 6 (from device): http://www.facebook.com (unapproved): `--2019-07-10 19:17:39--  https://www.facebook.com/` `Resolving www.facebook.com (www.facebook.com)... 31.13.71.36, 2a03:2880:f112:83:face:b00c:0:25de` `Connecting to www.facebook.com (www.facebook.com)|31.13.71.36|:443... `**`failed: Connection refused.`** `Connecting to www.facebook.com (www.facebook.com)|2a03:2880:f112:83:face:b00c:0:25de|:443... failed: Network is unreachable.` Procedure 7 (from laptop to Pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-10 19:20:06--  `*`http://192.168.20.222/`* **`Connecting to 192.168.20.222:80... failed: Connection refused.`** **Controller:** Procedure 4 (from Pi to controller):

Test Case Field	Description
	https://www.trytechy.com/ (approved): `--2019-07-10 17:29:55--  https://www.trytechy.com/` `Resolving www.trytechy.com (www.trytechy.com)... 54.230.193.215, 54.230.193.99, 54.230.193.140, ...` **`Connecting to www.trytechy.com (www.trytechy.com)|54.230.193.215|:443... connected.`** **`HTTP request sent, awaiting response... 200 OK`** `Length: unspecified [text/html]` `Saving to: ‘index.html’` `    0K .......... ......` `1.80M=0.009s` `2019-07-10 17:29:55 (1.80 MB/s) - ‘index.html’ saved [16529]` Procedure 5 (from laptop to pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-10 17:30:04--  `*`http://192.168.20.222/`* **`Connecting to 192.168.20.222:80... failed: Connection refused.`** Procedure 6 (from pi to local hosts): *http://192.168.20.232/* (unapproved): `--2019-07-10 17:37:09--  `*`http://192.168.20.232/`* **`Connecting to 192.168.20.232:80... failed: Connection refused.`** *http://192.168.20.110/* (unapproved): `--2019-07-10 17:38:49--  `*`http://192.168.20.110/`* **`Connecting to 192.168.20.110:80... failed: Connection refused.`**

Test Case Field	Description
	http://192.168.20.183/ (unapproved): `--2019-07-10 17:46:38--  http://192.168.20.183/` **`Connecting to 192.168.20.183:80... failed: Connection refused.`**
	http://192.168.20.142/ (unapproved): `--2019-07-10 17:36:38--  http://192.168.20.142/` **`Connecting to 192.168.20.142:80... failed: Connection refused.`**
	http://192.168.20.117/ (unapproved): `--2019-07-10 17:36:55--  http://192.168.20.117/` **`Connecting to 192.168.20.117:80... failed: Connection refused.`**
	http://192.168.20.171/ (unapproved): `--2019-07-10 17:47:18--  http://192.168.20.171/` **`Connecting to 192.168.20.171:80... failed: Connection refused.`**
	http://192.168.20.181/ (unapproved): `--2019-07-10 17:47:49--  http://192.168.20.181/` **`Connecting to 192.168.20.181:80... failed: Connection refused.`**
	http://192.168.20.247/ (unapproved): `--2019-07-10 17:48:13--  http://192.168.20.247/` **`Connecting to 192.168.20.247:80... failed: Connection refused.`**
	Procedure 7 (from laptop to Pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-10 17:50:22--  http://192.168.20.222/`

Test Case Field	Description
	Connecting to 192.168.20.222:80... failed: Connection refused.
	My Controller: Procedure 4 (from device): https://www.google.com (approved): `--2019-07-10 18:13:12--  https://www.google.com/` `Resolving www.google.com (www.google.com)... 172.217.164.132, 2607:f8b0:4004:814::2004` **`Connecting to www.google.com (www.google.com)|172.217.164.132|:443... connected.`** `HTTP request sent, awaiting response... 200 OK` `Length: unspecified [text/html]` `Saving to: 'index.html.1'` `0K .......... .. 14.9M=0.001s` `2019-07-10 18:13:12 (14.9 MB/s) - 'index.html.1' saved [12327]`
	Procedure 5 (from laptop to pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-24 18:22:48--  ` *`http://192.168.20.222/`* **`Connecting to 192.168.20.222:80... failed: Connection refused.`**
	Procedure 6 (from device): *http://192.168.20.110/* (unapproved): `--2019-07-10 18:29:42--  ` *`http://192.168.20.110/`* **`Connecting to 192.168.20.110:80... failed: Connection refused.`**
	http://192.168.20.117/ (unapproved): `--2019-07-10 18:29:34--  ` *`http://192.168.20.117/`* **`Connecting to 192.168.20.117:80... failed: Connection refused.`**
	http://192.168.20.142/ (unapproved):

Test Case Field	Description
	`--2019-07-10 18:30:26-- http://192.168.20.142/` **`Connecting to 192.168.20.142:80... failed: Connection refused.`**
	http://192.168.20.171/ (unapproved): `--2019-07-10 18:29:55-- http://192.168.20.171/` **`Connecting to 192.168.20.171:80... failed: Connection refused.`**
	http://192.168.20.181/ (unapproved): `--2019-07-10 18:29:08-- http://192.168.20.181/` **`Connecting to 192.168.20.181:80... failed: Connection refused.`**
	http://192.168.20.183/ (unapproved): `--2019-07-10 18:29:23-- http://192.168.20.183/` **`Connecting to 192.168.20.183:80... failed: Connection refused.`**
	http://192.168.20.197/ (unapproved): `--2019-07-10 18:28:32-- http://192.168.20.197/` **`Connecting to 192.168.20.197:80... failed: Connection refused.`**
	http://192.168.20.232/ (unapproved): `--2019-07-10 18:30:36-- http://192.168.20.232/` **`Connecting to 192.168.20.232:80... failed: Connection refused.`**
	http://192.168.20.247/ (unapproved): `--2019-07-10 18:28:45-- http://192.168.20.247/` **`Connecting to 192.168.20.247:80... failed: Connection refused.`**
	Procedure 7 (from laptop to Pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-10 18:29:13--` http://192.168.20.222/ **`Connecting to 192.168.20.222:80... failed: Connection refused.`**
	Same Manufacturer 1 (.197):

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
Procedure 4 (from device):
http://192.168.20.222/ (approved):

<code>--2019-07-12 16:04:46-- http://192.168.20.222/</code>
<code>Connecting to 192.168.20.222:80... connected.</code>
<code>HTTP request sent, awaiting response... 200 OK</code>
<code>Length: 10701 (10K) [text/html]</code>
<code>Saving to: ‘index.html.9’</code>
<code>   0K ....</code>
<code>100% 104K=0.1s</code>
<code>2019-07-12 16:04:46 (104 KB/s) - ‘index.html.9’ saved [10701/10701]</code>
<hr>
Procedure 5 (from laptop to pi):

<code>[mud@localhost ~]$ wget 192.168.20.222</code>
<code>--2019-07-12 16:08:28--</code> http://192.168.20.222/
<code>Connecting to 192.168.20.222:80... failed: Connection refused.</code>
<hr>
Procedure 6 (from device):
http://192.168.20.232/ (unapproved):

<code>--2019-07-12 16:06:35-- http://192.168.20.232/</code>
<code>Connecting to 192.168.20.232:80... failed: Connection refused.</code>
<hr>
http://192.168.20.110:443/ (unapproved):

<code>--2019-07-12 16:06:16-- http://192.168.20.110:443/</code>
<code>Connecting to 192.168.20.110:443... failed: Connection refused.</code>
<hr>
http://192.168.20.117/ (unapproved):

<code>--2019-07-12 16:06:01-- http://192.168.20.117/</code>
<code>Connecting to 192.168.20.117:80... failed: Connection refused.</code>
<hr>
http://192.168.20.181/ (unapproved):

<code>--2019-07-12 16:05:39-- http://192.168.20.181/</code>
</td></tr>
</table>

Test Case Field	Description
	Connecting to 192.168.20.181:80... failed: Connection refused. *http://192.168.20.183/* (unapproved): --2019-07-12 16:05:11-- *http://192.168.20.183/* **Connecting to 192.168.20.183:80... failed: Connection refused.** Procedure 7 (from laptop to Pi): [mud@localhost ~]$ wget 192.168.20.222 --2019-07-12 16:12:03-- http://192.168.20.222/ **Connecting to 192.168.20.222:80... failed: Connection refused.** **Manufacturer:** Procedure 4 (from device): *http://192.168.20.183/* (approved): --2019-07-12 15:57:00-- *http://192.168.20.183/* **Connecting to 192.168.20.183:80... connected.** **HTTP request sent, awaiting response... 200 OK** Length: 10701 (10K) [text/html] Saving to: 'index.html.21' 0K .... 100% 26.9M=0s 2019-07-12 15:57:00 (26.9 MB/s) - 'index.html.21' saved [10701/10701] Procedure 5 (from laptop to pi): [mud@localhost ~]$ wget 192.168.20.222 --2019-07-12 15:59:31-- http://192.168.20.222/ **Connecting to 192.168.20.222:80... failed: Connection refused.** Procedure 6 (from device):

Test Case Field	Description
	http://192.168.20.110:443/ (unapproved): `--2019-07-12 15:58:13-- http://192.168.20.110:443/` **`Connecting to 192.168.20.110:443... failed: Connection refused.`** *http://192.168.20.117/* (unapproved): `--2019-07-12 15:57:19-- http://192.168.20.117/` **`Connecting to 192.168.20.117:80... failed: Connection refused.`** *http://192.168.20.232/* (unapproved): `--2019-07-12 15:57:29-- http://192.168.20.232/` **`Connecting to 192.168.20.232:80... failed: Connection refused.`** http://192.168.20.197 (unapproved): `--2019-07-12 15:58:35-- http://192.168.20.197/` **`Connecting to 192.168.20.197:80... failed: Connection refused.`** Procedure 7 (from laptop to Pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-12 15:59:31--` http://192.168.20.222/ **`Connecting to 192.168.20.222:80... failed: Connection refused.`** **Same Manufacturer:** Procedure 8 (from device): *http://192.168.20.197/* (approved): `--2019-07-12 16:27:24-- http://192.168.20.197/` **`Connecting to 192.168.20.197:80... connected.`** **`HTTP request sent, awaiting response... 200 OK`** `Length: 10701 (10K) [text/html]` `Saving to: 'index.html.43'` `0K ..........` `100% 3.75M=0.003s` `2019-07-12 16:27:24 (3.75 MB/s) - 'index.html.43' saved [10701/10701]`

Test Case Field	Description
	Procedure 6 (from device): *http://192.168.20.183/* (unapproved): `--2019-07-12 16:27:36-- http://192.168.20.183/` **`Connecting to 192.168.20.183:80... failed: Connection refused.`** *http://192.168.20.181/* (unapproved): `--2019-07-12 16:28:11-- http://192.168.20.181/` **`Connecting to 192.168.20.181:80... failed: Connection refused.`** *http://192.168.20.142/* (unapproved): `--2019-07-12 16:27:48-- http://192.168.20.142/` **`Connecting to 192.168.20.142:80... failed: Connection refused.`** *http://192.168.20.117/* (unapproved): `--2019-07-12 16:28:20-- http://192.168.20.117/` **`Connecting to 192.168.20.117:80... failed: Connection refused.`** *http://192.168.20.110:443/* (unapproved): `--2019-07-12 16:27:59-- http://192.168.20.110:443/` **`Connecting to 192.168.20.110:443... failed: Connection refused.`** **Procedure 9:** `pi@same-manufacture-pi:~ $ wget 192.168.20.222` `--2019-07-24 20:49:51--` http://192.168.20.222/ **`Connecting to 192.168.20.222:80... failed: Connection refused.`**
Overall Results	Pass

358 As explained above, test IoT-6-v6 is identical to test IoT-6-v4 except that it uses IPv6, DHCPv6, and IANA
359 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

360 ### 3.1.2.7 Test Case IoT-7-v4

361 **Table 3-8: Test Case IoT-7-v4**

Test Case Field	Description
Parent Requirement	(CR-11) If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, the DHCP server must notify the MUD manager of any corresponding change to the DHCP state of the MUD-enabled IoT device, and the MUD manager should remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.
Testable Requirement	(CR-11.a) The MUD-enabled IoT device shall explicitly release the IP address lease (i.e., it sends a DHCP release message to the DHCP server). (CR-11.a.1) The DHCP server shall notify the MUD manager that the device's IP address lease has been released. (CR-11.a.2) The MUD manager should remove all policies associated with the disconnected IoT device that had been configured on the MUD PEP router/switch.
Description	Shows that when a MUD-enabled IoT device explicitly releases its IP address lease, the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch
Associated Test Case(s)	IoT-1-v4 (or IoT-1-v6 when IPv6 addressing is used)
Associated Cybersecurity Framework Subcategory(ies)	PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Fe-samemanufacturer.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the policies defined in the MUD file in Section 3.1.3 for the IoT device in question.

Test Case Field	Description
Procedure	1. As stipulated in the preconditions, right before this test, test IoT-1-v4 (or IoT-1-v6) must have been run successfully. Verify that the MUD PEP router/switch for the IoT device has been configured to enforce the policies listed in the preconditions section above for the IoT device in question. 2. Cause a DHCP release of the IoT device in question. 3. Check the log file for the MUD manager to verify that it was notified of the change of DHCP state. 4. Verify that all the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Expected Results	All of the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Actual Results	**Procedure 2:** `pi@main-pi-Build2:~ $ ` **`sudo dhclient -r`** --- **Procedure 3:** **`MUD Manager:`** `2019-07-11 18:57:30 DEBUG::GENERAL::2019-07-` **`11T18:57:29Z|DEL|Wired|DHCP|-|MUD|-|-|b8:27:eb:eb:6c:8b|192.168.20.226|main-pi-Build2|`** `2019-07-11 18:57:30 DEBUG::GENERAL::Executing on dhcpmasq info` `2019-07-11 18:57:30 INFO::GENERAL::DEL Device Action: IP: 192.168.20.226, MAC: b8:27:eb:eb:6c:8b` `2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/find_device_in_db.sh -d /etc/osmud/state/mudfiles/mudStateFile.txt -m b8:27:eb:eb:6c:8b -i 192.168.20.226 -s /etc/osmud/state/ipSets -a DELETE -u NONE` `2019-07-11 18:57:30 DEBUG::GENERAL::Return: 4864.` `2019-07-11 18:57:30 DEBUG::GENERAL::FinalReturn: 19.` `2019-07-11 18:57:30 ERROR::DEVICE_INTERFACE::FinalReturn: 19.` **`2019-07-11 18:57:30 DEBUG::CONTROLLER::MUD Controller: A delete event associated with a MUD file is being processed. IP: 192.168.20.226.`** **`2019-07-11 18:57:30 DEBUG::GENERAL::rm -f /tmp/osmud/*`**

Test Case Field	Description
	2019-07-11 18:57:30 DEBUG::GENERAL::cp /etc/osmud/state/ip-Sets/* /tmp/osmud **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/re-move_ip_fw_rule.sh -i 192.168.20.226 -m b8:27:eb:eb:6c:8b -d /tmp/osmud** **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/re-move_from_ipset.sh -d /tmp/osmud -i 192.168.20.226** **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/com-mit_ip_fw_rules.sh -d /etc/osmud/state/ipSets -t /tmp/osmud** **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/re-move_mud_db_entry.sh -d /etc/osmud/state/mudfiles/mudState-File.txt -i 192.168.20.226 -m b8:27:eb:eb:6c:8b** **2019-07-11 18:57:30 DEBUG::GENERAL::Success returned from for transaction** **Procedure 4:** **ROUTER/PEP:** # OSMUD start # # DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CONFIGURATION # config ipset option enabled 1 option name mudfiles_nist_getyikes_com-SMTD option match dest_ip option storage hash option family ipv4 option external mudfiles_nist_getyikes_com-SM config ipset option enabled 1 option name mudfiles_nist_getyikes_com-SMFD option match src_ip option storage hash option family ipv4 option external mudfiles_nist_getyikes_com-SM config ipset option enabled 1 option name mudfileserver-SMTD option match dest_ip option storage hash option family ipv4 option external mudfileserver-SM

Test Case Field	Description
	`config ipset` `    option enabled 1` `    option name mudfileserver-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfileserver-SM` `config ipset` `    option enabled 1` `    option name www_facebook_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external www_facebook_com-SM` `config ipset` `    option enabled 1` `    option name www_facebook_com-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external www_facebook_com-SM` `config ipset` `    option enabled 1` `    option name www_gmail_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external www_gmail_com-SM` `config ipset` `    option enabled 1` `    option name www_gmail_com-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external www_gmail_com-SM` `config rule` `        option enabled   '1'` `        option name      'mud_192.168.20.197_same-` `manufacture-pi_cl0-frdev'` `        option target    ACCEPT` `        option src       lan` `        option dest      wan` `        option proto     tcp`

Test Case Field	Description
	`option family ipv4` `option src_ip 192.168.20.197` `option dest_ip 198.71.233.87` `config rule` `option enabled '1'` `option name 'mud_192.168.20.197_same-manufacture-pi_cl0-todev'` `option target ACCEPT` `option src wan` `option dest lan` `option proto tcp` `option family ipv4` `option src_ip 198.71.233.87` `option dest_ip 192.168.20.197` `config rule` `option enabled '1'` `option name 'mud_192.168.20.197_same-manufacture-pi_myman0-frdev-SM'` `option target ACCEPT` `option src lan` `option dest lan` `option proto tcp` `option family ipv4` `option src_ip 192.168.20.197` `option ipset www_facebook_com-SMTD` `option dest_port 80:80` `config rule` `option enabled '1'` `option name 'mud_192.168.20.197_same-manufacture-pi_myman0-todev-SM'` `option target ACCEPT` `option src lan` `option dest lan` `option proto tcp` `option family ipv4` `option ipset www_facebook_com-SMFD` `option dest_ip 192.168.20.197` `option dest_port 80:80` `config rule` `option enabled '1'` `option name 'mud_192.168.20.197_same-manufacture-pi_REJECT-ALL-LOCAL-FROM'` `option target REJECT` `option src lan`

Test Case Field	Description
	option dest lan option proto all option family ipv4 option src_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufacture-pi_REJECT-ALL-LOCAL-TO' option target REJECT option src lan option dest lan option proto all option family ipv4 option src_ip any option dest_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufacture-pi_REJECT-ALL' option target REJECT option src lan option dest wan option proto all option family ipv4 option src_ip 192.168.20.197 # OSMUD end
Overall Results	Pass

362 As explained above, test IoT-7-v6 is identical to test IoT-7-v4 except that it uses IPv6, DHCPv6, and IANA
363 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

364 *3.1.2.8 Test Case IoT-8-v4*

365 **Table 3-9: Test Case IoT-8-v4**

Test Case Field	Description
Parent Requirement	(CR-11) If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, the DHCP server must notify the MUD manager of any

Test Case Field	Description
	corresponding change to the DHCP state of the MUD-enabled IoT device, and the MUD manager should remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.
Testable Requirement	(CR-11.b) The MUD-enabled IoT device's IP address lease shall expire. (CR-11.b.1) The DHCP server shall notify the MUD manager that the device's IP address lease has expired. (CR-11.b.2) The MUD manager should remove all policies associated with the affected IoT device that had been configured on the MUD PEP router/switch.
Description	Shows that when a MUD-enabled IoT device's IP address lease expires, the MUD-related configuration for that IoT device will be removed from its MUD PEP router/switch
Associated Test Case(s)	IoT-1-v4 (or IoT-1-v6 when IPv6 addressing is used)
Associated Cybersecurity Framework Subcategory(ies)	PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Fe-manufacturer1.json*
Preconditions	Test IoT-1-v4 (or IoT-1-v6) has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the policies defined in the MUD file in Section 3.1.3 for the IoT device in question.
Procedure	1. Configure the DHCP server to have a DHCP lease time of 60 minutes. 2. Run test IoT-1-v4 (or IoT-1-v6). 3. Verify that the MUD PEP router/switch for the IoT device has been configured to enforce the policies listed above for the IoT device in question. 4. Disconnect the IoT device in question from the network.

Test Case Field	Description
	5. After 60 minutes have elapsed, (1) look at the log file for the MUD manager to verify that it has received notice of the change of DHCP state, and (2) verify that all of the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Expected Results	Once 60 minutes have elapsed after disconnecting the IoT device from the network, all of the configuration rules listed above have been removed from the MUD PEP router/switch for the IoT device in question.
Actual Results	**Procedures 1–4:** **Completed; excluded for brevity** **Procedure 5:** 1. MUD MANAGER: **2019-07-12 17:34:49 DEBUG::GENERAL::2019-07-12T17:34:49Z|DEL|Wired|DHCP|-|MUD|-|-|b8:27:eb:a2:88:f3|192.168.20.184|manufacturer-pi|** 2019-07-12 17:34:49 DEBUG::GENERAL::Executing on dhcpmasq info 2019-07-12 17:34:49 INFO::GENERAL::DEL Device Action: IP: 192.168.20.184, MAC: b8:27:eb:a2:88:f3 2019-07-12 17:34:49 DEBUG::GENERAL::/etc/osmud/find_device_in_db.sh -d /etc/osmud/state/mudfiles/mudStateFile.txt -m b8:27:eb:a2:88:f3 -i 192.168.20.184 -s /etc/osmud/state/ipSets -a DELETE -u NONE 2019-07-12 17:34:49 DEBUG::GENERAL::Return: 3328. 2019-07-12 17:34:49 DEBUG::GENERAL::FinalReturn: 13. 2019-07-12 17:34:49 ERROR::DEVICE_INTERFACE::FinalReturn: 13. **2019-07-12 17:34:49 DEBUG::CONTROLLER::MUD Controller: A delete event associated with a MUD file is being processed. IP: 192.168.20.184**.2019-07-12 17:34:49 DEBUG::GENERAL::rm -f /tmp/osmud/* 2019-07-12 17:34:49 DEBUG::GENERAL::cp /etc/osmud/state/ipSets/* /tmp/osmud 2019-07-12 17:34:49 DEBUG::GENERAL::/etc/osmud/**remove**_ip_fw_rule.sh -i 192.168.20.184 -m b8:27:eb:a2:88:f3 -d /tmp/osmud 2019-07-12 17:34:49 DEBUG::GENERAL::/etc/osmud/**remove**_from_ipset.sh -d

Test Case Field	Description
	`/tmp/osmud -i 192.168.20.184` `2019-07-12 17:34:49` `DEBUG::GENERAL::/etc/osmud/commit_ip_fw_rules.sh -d /etc/osmud/state/ipSets -t /tmp/osmud` `2019-07-12 17:34:50` `DEBUG::GENERAL::/etc/osmud/`**`remove`**`_mud_db_entry.sh -d /etc/osmud/state/mudfiles/mudStateFile.txt -i 192.168.20.184 -m b8:27:eb:a2:88:f3` **`2019-07-12 17:34:50 DEBUG::GENERAL::Success returned from for transaction`** 2. Router/PEP: `# OSMUD start` `#` `# DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-FIGURATION` `#` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfiles_nist_getyikes_com-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4` `    option external mudfiles_nist_getyikes_com-SM` `config ipset` `    option enabled 1` `    option name mudfileserver-SMTD` `    option match dest_ip` `    option storage hash` `    option family ipv4` `    option external mudfileserver-SM` `config ipset` `    option enabled 1` `    option name mudfileserver-SMFD` `    option match src_ip` `    option storage hash` `    option family ipv4`

Test Case Field	Description
	`` option external mudfileserver-SM config ipset option enabled 1 option name www_facebook_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_facebook_com-SMFD option match src_ip option storage hash option family ipv4 option external www_facebook_com-SM config ipset option enabled 1 option name www_gmail_com-SMTD option match dest_ip option storage hash option family ipv4 option external www_gmail_com-SM config ipset option enabled 1 option name www_gmail_com-SMFD option match src_ip option storage hash option family ipv4 option external www_gmail_com-SM config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.197 option dest_ip 198.71.233.87 config rule option enabled '1' ```

Test Case Field	Description
	`` option name 'mud_192.168.20.197_same-manufac- ture-pi_cl0-todev' option target ACCEPT option src wan option dest lan option proto tcp option family ipv4 option src_ip 198.71.233.87 option dest_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_myman0-frdev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option src_ip 192.168.20.197 option ipset www_facebook_com-SMTD option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_myman0-todev-SM' option target ACCEPT option src lan option dest lan option proto tcp option family ipv4 option ipset www_facebook_com-SMFD option dest_ip 192.168.20.197 option dest_port 80:80 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_REJECT-ALL-LOCAL-FROM' option target REJECT option src lan option dest lan option proto all option family ipv4 option src_ip 192.168.20.197 config rule option enabled '1' ```

Test Case Field	Description
	option name 'mud_192.168.20.197_same-manufac- ture-pi_REJECT-ALL-LOCAL-TO' option target REJECT option src lan option dest lan option proto all option family ipv4 option src_ip any option dest_ip 192.168.20.197 config rule option enabled '1' option name 'mud_192.168.20.197_same-manufac- ture-pi_REJECT-ALL' option target REJECT option src lan option dest wan option proto all option family ipv4 option src_ip 192.168.20.197 # OSMUD end
Overall Results	Pass

As explained above, test IoT-8-v6 is identical to test IoT-8-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

3.1.2.9 Test Case IoT-9-v4

Table 3-10: Test Case IoT-9-v4

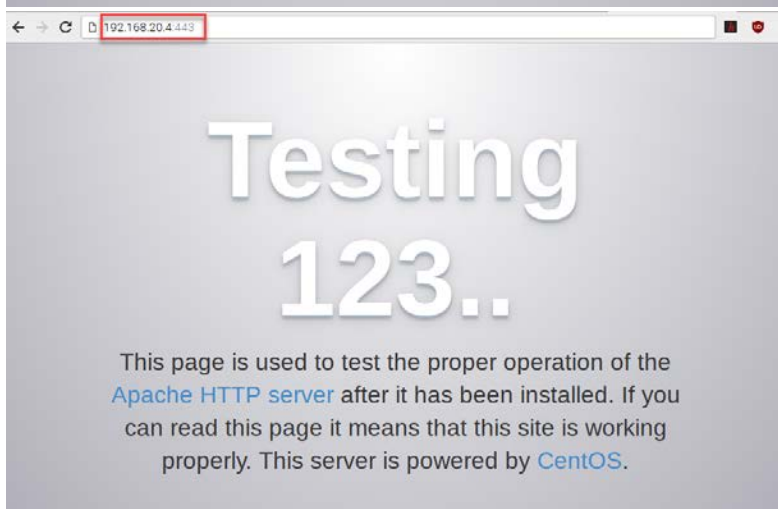
Test Case Field	Description
Parent Requirements	(CR-13) The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the MUD PEP router/switch will get configured with all possible instantiations of that rule, insofar as each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the MUD PEP router/switch.
Testable Requirements	(CR-13.a) The MUD file for a device shall contain a rule involving an external **domain that can resolve** to multiple IP addresses when queried

Test Case Field	Description
	by the MUD PEP router/switch. An ACL for permitting access to each of those IP addresses will be inserted into the MUD PEP router/switch for the device in question, and the device will be permitted to communicate with all of those IP addresses.
Description	Shows that if a domain in a MUD file rule resolves to multiple IP addresses when the address resolution is queried by the network gateway, then 1. ACLs instantiating that MUD file rule corresponding to each of these IP addresses will be configured in the gateway for the IoT device associated with the MUD file, and 2. the IoT device associated with the MUD file will be permitted to communicate with all of the IP addresses to which that domain resolves
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Yikesmain.json*
Preconditions	1. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 2. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 3.1.3. (Therefore, the MUD file used in the test permits the device to send data to *www.updateserver.com*.) 3. The tester has access to a DNS server that will be used by the MUD PEP router/switch and can configure it so that it will resolve the domain *www.updateserver.com* to any of these addresses when queried by the MUD PEP router/switch: x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.

Test Case Field	Description
	4. There is an update server running at each of these three IP addresses.
Procedure	1. Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 2. Run test IoT-1-v4 (or IoT-1-v6). The result should be that the MUD PEP router/switch has been configured to explicitly permit the IoT device to initiate communication with *www.updateserver.com*. 3. Verify that the MUD PEP router/switch has been configured with ACLs that permit the IoT device to send data to IP addresses x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. 4. Have the device in question attempt to connect to x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to permit the IoT device to send data to IP addresses x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. The IoT device is permitted to send data to each of the update servers at these addresses.
Actual Results	**Procedures 1–2:** **Completed; excluded for brevity** **Procedure 3:** **MUD MANAGER:** **2019-07-15 20:28:32 DEBUG::GENERAL::2019-07-15T20:28:31Z|NEW|Wired|DHCP|1,28,2,3,15,6,119,12,44,47,26,121,42|MUD|https://mudfiles.nist.getyikes.com/yikesmain.json|-|b8:27:eb:eb:6c:8b|192.168.20.222|main-pi-Build2|** 2019-07-15 20:28:32 DEBUG::GENERAL::Executing on dhcpmasq info 2019-07-15 20:28:32 INFO::GENERAL::NEW Device Action: IP: 192.168.20.222, MAC: b8:27:eb:eb:6c:8b 2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() doing it now.... 2019-07-15 20:28:32 DEBUG::COMMUNICATION::https://mudfiles.nist.getyikes.com/yikesmain.json 2019-07-15 20:28:32 DEBUG::COMMUNICATION::Found HTTPS

Test Case Field	Description
	2019-07-15 20:28:32 DEBUG::COMMUNICATION::in write data 2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() success 2019-07-15 20:28:32 DEBUG::COMMUNICATION::MUD File Server returned success state. 2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() doing it now.... 2019-07-15 20:28:32 DEBUG::COMMUNICATION::https://mudfiles.nist.getyikes.com/yikesmain.p7s 2019-07-15 20:28:32 DEBUG::COMMUNICATION::Found HTTPS 2019-07-15 20:28:32 DEBUG::COMMUNICATION::in write data **2019-07-15 20:28:32 DEBUG::COMMUNICATION::curl_easy_perform() success** **2019-07-15 20:28:32 DEBUG::COMMUNICATION::MUD File Server returned success state.** **2019-07-15 20:28:32 DEBUG::MUD_FILE_OPERATIONS::IN ****NEW**** MUD and SIG FILE RETRIEVED!!!** **2019-07-15 20:28:32 DEBUG::GENERAL::IN ****NEW**** validateMudFileWithSig()** **2019-07-15 20:28:32 DEBUG::GENERAL::openssl cms -verify -in /etc/osmud/state/mudfiles/yikesmain.p7s -inform DER -content /etc/osmud/state/mudfiles/yikesmain.json -purpose any > /dev/null** 2019-07-15 20:28:32 DEBUG::GENERAL::IN ****NEW**** executeMudWithDhcpContext() 2019-07-15 20:28:32 DEBUG::GENERAL::/etc/osmud/create_mud_db_entry.sh -d /etc/osmud/state/mudfiles/mudStateFile.txt -i 192.168.20.222 -m b8:27:eb:eb:6c:8b -c main-pi-Build2 -u https://mudfiles.nist.getyikes.com/yikesmain.json -f /etc/osmud/state/mudfiles/yikesmain.json **[Logs omitted for brevity]** **2019-07-15 20:28:32 DEBUG::GENERAL::***www.updateserver.com* **2019-07-15 20:28:33 DEBUG::GENERAL::192.168.20.4** **2019-07-15 20:28:33 DEBUG::GENERAL::192.168.20.238** **2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 192.168.20.4 -b 443:443 -p tcp -n cl2-frdev -t ACCEPT -f all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222** **2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/create_ip_fw_rule.sh -s lan -d wan -i 192.168.20.222 -a any -j 192.168.20.238 -b 443:443 -p tcp -n cl2-frdev -t ACCEPT -f**

Test Case Field	Description
	all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222 **[Logs omitted for brevity]** 2019-07-15 20:28:33 DEBUG::GENERAL::**Success returned from for transaction**
	Router/PEP: config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 **option src_ip 192.168.20.222** **option dest_ip 192.168.20.4** option dest_port 443:443 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-frdev' option target ACCEPT option src lan option dest wan option proto tcp option family ipv4 **option src_ip 192.168.20.222** **option dest_ip 192.168.20.238** option dest_port 443:443
	Procedure 4:

Test Case Field	Description
	 
Overall Results	Pass

Test case IoT-9-v6 is identical to test case IoT-9-v4 except that IoT-9-v6 uses IPv6 addresses rather than IPv4 addresses.

372 ### 3.1.2.10 Test Case IoT-10-v4

373 **Table 3-11: Test Case IoT-10-v4**

Test Case Field	Description
Parent Requirements	(CR-12) The IoT DDoS example implementation shall include a MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed for the MUD file indicated by the MUD URL**.** The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.
Testable Requirements	(CR-12.a) The MUD manager shall check if the file associated with the MUD URL is present in its cache and shall determine that it is. (CR-12.a.1) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If so, the MUD manager shall apply the contents of the cached MUD file. (CR-12.a.2) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file. If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the cached MUD file for that device's MUD URL, assuming that the amount of time that has elapsed since the cached MUD file was retrieved is less than or equal to the number of hours in the file's cache-validity value. If the cache validity has expired for the respective file, the MUD manager should fetch a new MUD file from the MUD file server.
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2, PR.PT-3

Test Case Field	Description
IoT Device(s) Under Test	To be determined (TBD) (Not testable in Build 2's preproduction of Yikes!)
MUD File(s) Used	TBD (Not testable in Build 2's preproduction of Yikes!)
Preconditions	1. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 2. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 3.1.3.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Run test IoT-1-v4 (or IoT-1-v6). 2. Within 24 hours (i.e., within the cache-validity period for the MUD file) of running test IoT-1-v4 (or IoT-1-v6), verify that the IoT device that was connected during test IoT-1-v4 (or IoT-1-v6) is still up and running on the network. Power on a second IoT device that has been configured to emit the same MUD URL as the device that was connected during test IoT-1-v4 (or IoT-1-v6), and connect it to the test network. This should set in motion the following series of steps, which should occur automatically. 3. The IoT device automatically emits a DHCPv4 message containing the device's MUD URL (IANA code 161). (Note that in the v6 version of this test, IPv6, DHCPv6, and IANA code 112 will be used.) 4. The DHCP server receives the DHCPv4 message containing the IoT device's MUD URL. 5. The DHCP server offers an IP address lease to the newly connected IoT device. 6. The IoT device requests this IP address lease, which the DHCP server acknowledges. 7. The DHCP server sends the MUD URL to the MUD manager. 8. The MUD manager determines that it has this MUD file cached and checks that the amount of time that has elapsed since the cached

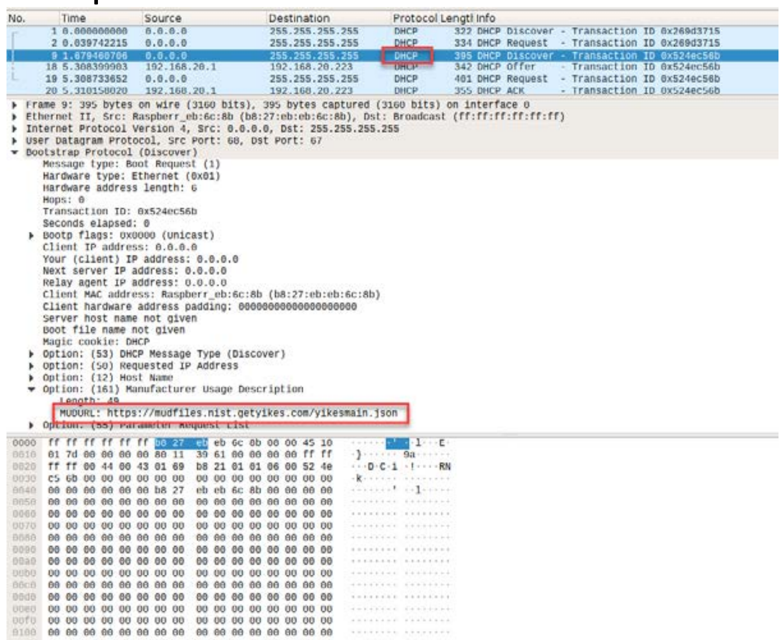
Test Case Field	Description
	file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If the cache validity has been exceeded, the MUD manager will fetch a new MUD file. (Run the test both ways—with a cache-validity period that has expired and with one that has not.) 9. The MUD manager translates the MUD file's contents into appropriate route filtering rules and installs these rules onto the MUD PEP for the IoT device in question so that this router/switch is now configured to enforce the policies specified in the MUD file.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. The expected configuration should resemble the following. **Cache is valid** (the MUD manager does NOT retrieve the MUD file from the MUD file server)**:** TBD (Not testable in Build 2's preproduction of Yikes!) **Cache is not valid** (the MUD manager does retrieve the MUD file from the MUD file server)**:** TBD (Not testable in Build 2's preproduction of Yikes!) All protocol exchanges described in steps 1–9 above are expected to occur and can be viewed via Wireshark if desired. If the router/switch does not get configured in accordance with the MUD file, each exchange of DHCP and MUD-related protocol traffic should be viewed on the network via Wireshark to determine which transactions did not proceed as expected, and the observed and absent protocol exchanges should be described here.
Actual Results	TBD (Not testable in Build 2's preproduction of Yikes!)
Overall Results	TBD (Not testable in Build 2's preproduction of Yikes!)

Test case IoT-10-v6 is identical to test case IoT-10-v4 except that IoT-10-v6 tests requirement CR-1.a.2, whereas IoT-10-v4 tests requirement CR-1.a.1. Hence, as explained above, test IoT-10-v6 uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

3.1.2.11 Test Case IoT-11-v4

Table 3-12: Test Case IoT-11-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL).
Testable Requirements	(CR-1.a) Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one MUD URL, in https scheme, within the DHCP transaction. (CR-1.a.1) The DHCP server shall be able to receive DHCPv4 DISCOVER and REQUEST with IANA code 161 (OPTION_MUD_URL_V4) from the MUD-enabled IoT device.
Description	Shows that the IoT DDoS example implementation includes IoT devices that can emit a MUD URL via DHCP
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*Yikesmain.json*
Preconditions	Device has been developed to emit MUD URL in DHCP transaction
Procedure	1. Power on a device and connect it to the network. 2. Verify that the device emits a MUD URL in a DHCP transaction. (Use Wireshark to capture the DHCP transaction with options present.)
Expected Results	DHCP transaction with MUD option 161 enabled and MUD URL included

Test Case Field	Description
Actual Results	**MUD option included in DHCP transaction:** 
Overall Results	Pass

3.1.3 MUD Files

This section contains the MUD files that were used in the Build 2 functional demonstration.

3.1.3.1 Fe-controller.json

The complete Fe-controller.json MUD file has been linked to this document. To access this MUD file please click the link below.

[Fe-controller.json]

3.1.3.2 Fe-localnetwork-from2.json

The complete Fe-localnetwork-from2.json MUD file has been linked to this document. To access this MUD file please click the link below.

[Fe-localnetwork-from2.json]

3.1.3.3 Fe-localnetwork-to2.json

The complete fe-localnetwork-to2.json MUD file has been linked to this document. To access this MUD file please click the link below.

Fe-localnetwork-to2.json

3.1.3.4 Fe-manufacturer1.json

The complete Fe-manufacturer1.json MUD file has been linked to this document. To access this MUD file please click the link below.

Fe-manufacturer1.json

3.1.3.5 Fe-manufacturer2.json

The complete Fe-manufacturer2.json MUD file has been linked to this document. To access this MUD file please click the link below.

Fe-manufacturer2.json

3.1.3.6 Fe-mycontroller.json

The complete Fe-mycontroller.json MUD file has been linked to this document. To access this MUD file please click the link below.

Fe-mycontroller.json

3.1.3.7 Fe-samemanufacturer-from2.json

The complete Fe-samemanufacturer-from2.json MUD file has been linked to this document. To access this MUD file please click the link below.

Fe-samemanufacturer-from2.json

3.1.3.8 Fe-samemanufacturer-to2.json

The complete Fe-samemanufacturer-to2.json MUD file has been linked to this document. To access this MUD file please click the link below.

Fe-samemanufacturer-to2.json

3.1.3.9 Yikesmain.json

The complete Yikesmain.json MUD file has been linked to this document. To access this MUD file please click the link below.

Yikesmain.json

3.2 Demonstration of Non-MUD-Related Capabilities

In addition to supporting MUD, Build 2 supports capabilities with respect to device discovery, identification, categorization, and application of traffic rules based on device make and model. Table 3-13 lists the non-MUD-related capabilities that were demonstrated for Build 2. Before examining these capabilities, however, it is instructive to define terminology and provide an overview of Build 2's non-MUD-related capabilities.

3.2.1 Terminology

The terminology that is used to describe non-MUD capabilities is not standardized. To avoid confusion, we offer the following definitions for use in this section:

- Device discovery—detection that a device is on the network
- Device identity—an identifier that a build assigns to the device and uses to keep track of the device. In Build 2, when a device is discovered, it is assigned a unique identity.
- Device identification—determination of the device's make (i.e., manufacturer) and model. In Build 2, each make and model combination may be associated with internet traffic rules that, if present, will be applied to all devices having that same make and model.
- Category—a predefined class to which devices are assigned based on their make and model. Each category is associated with traffic rules (for both local traffic and internet traffic) that will be applied to all devices in that category.
- Device categorization—determination of which of the build's predefined categories to which to assign the device. The device's make and model determine its category, e.g., if the device is determined to be a Samsung Galaxy S8, it is placed in the phone category.
- Traffic policy—a set of traffic rules that may be associated with a category of devices or a set of devices having the same make and model; the traffic policy determines to what other local devices and remote domains these devices are permitted to initiate communication.

3.2.2 General Overview of Build 2's Non-MUD Functionality

Once Build 2 discovers a device on the network, it applies the following non-MUD capabilities to it:

- automatic (if possible) identification of the device's make (i.e., manufacturer) and model
- categorization of the device based on its make and model
- association of the device category with a traffic policy that indicates what communication devices in that category are permitted to initiate. This policy consists of rules that apply to both local and internet communications. The rules in this policy can be viewed using the Yikes! User Interface (UI). By selecting the specific category (e.g., "cellphone" or "computer") on the UI Categories page, one can see two categories of rules, Local Network and Internet:

- Internet rules that may be set to either
  - Allow All Internet Traffic, which indicates that all devices in this category are permitted to initiate communications to all internet domains

  or

  - IoT Specific Sites, which indicates that there may be additional rules configured on the router that apply to specific makes and models of devices in this category and that restrict the internet sites to which those devices are permitted to initiate communications. (These per-make-and-model rules are stored in the cloud and viewed using the Yikes! UI. The IoT Devices tab displays the list of domain names to which communications may be initiated. For this version of the Yikes! cloud, these rules were set manually based on Build 2 test cases.)
- Local Network rules that may be set to either
  - Allow All, which, if set, indicates that devices in this category are permitted to initiate communications to all other devices on the local network

  or

  - any combination of other categories (cell phones, printers, tablets, printers, etc.) These indicate the other categories of devices on the local network to which devices in this category are permitted to initiate communications.

3.2.3 Non-MUD-Related Functional Capabilities

Table 3-13 lists the non-MUD-related capabilities that were demonstrated for Build 2. We use the letter "Y" as a prefix for these functional capability identifiers in the table below because these capabilities are specific to Build 2, which uses Yikes! equipment.

Table 3-13: Non-MUD-Related Functional Capabilities Demonstrated

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
Y-1	**Device Identification–**The device is detected, and its make and model are identified upon connection to the network.			

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
Y-1.a		The non-MUD-capable device's **make and model are correctly identified** based on some combination of information such as the device's media access control (MAC) address, DHCP header information, and lookup in repositories.		YnMUD-1-v4, YnMUD-1-v6
Y-1.b		The non-MUD-capable **device's make and model cannot be identified.**		YnMUD-1-v4, YnMUD-2-v6
Y-1.c		The non-MUD-capable **device's make and model can be assigned manually.**		YnMUD-2-v4, YnMUD-3-v6
Y-2	**Device Categorization–**The device is correctly categorized according to its type (e.g., phone, printer, computer, watch) upon connection to the network.			
Y-2.a		The non-MUD-capable **device is correctly categorized based on its make and model.**	The device make and model were determined using some combination of MAC address, DHCP header information, and lookup in repositories.	YnMUD-1-v4, YnMUD-1-v6

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
Y-2.b		The **make and model of the non-MUD-capable device cannot be determined.**	The non-MUD-capable **device is designated as uncategorized.**	YnMUD-1-v4, Yn-MUD-1-v6
Y-2.c		The non-MUD-capable **device's category can be assigned manually.**		YnMUD-2-v4, Yn-MUD-3-v6
Y-3	**Rules regarding initiation of (south-north) communications to internet sites** by the non-MUD-capable device **are enforced according to** rules associated with **the device's category and, possibly, its make and model.**			
Y-3.a		The device's category has **the Allow All Internet Traffic rule set** (i.e., the IoT Specific Sites rule is not set).	The device will be **permitted to connect to any internet location.**	YnMUD-3-v4, Yn-MUD-3-v6
Y-3.b		The device's category has **the IoT Specific Sites rule set,** indicating that there may be **rules associated with specific makes and models of devices in this category** that further restrict the internet locations		

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
		to which those devices are able to initiate communications.		
Y-3.b.1			There **are (south to north) rules associated with the device's make and model,** so the device will be **allowed to initiate communications with the internet sites permitted by those rules but prohibited from initiating communications to all other internet sites.**	YnMUD-3-v4, YnMUD-3-v6
Y-3.b.2			There are **no (south to north) rules associated with a device's make and model,** so that device will be **allowed to initiate communications with all internet sites.**	YnMUD-3-v4, YnMUD-3-v6
Y-3.c			There **are (north to south) rules associated with a device's make and model,** so that device will be **allowed to receive communications from the internet sites permitted by the rules but prohibited**	**N/A for IPv4** due to NAT

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
			from receiving communications from all other internet sites.	
Y-3.d			There are **no (north to south) rules associated with a device's make and model,** so that device will be **allowed to receive communications from all internet sites.**	**N/A for IPv4** due to NAT
Y-4	**Lateral (east-west) communications** of the non-MUD-capable device to other devices on the local network are **enforced according to the policy associated with the device's category.**			
Y-4.a		**A rule** associated with the device's category **permits the device to initiate communications with local devices in category X, but** there is **no such rule that permits the device to initiate communications with local devices in category Y.**		YnMUD-4-v4, YnMUD-4-v6

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
Y-4.a.1			The device will be allowed to **initiate communications to** any local device that is in **category X.**	YnMUD-4-v4, YnMUD-4-v6
Y-4.a.2			The device will be **prohibited from initiating communications to** any local device that is in **category Y.**	YnMUD-4-v4, YnMUD-4-v6
Y-5	**In response to threat information, all devices on the local network are prohibited from visiting specific domains and IP addresses.**			
Y-5.a		Threat intelligence indicates a **specific internet domain that should not be trusted.**	**Devices are prohibited from initiating communications to the internet domain listed in the threat intelligence. In addition, they are prohibited from initiating communications to any other domains and IP addresses that are associated with the same threat campaign as this domain.**	YnMUD-5-v4, YnMUD-5-v6

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
Y-5.b		Threat intelligence indicates a **specific IP address that should not be trusted.**	**Devices are prohibited from initiating communications to the IP address listed in the threat intelligence. In addition, they are prohibited from initiating communications to any other IP addresses and domains that are associated with the same threat campaign as this IP address.**	YnMUD-6-v4, YnMUD-6-v6
Y-5.c		Threat intelligence was received more than 24 hours prior, indicating domains and IP addresses that should not be trusted, and those domains and IP addresses were blocked by ACLs installed on the router.	**After 24 hours, these ACLs are no longer configured in the router.**	YnMUD-7-v4, YnMUD-7-v6

3.2.4 Exercises to Demonstrate the Above Non-MUD-Related Capabilities

This section contains the exercises that were performed to verify that Build 2 supports the non-MUD-related capabilities listed in Table 3-13.

To support these tests, the following domains must be available on the internet (i.e., outside the local network):

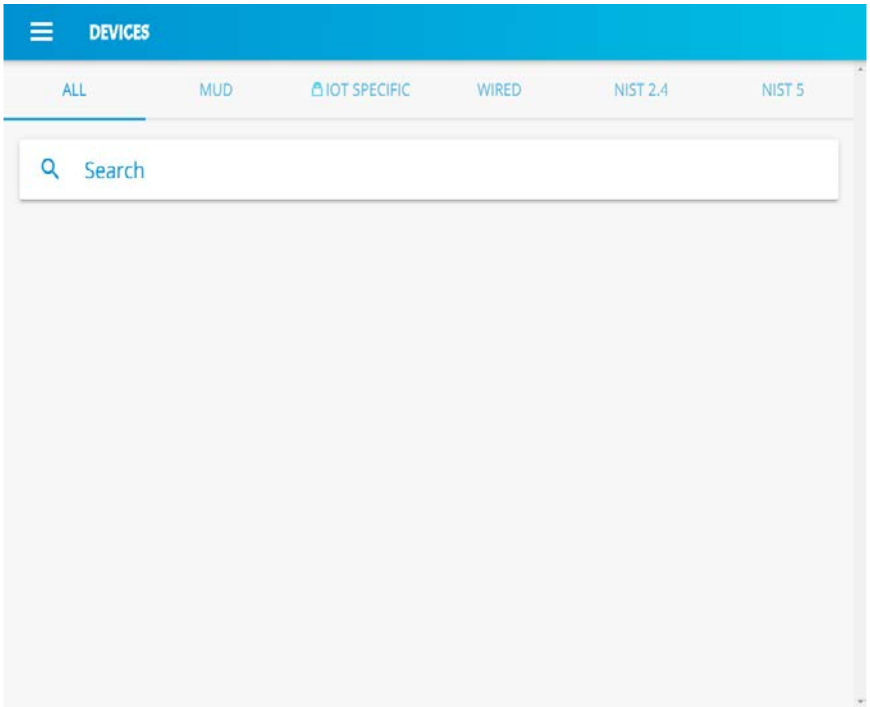
- www.google.com
- www.osmud.org
- www.trytechy.com

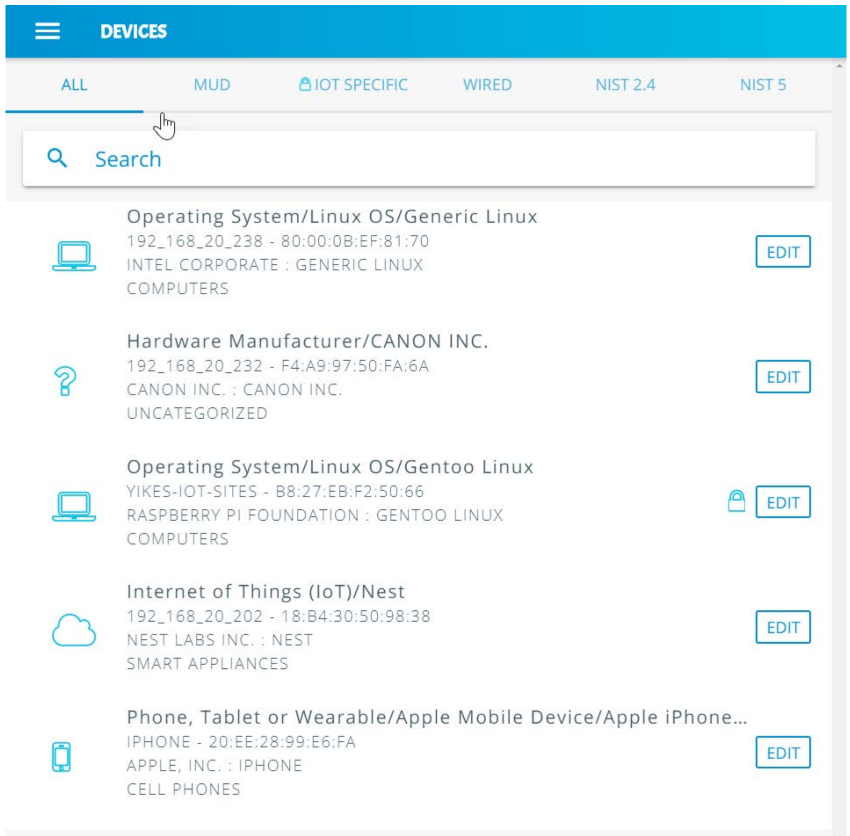
3.2.4.1 Exercise YnMUD-1-v4

Table 3-14: Exercise YnMUD-1-v4

Exercise Field	Description
Parent Capability	(Y-1) Device Identification–The device is detected, and its make and model are identified upon connection to the network. (Y-2) Device Categorization–The device is correctly categorized according to its type (e.g., phone, printer, computer, watch) upon connection to the network.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-1.a) The non-MUD-capable device's make and model are correctly identified based on some combination of information such as the device's MAC address, DHCP header information, and lookup in repositories. (Y-2.a) The non-MUD-capable device is correctly categorized based on its make and model. The device make and model were determined using some combination of MAC address, DHCP header information, and lookup in repositories. (Y-1.b) The non-MUD-capable device's make and model cannot be identified. (Y-2.b) The make and model of the non-MUD-capable device cannot be determined. The non-MUD-capable device is designated as uncategorized.
Description	Verify that upon detection, when possible, the make (i.e., manufacturer) and model of a non-MUD-capable device are identified correctly based on some combination of its MAC address, DHCP header information, and lookup through the Yikes! cloud service; the device is assigned to the correct category; and it is assigned a unique identity. In addition, verify that a non-MUD-capable device whose make and model cannot be determined will be assigned to the "uncategorized" category.
Associated Exercises	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, DE.AE-1, DE.CM-1

Exercise Field	Description
IoT Device(s) Used	- Laptop–with network-scanning software loaded - Cell phone–with network-scanning application loaded - Printer - Nest Camera to serve as an actual IoT device - Raspberry PI emulating an IoT device
Policy Used	N/A
Preconditions	The Yikes! router is installed on the local network and connected to the internet. The Yikes! account is set up and available to the user at https://nist.getyikes.com. The IoT devices listed above are available to be connected to the local network.
Procedure	1. Use the Yikes! UI to determine whether any devices are present (either active or inactive) on the network. 2. If any devices are present, they are to be deleted. Then verify that no devices are present (either active or inactive) on the network. 3. Connect each of the five devices above to the local network. 4. Validate that each device has appeared in Yikes! UI.
Demonstrated Results	Access the Yikes! UI, go to the Devices page, click the ALL tab, and verify that the following information is present, showing that each device has been given a unique identifier (not necessarily ID_X), has had its make and model correctly identified (if possible), and has been categorized appropriately: **Procedures 1–2:**

Exercise Field	Description
	 **Procedures 3–4:**

Exercise Field	Description
	

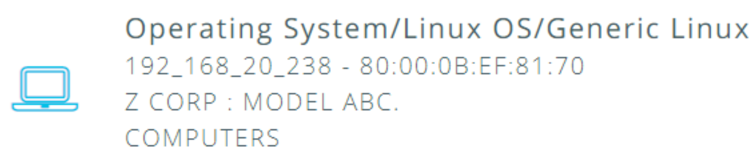
Device	Device ID	Make	Model	Category
Laptop	ID_1	Dell	E6540	Computer
Cell Phone	ID_2	Apple	iPhone 7	Cell Phone
Printer	ID_3	Canon	MX922	Uncategorized
Camera	ID_4	Nest	Indoor Cam	Smart Appliances
Test-PI	ID_5	Raspberry	Pi B+	Computer

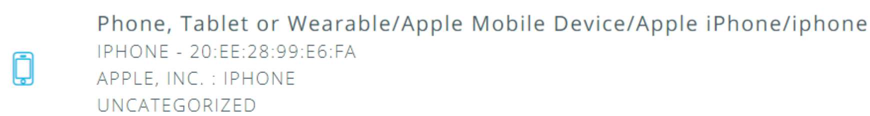
Exercise YnMUD-1-v6 is identical to exercise YnMUD-1-v4 except that it uses IPv6 instead of IPv4.

484 ### 3.2.4.2 Exercise YnMUD-2-v4

485 **Table 3-15: Exercise YnMUD-2-v4**

Exercise Field	Description
Parent Capability	(Y-1) Device Identification–The device is detected, and its make and model are identified upon connection to the network. (Y-2) Device Categorization–The device is correctly categorized according to its type (e.g., phone, printer, computer, watch) upon connection to the network.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-1.c) The non-MUD-capable device's make and model can be assigned manually. (Y-2.c) The non-MUD-capable device's category can be assigned manually.
Description	Verify that a non-MUD-capable device can have its make, model, or category assigned manually.
Associated Exercises	YnMUD-1-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-3
IoT Device(s) Used	Same as for exercise YnMUD-1-v4
Policy Used	N/A
Preconditions	Same as for exercise YnMUD-1-v4
Procedure	1. Run exercise YnMUD-1-v4. 2. Use the Yikes! UI to modify the make (i.e., manufacturer) of Device X to be Z Corp. 3. Use the Yikes! UI to modify the model of Device X to be Model ABC. 4. Use the Yikes! UI to modify the category of the cell phone to be Uncategorized.

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td>Demonstrated Results</td><td>Access the Yikes! UI, go to the Device tab, and verify that the following information is present:
Procedure 1: Completed; excluded for brevity
Procedures 2–3:


Procedure 4:


<table>
<tr><th>Device</th><th>Device ID</th><th>Make</th><th>Model</th><th>Category</th></tr>
<tr><td>Laptop</td><td>ID_1</td><td>Dell</td><td>E6540</td><td>Computer</td></tr>
<tr><td>Cell Phone</td><td>ID_2</td><td>Apple</td><td>iPhone7</td><td>Cell phone</td></tr>
<tr><td>Printer</td><td>ID_3</td><td>Canon</td><td>MX922</td><td>Uncategorized</td></tr>
<tr><td>Camera</td><td>ID_4</td><td>Nest</td><td>Indoor Cam</td><td>Smart Appliances</td></tr>
<tr><td>Test-PI</td><td>ID_5</td><td>Raspberry</td><td>Pi B+</td><td>Computer</td></tr>
</table></td></tr>
</table>

Exercise YnMUD-2-v6 is identical to exercise YnMUD-2-v4 except that it uses IPv6 instead of IPv4.

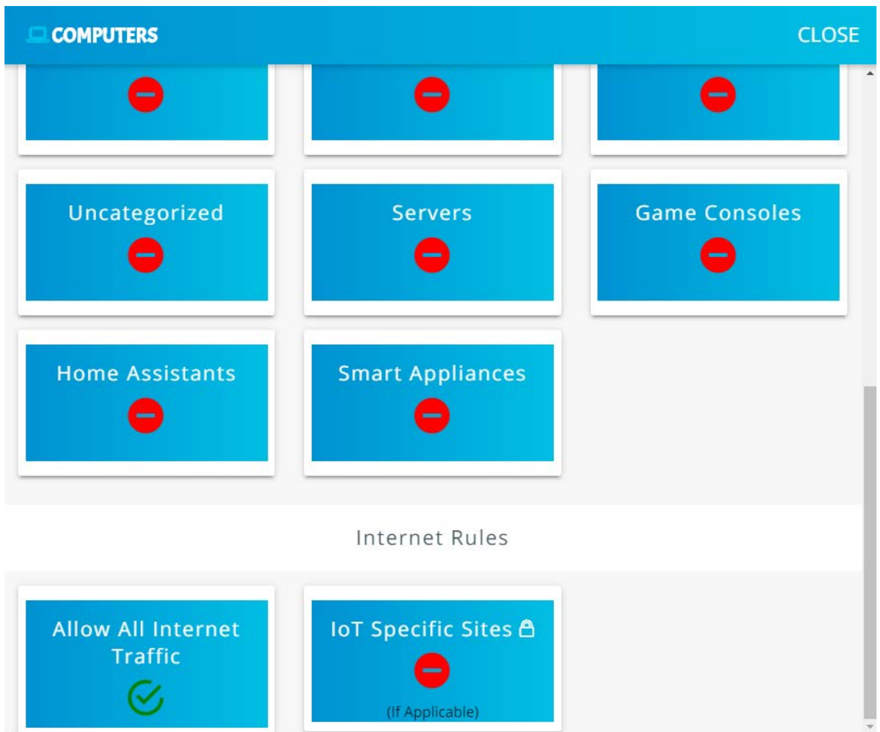
3.2.4.3 Exercise YnMUD-3-v4

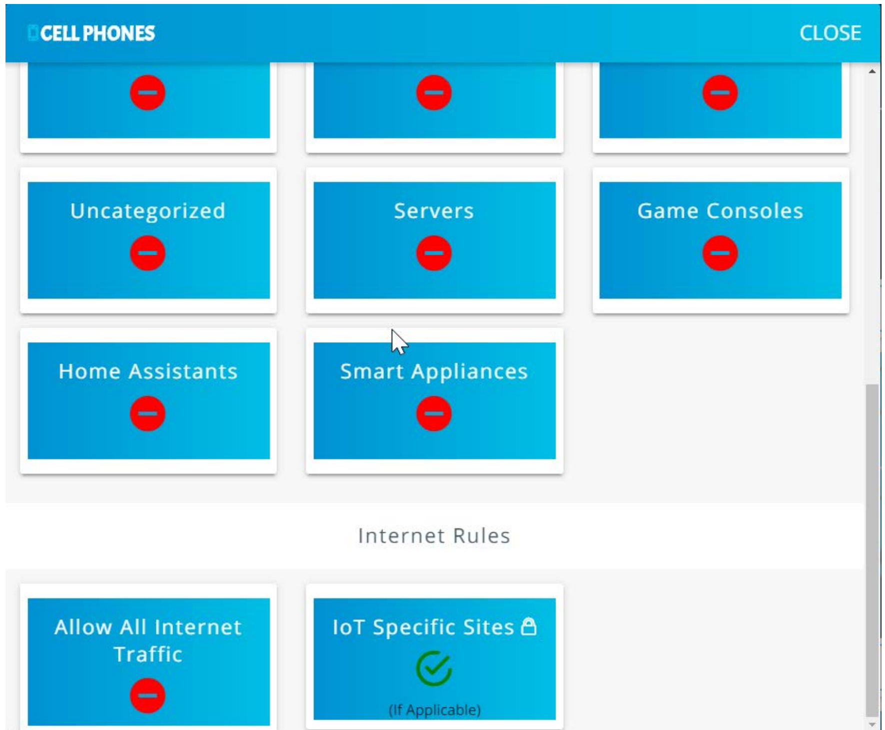
Table 3-16: Exercise YnMUD-3-v4

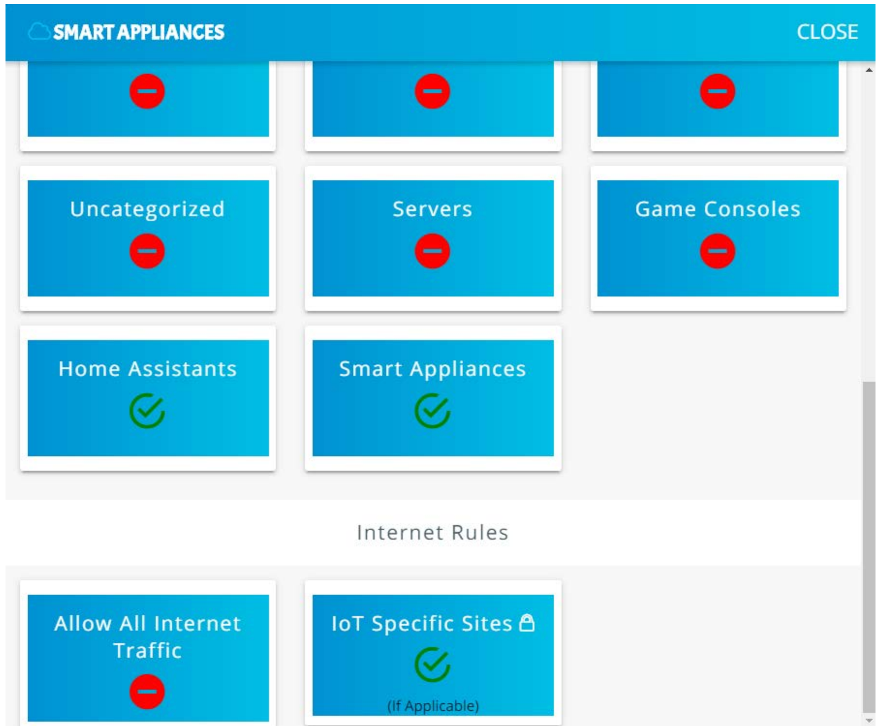
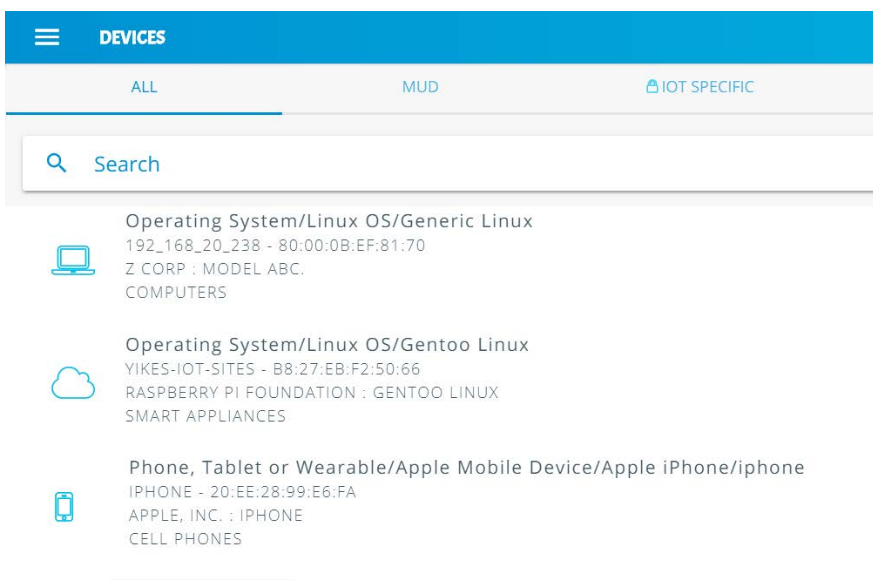
Exercise Field	Description
Parent Capability	(Y-3) Rules regarding initiation of (south-north) communications to internet sites by the non-MUD-capable device are enforced according to

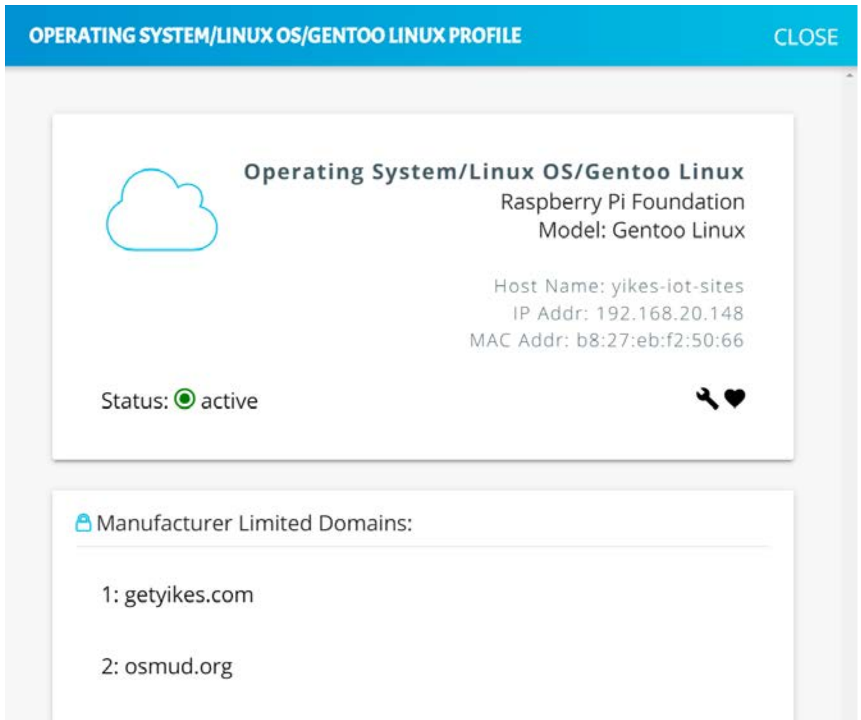
Exercise Field	Description
	rules associated with the device's category and, possibly, its make and model.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-3.a) The device's category has the Allow All Internet Traffic rule set (i.e., the IoT Specific Sites rule is not set). The device will be permitted to connect to any internet location. (Y-3.b) The device's category has the IoT Specific Sites rule set, indicating that there may be rules associated with specific makes and models of devices in this category that further restrict the internet locations to which those devices are able to initiate communications. (Y-3.b.1) There are (south to north) rules associated with the device's make and model, so the device will be allowed to initiate communications with the internet sites permitted by those rules but prohibited from initiating communications to all other internet sites. (Y-3.b.2) There are no (south to north) rules associated with a device's make and model, so that device will be allowed to initiate communications with all internet sites.
Description	Verify that once a device has been categorized, the device will be able to initiate communications to internet sites as constrained by any south-to-north rules that may be in place on the router that pertain to the device's make and model. In particular: - If the IoT Specific Sites rule is not set for the device's category, the device will be permitted to initiate communication with all internet sites. - If the IoT Specific Sites rule is set for this device's category and there are south-to-north rules on the router that apply to the device's make and model, the device will be restricted to initiating communications to only those internet sites permitted by those rules on the router. - If the IoT Specific Sites rule is set for this device's category but there are no south-to-north rules on the router that apply to the device's make and model, the device will not be permitted to initiate communication with any internet sites.
Associated Exercises	N/A

Exercise Field	Description
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, ID.AM-4, PR.AC-1, PR.AC-3, PR.AC-4, PR.AC-5
IoT Device(s) Used	- Laptop - iPhone 7 cell phone - Raspberry Pi
Policy Used	In the Yikes! UI, the Smart Appliances and Cell Phone internet rule is set to IoT Specific Sites. On the router, one ACL rule applies to the Raspberry Pi that permits it to visit www.getyikes.com and www.osmud.org, but there are no device-specific rules that apply to cell phones. On the router, there are no rules that apply to iPhone 7 devices. In the Yikes! UI, the Computer internet rule is set to Allow All Internet Traffic rather than to IoT Specific Sites.
Preconditions	The Smart Appliance, Cell Phone, and Computer category rules in the Yikes! UI and the ACL rules on the router are configured as described in the policy row above. (The presence of the Smart Appliances, Cell Phone, and Computer category rules can be verified by accessing the Yikes! UI. Using the UI, we should also be able to see the fully qualified domain names (FQDNs) of the sites that the rules permit each make and model of connected appliance and cell phone to access if any exist. The presence of the ACL rules can be verified only by logging in to the router.)
Procedure	1. Validate Yikes! UI configuration for Smart Appliances, Cell Phone, and Computer categories. 2. Connect the iPhone 7, Raspberry Pi, and laptop to the network. 3. Validate that the Raspberry Pi can browse to www.osmud.org and www.getyikes.com but not to www.google.com. 4. Validate that the iPhone 7 cannot browse to www.google.com, www.osmud.org, and www.getyikes.com. 5. Validate that a computer on the network can browse to www.google.com, www.osmud.org, and www.getyikes.com.

Exercise Field	Description
	6. Log in to the router to validate that the appropriate ACL rules are in place.
Demonstrated Results	Cell phone access is permitted and prohibited as expected in the procedure steps above. Computer access is permitted as expected. **Procedure 1:** **Computers**  **Cell Phones**

Exercise Field	Description
	 **Smart Appliances**

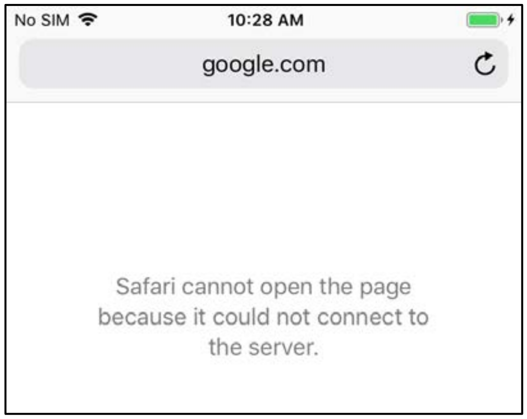
Exercise Field	Description
	 **Procedure 2:** 

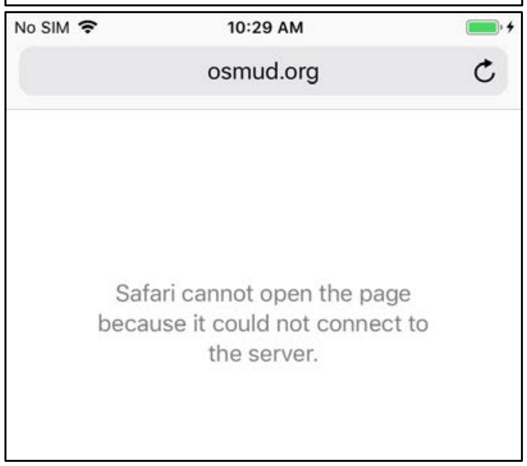
Exercise Field	Description
	Procedure 3: **Smart Appliance**  **Yikes! approved communication:** `pi@yikes-iot-sites:~ $ wget https://osmud.org` `--2019-07-29 10:28:56--  https://osmud.org/` `Resolving osmud.org (osmud.org)... 198.71.233.87` `Connecting to osmud.org (osmud.org)|198.71.233.87|:443... connected.` `HTTP request sent, awaiting response... 200 OK` `Length: unspecified [text/html]` `Saving to: ‘index.html.1’` `index.html.1          [ <=>              ]  24.12K  --.-KB/s    in 0.02s` `2019-07-29 10:28:58 (1.30 MB/s) - ‘index.html.1’ saved [24697]`

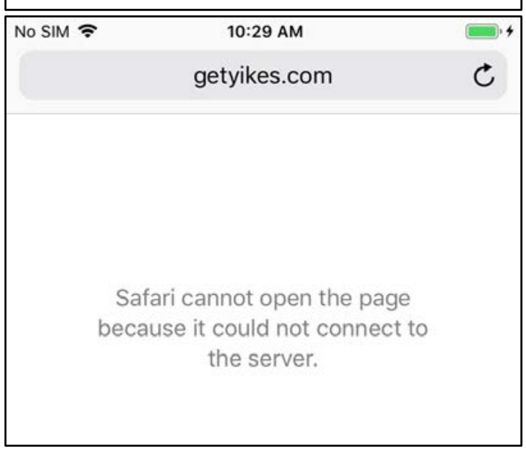
Exercise Field	Description
	pi@yikes-iot-sites:~ $ wget https://getyikes.com --2019-07-29 10:29:05-- https://getyikes.com/ Resolving getyikes.com (getyikes.com)... 54.213.16.153 Connecting to getyikes.com (getyikes.com)|54.213.16.153|:443... connected. HTTP request sent, awaiting response... 200 OK Length: 15759 (15K) [text/html] Saving to: 'index.html.2' index.html.2 100%[===================>] 15.39K --.-KB/s in 0.1s 2019-07-29 10:29:06 (119 KB/s) - 'index.html.2' saved [15759/15759] **Yikes! unapproved communication:** pi@yikes-iot-sites:~ $ wget https://www.google.com --2019-07-29 10:29:29-- https://www.google.com/ Resolving www.google.com (www.google.com)... 74.125.136.99, 74.125.136.103, 74.125.136.106, ... Connecting to www.google.com (www.google.com)|74.125.136.99|:443... failed: Connection refused. Connecting to www.google.com (www.google.com)|74.125.136.103|:443... failed: Connection refused. Connecting to www.google.com (www.google.com)|74.125.136.106|:443... failed: Connection refused. Connecting to www.google.com (www.google.com)|74.125.136.147|:443... failed: Connection refused. Connecting to www.google.com (www.google.com)|74.125.136.105|:443... failed: Connection refused. Connecting to www.google.com (www.google.com)|74.125.136.104|:443... failed: Connection refused. Connecting to www.google.com (www.google.com)|2607:f8b0:4002:c06::6a|:443... failed: Network is unreachable.

Procedure 4:

Cell Phone







Procedure 5:

Computers

Exercise Field	Description
	[mud@localhost ~]$ wget www.google.com --2019-07-23 14:47:52-- http://www.google.com/ Resolving www.google.com (www.google.com)... 172.217.164.68, 2607:f8b0:4002:c08::67 Connecting to www.google.com (www.google.com)|172.217.164.68|:80... connected. HTTP request sent, awaiting response... 200 OK Length: unspecified [text/html] Saving to: 'index.html.13' [<=>] 11,492 --.-K/s in 0.005s 2019-07-23 14:47:53 (2.30 MB/s) - 'index.html.13' saved [11492] [mud@localhost ~]$ wget osmud.org --2019-07-23 14:48:11-- http://osmud.org/ Resolving osmud.org (osmud.org)... 198.71.233.87 Connecting to osmud.org (osmud.org)|198.71.233.87|:80... connected. HTTP request sent, awaiting response... 301 Moved Permanently Location: https://osmud.org/ [following] --2019-07-23 14:48:11-- https://osmud.org/ Connecting to osmud.org (osmud.org)|198.71.233.87|:443... connected. HTTP request sent, awaiting response... 200 OK Length: unspecified [text/html] Saving to: 'index.html.14' [<=>] 24,697 --.-K/s in 0.009s 2019-07-23 14:48:11 (2.73 MB/s) - 'index.html.14' saved [24697] [mud@localhost ~]$ wget getyikes.com --2019-07-23 14:48:36-- http://getyikes.com/ Resolving getyikes.com (getyikes.com)... 54.213.16.153 Connecting to getyikes.com (getyikes.com)|54.213.16.153|:80... connected. HTTP request sent, awaiting response... 301 Moved Permanently Location: https://getyikes.com/ [following] --2019-07-23 14:48:36-- https://getyikes.com/ Connecting to getyikes.com (getyikes.com)|54.213.16.153|:443... connected. HTTP request sent, awaiting response... 200 OK

Exercise Field	Description
	`Length: 15759 (15K) [text/html]` `Saving to: ‘index.html.15’` `100%[====================================>] 15,759     --.-K/s   in 0.09s` `2019-07-23 14:48:37 (180 KB/s) - ‘index.html.15’ saved [15759/15759]`

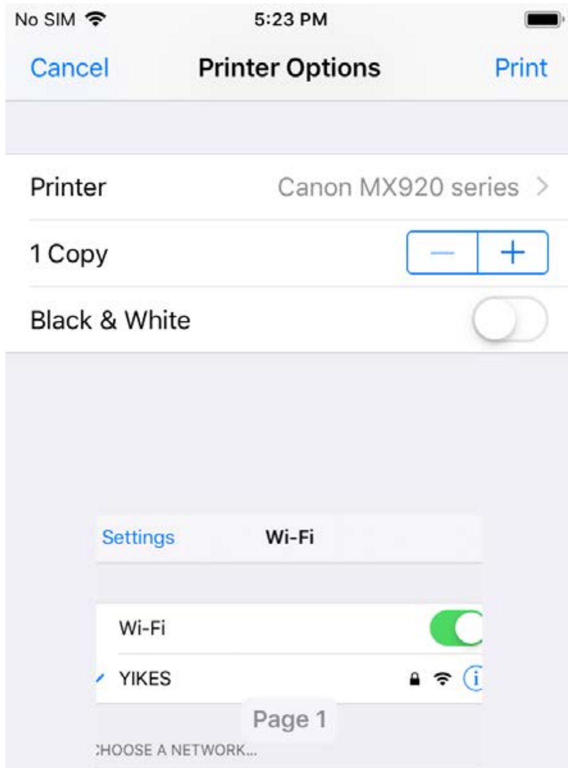
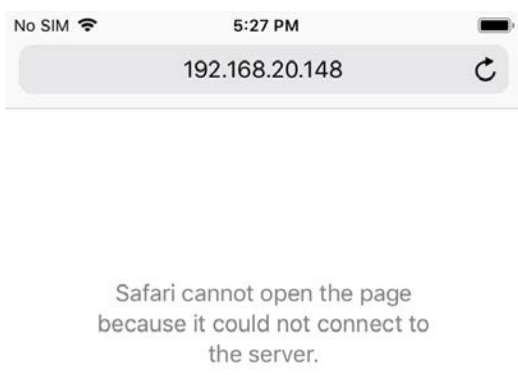
As explained above, exercise YnMUD-3-v6 is identical to exercise YnMUD-3-v4 except that it uses IPv6 instead of IPv4.

3.2.4.4 Exercise YnMUD-4-v4

Table 3-17: Exercise YnMUD-4-v4

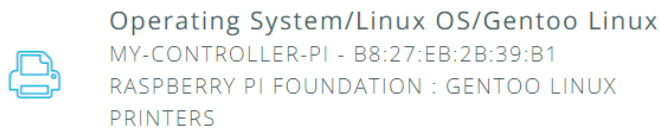
Exercise Field	Description
Parent Capability	(Y-4) Lateral (east-west) communications of the non-MUD-capable device to other devices on the local network are enforced according to the policy associated with the device’s category.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-4.a) A rule associated with the device’s category permits the device to initiate communications with local devices in category X, but there is no such rule that permits the device to initiate communications with local devices in category Y. (Y-4.a.1) The device will be allowed to initiate communications to any local device that is in category X. (Y-4.a.2) The device will be prohibited from initiating communications to any local device that is in category Y.
Description	Verify that once a device has been identified and categorized, the communications that it initiates to other devices on the local network will be restricted according to the local network (east-west) rules in place for the device’s category.
Associated Exercises	YnMUD-1-v4

Exercise Field	Description
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, ID.AM-4, PR.AC-1, PR.AC-3, PR.AC-4, PR.AC-5
IoT Device(s) Used	Same as for exercise YnMUD-1-v4
Policy Used	In the Yikes! UI: - The Cell Phone local rules are set to allow cell phones to initiate communications to printers but not to any other category of devices. - The Computer local rules are set to allow computers to initiate communications to all other devices. - The Printer local rules are set to deny printers from initiating communications to all other devices.
Preconditions	Same as for exercise YnMUD-1-v4. In addition, the device category rules are as described in the policy row above (the presence of these rules can be verified by accessing the Yikes! UI). Add several devices to the Printer and Laptop categories.
Procedure	1. Execute the procedures defined in exercise YnMUD-1-v4 and verify that the exercise has achieved the expected results (all IoT devices have had their make and model identified, if possible, and they have all been categorized correctly). 2. Verify that the cell phone can print a file successfully. 3. Verify that the cell phone cannot communicate with the connected appliance. 4. Recategorize a Raspberry Pi as a printer. 5. Verify that the Raspberry Pi cannot communicate with the laptop. 6. Verify that the laptop can send traffic to each of the other devices.
Demonstrated Results	When using the scanning software on the phone and laptop, only the devices that we expected to see in the procedural steps above could be seen. **Procedure 1: Completed; excluded for brevity**

Exercise Field	Description
	Procedure 2:  **Procedure 3:** 

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>

Procedure 4:



Procedure 5:

```
pi@my-controller-pi:~ $ wget 192.168.20.238

--2019-07-24 18:13:12--  http://192.168.20.238/

Connecting to 192.168.20.238:80... failed: Connection
refused.
```

Procedure 6:

Laptop to printer

```
[mud@localhost ~]$ wget 192.168.20.232
--2019-07-24 13:44:14--  http://192.168.20.232/
Connecting to 192.168.20.232:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 277
Saving to: 'index.html.17'

100%[======================================>] 277         --
.-K/s   in 0s

2019-07-24 13:44:14 (39.8 MB/s) - 'index.html.17' saved
[277/277]
```

Laptop to Pi categorized as printer

```
[mud@localhost ~]$ wget 192.168.20.117
--2019-07-24 14:03:29--  http://192.168.20.117/
Connecting to 192.168.20.117:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10701 (10K) [text/html]
Saving to: 'index.html.18'

100%[======================================>] 10,701      --
.-K/s   in 0.001s

2019-07-24 14:03:29 (8.95 MB/s) - 'index.html.18' saved
[10701/10701]
```

</td></tr>
</table>

As explained above, exercise YnMUD-4-v6 is identical to exercise YnMUD-4-v4 except that it uses IPv6 instead of IPv4.

3.2.4.5 Exercise YnMUD-5-v4

Table 3-18: Exercise YnMUD-5-v4

Exercise Field	Description
Parent Capability	(Y-5) In response to threat information, all devices on the local network are prohibited from visiting specific domains and IP addresses.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-5.a) Threat intelligence indicates a specific internet domain that should not be trusted. Devices are prohibited from initiating communications to the internet domain listed in the threat intelligence. In addition, they are prohibited from initiating communications to any other domains and IP addresses that are associated with the same threat campaign as this domain.
Description	Verify that when threat signaling information indicates that a specific domain is not safe, all devices on the local network will be restricted from initiating communications to that domain as well as to all other domains and IP addresses that are associated with the same threat campaign as this domain.
Associated Exercises	YnMUD-3-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.RA-2, ID.RA-3, PR.AC-3, PR.AC-4, PR.AC-5
IoT Device(s) Used	Use the same non-MUD-capable devices as for exercise YnMUD-3-v4: - laptop - Samsung Galaxy S8 cell phone - iPhone 7 cell phone
Policy Used	Use the same (non-MUD) Yikes! router policy as for exercise YnMUD-3-v4, specifically:

Exercise Field	Description
	In the Yikes! UI, the Computer internet rule is set to Allow All Internet Traffic rather than to IoT Specific Sites.
Preconditions	Threat signaling is enabled. Threat signaling intelligence indicates that internet domain *www.dangerousSite.org* is dangerous and devices shall be prohibited from visiting it. It also associates *www.dangerousSite1.org* with the same threat campaign as *www.dangerousSite.org*, and these domains are associated with IP addresses XX.XX.XX.XX and YY.YY.YY.YY. In addition, the other preconditions are the same as for exercise Yn-MUD-3-v4, specifically: The Computer category internet rule in the Yikes! UI is set to Allow All Internet Traffic rather than to IoT Specific Sites. Therefore, the ACL rules on the router are configured to permit the laptop to send traffic to any site.
Procedure	1. Log in to the router and verify that there is no ACL that prohibits visiting *www.dangerousSite.org*, *www.dangerousSite1.org*, or IP addresses XX.XX.XX.XX or YY.YY.YY.YY. 2. Run exercise YnMUD-3-v4 and verify that it has the expected results, i.e., verify that the laptop can browse to www.google.com, www.osmud.org, and www.getyikes.com. 3. At this point, the test has verified that the Yikes! router rules are being enforced as expected. Now test the threat signaling capability by using the laptop to try to browse to a site that is prohibited by the threat signaling information: *www.dangerousSite.org*. 4. Verify that the laptop is not permitted to connect to this site. 5. Verify that firewall rules corresponding to the threat response have been installed on the router, prohibiting communication with *www.dangerousSite.org*, *www.dangerousSite1.org*, and IP addresses XX.XX.XX.XX and YY.YY.YY.YY.
Demonstrated Results	With threat signaling enabled, the laptop is prohibited from initiating communications to domains flagged by threat signaling. **Procedure 1:** `config defaults`

Exercise Field	Description
	option syn_flood 1 option input ACCEPT option output ACCEPT option forward REJECT # Uncomment this line to disable ipv6 rules # option disable_ipv6 1 config zone option name lan list network 'lan' option input ACCEPT option output ACCEPT option log '1' config zone option name wan list network 'wan' list network 'wan6' option input REJECT option output ACCEPT option forward REJECT option masq 1 option mtu_fix 1 option log '1' config forwarding option src lan option dest wan # We need to accept udp packets on port 68, # see https://dev.openwrt.org/ticket/4108 config rule option name Allow-DHCP-Renew option src wan option proto udp option dest_port 68 option target ACCEPT option family ipv4 # Allow IPv4 ping config rule option name Allow-Ping option src wan option proto icmp option icmp_type echo-request option family ipv4 option target ACCEPT config rule option name Allow-IGMP option src wan option proto igmp

Exercise Field	Description
	option family ipv4 option target ACCEPT # Allow DHCPv6 replies # see https://dev.openwrt.org/ticket/10381 config rule option name Allow-DHCPv6 option src wan option proto udp option src_ip fc00::/6 option dest_ip fc00::/6 option dest_port 546 option family ipv6 option target ACCEPT config rule option name Allow-MLD option src wan option proto icmp option src_ip fe80::/10 list icmp_type '130/0' list icmp_type '131/0' list icmp_type '132/0' list icmp_type '143/0' option family ipv6 option target ACCEPT # Allow essential incoming IPv6 ICMP traffic config rule option name Allow-ICMPv6-Input option src wan option proto icmp list icmp_type echo-request list icmp_type echo-reply list icmp_type destination-unreachable list icmp_type packet-too-big list icmp_type time-exceeded list icmp_type bad-header list icmp_type unknown-header-type list icmp_type router-solicitation list icmp_type neighbour-solicitation list icmp_type router-advertisement list icmp_type neighbour-advertisement option limit 1000/sec option family ipv6 option target ACCEPT # Allow essential forwarded IPv6 ICMP traffic config rule option name Allow-ICMPv6-Forward option src wan option dest *

Exercise Field	Description
	`option proto icmp` `list icmp_type echo-request` `list icmp_type echo-reply` `list icmp_type destination-unreachable` `list icmp_type packet-too-big` `list icmp_type time-exceeded` `list icmp_type bad-header` `list icmp_type unknown-header-type` `option limit 1000/sec` `option family ipv6` `option target ACCEPT` `config rule` `option name Allow-IPSec-ESP` `option src wan` `option dest lan` `option proto esp` `option target ACCEPT` `config rule` `option name Allow-ISAKMP` `option src wan` `option dest lan` `option dest_port 500` `option proto udp` `option target ACCEPT` `# include a file with users custom iptables rules` `config include` `option path /etc/firewall.user` `### EXAMPLE CONFIG SECTIONS` **[Omitted for brevity]** `config rule` `        option enabled '1'` `        option target 'ACCEPT'` `        option src 'wan'` `        option proto 'tcp'` `        option dest_port '80'` `        option name 'AllowYikesAdminRemoteWeb'` `config rule` `        option enabled '1'` `        option target 'ACCEPT'` `        option src 'wan'` `        option proto 'tcp'` `        option dest_port '22'` `        option name 'AllowYikesAdminRemoteSsh'`

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
#
Base OpenWRT firewall rules to force the local router to
be the only DNS server allowed.
NOTE: This needs /etc/config/dhcp update to added the
router IP address as the primary DNS server
See dhcp.q9sample.conf for an example of this
configuration
#
config rule
        option target 'ACCEPT'
        option dest_port '53'
        option name 'Quad9 DNS Allow'
        option src 'lan'
        option dest_ip '9.9.9.9'
        option proto 'tcp udp'
        option dest 'wan'
        option family 'ipv4'

config rule
        option enabled '1'
        option src 'lan'
        option name 'DNS BLOCK OTHER SERVERS'
        option dest_port '53'
        option target 'REJECT'
        option proto 'tcp udp'
        option dest 'wan'

OSMUD start
#
DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-
FIGURATION
#
</pre>
[Omitted for brevity]
<pre>
OSMUD end
AYIKES start
#
DO NOT EDIT THESE LINES. AYIKES WILL REPLACE WITH ITS CON-
FIGURATION
#

Begin YIKES ipset firewall declarations
</pre>
[Omitted for brevity]
<hr>
Procedure 2:
--2019-07-24 10:50:53-- http://www.google.com/
</td></tr>
</table>

Exercise Field	Description
	Resolving www.google.com (www.google.com)... 172.217.164.132, 2607:f8b0:4004:815::2004 Connecting to www.google.com (www.google.com)|172.217.164.132|:80... connected. HTTP request sent, awaiting response... 200 OK Length: unspecified [text/html] Saving to: ‘index.html’ 0K .... 45.5M=0s 2019-07-24 10:50:53 (45.5 MB/s) - ‘index.html’ saved [11462] --2019-07-24 10:55:51-- https://osmud.org/ Resolving osmud.org (osmud.org)... 198.71.233.87 Connecting to osmud.org (osmud.org)|198.71.233.87|:443... connected. HTTP request sent, awaiting response... 200 OK Length: unspecified [text/html] Saving to: ‘index.html’ 0K .... 2.58M=0.009s 2019-07-24 10:55:51 (2.58 MB/s) - ‘index.html’ saved [24697]
	Procedures 3–4: $ ping *www.dangerousSite.org* ping: cannot resolve www.dangerousSite.org: Unknown host $ ping www.dangerousSite.org PING www.dangerousSite.org(127.0.0.1): 56 data bytes 64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.049 ms 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.073 ms 64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.082 ms 64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.139 ms 64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.079 ms 64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.072 ms 64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.123 ms 64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.073 ms ç64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.066 ms ^C --- www.dangerousSite.org ping statistics --- 9 packets transmitted, 9 packets received, 0.0% packet loss round-trip min/avg/max/stddev = 0.049/0.084/0.139/0.027 ms
	$ ping *www.dangerousSite1.org* ping: cannot resolve *www.dangerousSite1.org*: Unknown host

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>

```
$ ping www.dangerousSite1.org
PING www.dangerousSite1.org(127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.052 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.073 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.089 ms
^C
--- www.dangerousSite1.org ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.052/0.077/0.109/0.022 ms
```

Procedure 5:

```
# Q9THREATRULES start
#
# DO NOT EDIT THESE LINES. Q9THRT WILL REPLACE WITH ITS CON-
FIGURATION
#

config ipset
    option enabled 1
    option name Q9TS-joyheat_comFD
    option match dest_ip
    option storage hash
    option family ipv4
    option external Q9TS-joyheat_comFD

config ipset
    option enabled 1
    option name Q9TS-joyheat_comTD
    option match src_ip
    option storage hash
    option family ipv4
    option external Q9TS-joyheat_comTD

config rule
        option enabled   '1'
        option name      'Q9TS-joyheat_comFD'
        option target    REJECT
        option src       lan
        option dest      wan
        option proto     all
        option family    ipv4
        option ipset     Q9TS-joyheat_comFD
        option src_ip    any

config rule
        option enabled   '1'
        option name      'Q9TS-joyheat_comTD'
```

</td></tr>
</table>

Exercise Field	Description
	`option target REJECT` `option src wan` `option dest lan` `option proto all` `option family ipv4` `option ipset Q9TS-joyheat_comTD` `option dest_ip any` `# Q9THREATRULES end`

As explained above, exercise YnMUD-5-v6 is identical to exercise YnMUD-5-v4 except that it uses IPv6 instead of IPv4.

3.2.4.6 Exercise YnMUD-6-v4

Table 3-19: Exercise YnMUD-6-v4

Exercise Field	Description
Parent Capability	(Y-5) In response to threat information, all devices on the local network are prohibited from visiting specific domains and IP addresses.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-5.b) Threat intelligence indicates a specific IP address that should not be trusted. Devices are prohibited from initiating communications to the IP address listed in the threat intelligence. In addition, they are prohibited from initiating communications to any other IP addresses and domains that are associated with the same threat campaign as this IP address.
Description	Verify that when threat signaling information indicates that a specific IP address (as opposed to domain) is not safe, all devices on the local network will be restricted from initiating communications to that IP address as well as to all other IP addresses and domains that are associated with the same threat campaign as this IP address.
Associated Exercises	YnMUD-3-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.RA-2, ID.RA-3, PR.AC-3, PR.AC-4, PR.AC-5

Exercise Field	Description
IoT Device(s) Used	Use the same non-MUD-capable devices as for exercise YnMUD-3-v4: - laptop - Samsung Galaxy S8 cell phone - iPhone 7 cell phone
Policy Used	Use the same (non-MUD) Yikes! router policy as for exercise YnMUD-3-v4, specifically: In the Yikes! UI, the Computer internet rule is set to Allow All Internet Traffic rather than to IoT Specific Sites.
Preconditions	Threat signaling is enabled. Threat signaling intelligence indicates that IP address XX.XX.XX.XX is dangerous, and devices shall be prohibited from visiting it. It also associates IP address YY.YY.YY.YY with the same threat campaign as IP address XX.XX.XX.XX and these IP addresses are associated with domains *www.dangerousSite.org* and *www.dangerousSite1.org*. In addition, the other preconditions are the same as for exercise YnMUD-3-v4, specifically: The Computer category internet rule in the Yikes! UI is set to Allow All Internet Traffic rather than to IoT Specific Sites. Therefore, the firewall rules on the router are configured to permit the laptop to send traffic to any site.
Procedure	1. Log in to the router and verify that there is no ACL that prohibits visiting IP address XX.XX.XX.XX, IP address YY.YY.YY.YY, *www.dangerousSite.org*, or *www.dangerousSite1.org* (where IP address XX.XX.XX.XX is an address that is associated with the same threat as *www.dangerousSite.org*). 2. Run exercise YnMUD-3-v4 and verify that it has the expected results, i.e., verify that the laptop can browse to www.google.com, www.osmud.org, and www.trytechy.com. 3. At this point, the test has verified that the Yikes! router rules are being enforced as expected. 4. Run exercise YnMUD-5-v4. As a result, there should now be firewall rules on the router that prohibit all devices on the network from

Exercise Field	Description
	communicating with all domains and IP addresses that are associated with the same threat as the domain *www.dangerousSite.org*. 5. Use the laptop to try to browse to one of the IP addresses that is associated with the same threat as *www.dangerousSite.org*: IP address XX.XX.XX.XX. 6. Verify that the laptop is not permitted to connect to this site. 7. Verify that firewall rule corresponding to the threat response has been installed on the router, prohibiting communication with *www.dangerousSite.org*, *www.dangerousSite1.org*, and IP addresses XX.XX.XX.XX and YY.YY.YY.YY.
Demonstrated Results	With threat signaling enabled, the laptop is prohibited from initiating communications to IP addresses flagged by threat signaling intelligence. **Procedures 1–3:** **Completed; excluded for brevity** **Procedure 4:** Laptop ping *www.dangerousSite.org* `NCCoEs-MBP:results nccoe$ ping wwww.dangerousSite.org` `PING www.dangerousSite.org(127.0.0.1): 56 data bytes` `64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.039 ms` `64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.136 ms` `64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.063 ms` `64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.141 ms` `64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.071 ms` `^C` `--- www.dangerousSite.org ping statistics ---` `5 packets transmitted, 5 packets received, 0.0% packet loss` `round-trip min/avg/max/stddev = 0.039/0.090/0.141/0.041 ms` `NCCoEs-MBP:results nccoe$` `NCCoEs-MBP:results nccoe$ ping 192.60.252.130` `PING 192.60.252.130 (192.60.252.130): 56 data bytes` `Request timeout for icmp_seq 0` `Request timeout for icmp_seq 1` `Request timeout for icmp_seq 2` `Request timeout for icmp_seq 3` `^C` `--- 192.60.252.130 ping statistics ---` `5 packets transmitted, 0 packets received, 100.0% packet loss`

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td><pre>NCCoEs-MBP:results nccoe$</pre>
Procedure 5:
<pre># Q9THREATRULES start
#
DO NOT EDIT THESE LINES. Q9THRT WILL REPLACE WITH ITS CON-
FIGURATION
#

config ipset
    option enabled 1
    option name Q9TS-joyheat_comFD
    option match dest_ip
    option storage hash
    option family ipv4
    option external Q9TS-joyheat_comFD

config ipset
    option enabled 1
    option name Q9TS-joyheat_comTD
    option match src_ip
    option storage hash
    option family ipv4
    option external Q9TS-joyheat_comTD

config rule
        option enabled   '1'
        option name   'Q9TS-joyheat_comFD'
        option target   REJECT
        option src   lan
        option dest   wan
        option proto   all
        option family   ipv4
        option ipset   Q9TS-joyheat_comFD
        option src_ip   any

config rule
        option enabled   '1'
        option name   'Q9TS-joyheat_comTD'
        option target   REJECT
        option src   wan
        option dest   lan
        option proto   all
        option family   ipv4
        option ipset   Q9TS-joyheat_comTD
        option dest_ip   any
Q9THREATRULES end
OSMUD start</pre></td></tr>
</table>

501 As explained above, exercise YnMUD-6-v6 is identical to exercise YnMUD-6-v4 except that it uses IPv6
502 instead of IPv4.

503 #### 3.2.4.7 Exercise YnMUD-7-v4

504 **Table 3-20: Exercise YnMUD-7-v4**

Exercise Field	Description
Parent Capability	(Y-5) In response to threat information, all devices on the local network are prohibited from visiting specific domains and IP addresses.
Subrequirement(s) of Parent Capability to Be Demonstrated	(Y-5.c) Threat intelligence was received more than 24 hours prior, indicating domains and IP addresses that should not be trusted, and those domains and IP addresses were blocked by ACLs installed on the router. After 24 hours, these ACLs have been removed from the router.
Description	Verify that 24 or more hours after ACLs have been installed on the router as a result of threat signaling intelligence, those ACLs will be removed.
Associated Exercises	YnMUD-5-v4 and YnMUD-6-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.RA-2, ID.RA-3, PR.AC-3, PR.AC-4, PR.AC-5
IoT Device(s) Used	Same as for tests YnMUD-5-v4 and YnMUD-6-v4
Policy Used	Same as the policy used for tests YnMUD-3-v4, YnMUD-5-v4, and YnMUD-6-v4
Preconditions	Threat signaling is enabled. Threat signaling intelligence indicates that www.dangerousSite.org, www.dangerousSite1.org, and IP addresses XX.XX.XX.XX and YY.YY.YY.YY are dangerous, and devices shall be prohibited from visiting them.
Procedure	Run test YnMUD-5-v4 and verify that the laptop is not permitted to access www.dangerousSite.org, www.dangerousSite1.org, and IP addresses XX.XX.XX.XX and YY.YY.YY.YY. Log on to the router and verify that ACLs have been installed on it prohibiting communication with www.dangerousSite.org, www.dangerousSite1.org, and IP addresses XX.XX.XX.XX and YY.YY.YY.YY.

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>Let 24 hours elapse.
Log on to the router and verify that the ACLs that had prohibited communication with www.dangerousSite.org, www.dangerousSite1.org, and IP addresses XX.XX.XX.XX and YY.YY.YY.YY are no longer there.</td></tr>
<tr><td>Demonstrated Results</td><td>ACL rules that had been installed as a result of threat signaling intelligence were removed after 24 hours.
Procedure 1:
Completed; see YnMUD-6-v4
Procedure 2:
<pre>
Q9THREATRULES start
#
DO NOT EDIT THESE LINES. Q9THRT WILL REPLACE WITH ITS CON-
FIGURATION
#

config ipset
    option enabled 1
    option name Q9TS-joyheat_comFD
    option match dest_ip
    option storage hash
    option family ipv4
    option external Q9TS-joyheat_comFD

config ipset
    option enabled 1
    option name Q9TS-joyheat_comTD
    option match src_ip
    option storage hash
    option family ipv4
    option external Q9TS-joyheat_comTD

config rule
        option enabled   '1'
        option name   'Q9TS-joyheat_comFD'
        option target   REJECT
        option src   lan
        option dest   wan
        option proto   all
        option family   ipv4
        option ipset   Q9TS-joyheat_comFD
        option src_ip   any

config rule
        option enabled   '1'
        option name   'Q9TS-joyheat_comTD'
        option target   REJECT
</pre></td></tr>
</table>

Exercise Field	Description
	`` option src wan option dest lan option proto all option family ipv4 option ipset Q9TS-joyheat_comTD option dest_ip any # Q9THREATRULES end # OSMUD start ``` **Procedure 4:** ``` root@OpenWrt:~# cat /etc/config/firewall config defaults option syn_flood 1 option input ACCEPT option output ACCEPT option forward REJECT # Uncomment this line to disable ipv6 rules # option disable_ipv6 1 config zone option name lan list network 'lan' option input ACCEPT option output ACCEPT option log '1' config zone option name wan list network 'wan' list network 'wan6' option input REJECT option output ACCEPT option forward REJECT option masq 1 option mtu_fix 1 option log '1' config forwarding option src lan option dest wan # We need to accept udp packets on port 68, # see https://dev.openwrt.org/ticket/4108 config rule option name Allow-DHCP-Renew option src wan option proto udp option dest_port 68 option target ACCEPT option family ipv4 ```

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
Allow IPv4 ping
config rule
        option name   Allow-Ping
        option src   wan
        option proto   icmp
        option icmp_type   echo-request
        option family   ipv4
        option target   ACCEPT

config rule
        option name   Allow-IGMP
        option src   wan
        option proto   igmp
        option family   ipv4
        option target   ACCEPT
</pre>
[Omitted for brevity]
<pre>
Q9THREATRULES start
#
DO NOT EDIT THESE LINES. Q9THRT WILL REPLACE WITH ITS CON-
FIGURATION
#
Q9THREATRULES end
OSMUD start
#
DO NOT EDIT THESE LINES. OSMUD WILL REPLACE WITH ITS CON-
FIGURATION
#
</pre>
[Omitted for brevity]
<pre>
OSMUD end
AYIKES start
#
DO NOT EDIT THESE LINES. AYIKES WILL REPLACE WITH ITS CON-
FIGURATION
#

Begin YIKES ipset firewall declarations
</pre>
[Omitted for brevity]
<pre>
AYIKES end
</pre>
</td></tr>
</table>

As explained above, exercise YnMUD-7-v6 is identical to exercise YnMUD-7-v4 except that it uses IPv6 instead of IPv4.

4 Build 3

Build 3 uses equipment and cloud resources from CableLabs. The CableLabs Micronets Gateway on the local network; a cloud-based micro-services layer that hosts various Micronets services (e.g., software-defined networking [SDN] controller, Micronets Manager, MUD manager, configuration micro-service, identity server [optional], and DHCP/DNS configuration services) and a mobile application are used to perform IoT device onboarding via the Wi-Fi Easy Connect protocol and to manage and enforce trust domains on the local network, as well as support MUD. (Note that another name for the Wi-Fi Easy Connect protocol is Device Provisioning Protocol [DPP]. Throughout the remainder of this document, we use the term DPP for conciseness.)

4.1 Evaluation of MUD-Related Capabilities

The functional evaluation that was conducted to verify that Build 3 conforms to the MUD specification was based on the Build-3-specific requirements listed in Table 4-1.

4.1.1 Requirements

Table 4-1: MUD Use Case Functional Requirements

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-1	The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file).			IoT-1-v4, IoT-11-v4
CR-1.a		The device’s MUD file is located by using two items in the device’s		IoT-1-v4, IoT-11-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		bootstrapping information (which is encoded in its QR code): the information element and the public bootstrapping key.		
CR-1.a.1			The information element identifies a device vendor, and each vendor is assumed to have a well-known location for serving MUD files, so this element identifies the location of the device's MUD file server. The public bootstrapping key of the device identifies the device's MUD file.	IoT-1-v4, IoT-11-v4
CR-2	The IoT DDoS example implementation shall include the capability for the MUD URL to be provided to a MUD manager.			IoT-1-v4
CR-2.a		The device bootstrapping information shall be sent to the DPP configurator as part of the device DPP onboarding request.		IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-2.a.1			The bootstrapping information (and, in particular, the information element and public bootstrapping key) are **received at the DPP configurator.**	IoT-1-v4
CR-2.b		**The DPP configurator** shall use the bootstrapping information to **look up the MUD URL and send it to the MUD manager.**		IoT-1-v4
CR-2.b.1			**The MUD manager shall receive the MUD URL.**	IoT-1-v4
CR-3	The IoT DDoS example implementation shall include a **MUD manager that can request a MUD file and signature from a MUD file server.**			IoT-1-v4
CR-3.a		The MUD manager shall use the GET method (RFC 7231) to **request MUD and signature files** (per RFC 7230) from the MUD file server and can **validate the MUD file**		IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		server's TLS certificate by using the rules in RFC 2818.		
CR-3.a.1			**The MUD file server shall receive the https request from the MUD manager.**	IoT-1-v4
CR-3.b		**The MUD manager** shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it **cannot validate the MUD file server's TLS certificate** by using the rules in RFC 2818.		IoT-2-v4
CR-3.b.1			**The MUD manager shall drop the connection** to the MUD file server.	IoT-2-v4
CR-3.b.2			**The MUD manager shall send locally defined policy to the gateway** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-2-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-4	The IoT DDoS example implementation shall include a **MUD file server that can serve a MUD file and signature to the MUD manager.**			IoT-1-v4
CR-4.a		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the **certificate had not expired.**		IoT-1-v4
CR-4.b		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** was valid at the time of signing, i.e., the **certificate had already expired when it was**		IoT-3-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		used to sign the MUD file.		
CR-4.b.1			The MUD manager will not complete processing the MUD file. (The MUD file rules will not be applied.)	IoT-3-v4
CR-4.b.2			The MUD manager shall apply locally defined policy to the **gateway** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-3-v4
CR-5	The IoT DDoS example implementation shall include a **MUD manager** that **can translate local network configurations based on the MUD file.**			IoT-1-v4
CR-5.a		**The MUD manager shall successfully validate the signature of the MUD file.**		IoT-1-v4
CR-5.a.1			The MUD manager, after validation of the MUD file signature,	IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			shall **check for an existing MUD file and translate abstractions in the MUD file to gateway configurations.**	
CR-5.a.2			The MUD manager shall **cache** this newly received MUD file.	IoT-10-v4
CR-5.b		The MUD manager shall attempt to validate the signature of the **MUD file,** but the **signature validation fails** (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason).		IoT-4-v4
CR-5.b.1			**The MUD manager shall cease processing the MUD file.**	IoT-4-v4
CR-5.b.2			**The MUD manager shall send locally defined policy to the gateway** that handles whether to allow or block traffic to and	IoT-4-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			from the MUD-enabled IoT device.	
CR-6	The IoT DDoS example implementation shall include a **MUD manager that can configure the Micronets Gateway with ACLs that enforce the MUD file rules.**			IoT-1-v4
CR-6.a		**The MUD manager shall install ACLs** on the Micronets Gateway.		IoT-1-v4
CR-6.a.1			**The gateway shall have been configured to enforce the route filter sent by the MUD manager.**	IoT-1-v4
CR-7	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file.**			IoT-5-v4
CR-7.a		The MUD-enabled IoT device shall attempt to **initiate outbound traffic to approved internet services.**		IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-7.a.1			The gateway shall receive the attempt and shall **allow the traffic to pass** based on the filters from the MUD file.	IoT-5-v4
CR-7.b		An approved **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4
CR-7.b.1			The gateway shall receive the attempt and shall **allow it to pass** based on the filters from the MUD file.	IoT-5-v4
CR-8	The IoT DDoS example implementation shall **deny communications from a MUD-enabled IoT device to unapproved internet services** (i.e., services that are denied by virtue of not being explicitly approved).			IoT-5-v4
CR-8.a		The MUD-enabled IoT device shall **attempt to initiate outbound traffic to unapproved** (implicitly denied) **internet services.**		IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-8.a.1			**The gateway shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4
CR-8.b		**An unapproved** (implicitly denied) **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4
CR-8.b.1			**The gateway shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4
CR-8.c		The MUD-enabled IoT device shall initiate communications to an internet service that is **approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device.**		IoT-5-v4
CR-8.c.1			**The gateway shall receive the attempt**	IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			and shall deny it based on the filters from the MUD file.	
CR-8.d		An internet service shall initiate communications to a MUD-enabled device that is **approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service.**		IoT-5-v4
CR-8.d.1			**The gateway shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4
CR-9	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate laterally with devices that are approved** in the MUD file.			IoT-6-v4
CR-9.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to approved devices.**		IoT-6-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-9.a.1			**The gateway shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4
CR-9.b		An approved device **shall attempt to initiate a lateral connection to the MUD-enabled IoT device.**		IoT-6-v4
CR-9.b.1			**The gateway shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4
CR-10	The IoT DDoS example implementation shall **deny lateral communications from a MUD-enabled IoT device to devices that are not approved** in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved). (Note that this assumes that when devices are onboarded, they are placed in separate micronets from other local devices with which they are not permitted to communicate.			IoT-6-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	In practice, it means that for testing purposes, each device must be assigned to its own separate micronet.)			
CR-10.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to unapproved** (implicitly denied) **devices.**		IoT-6-v4
CR-10.a.1			**The gateway shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4
CR-10.b		**An unapproved** (implicitly denied) **device shall attempt to initiate a lateral connection** to the MUD-enabled IoT device.		IoT-6-v4
CR-10.b.1			**The gateway shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4
CR-11	If the IoT DDoS example implementation is designed such that its DHCP server			No test needed because the DHCP

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	does not act as a MUD manager and it forwards a MUD URL to a MUD manager, **the DHCP server must notify the MUD manager of any corresponding change to the DHCP state** of the MUD-enabled IoT device, and the MUD manager should **remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.**			server does not forward the MUD URL to the MUD manager.
CR-11.a		The MUD-enabled IoT **device shall explicitly release the IP address lease** (i.e., it sends a DHCP release message to the DHCP server).		N/A
CR-11.a.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has been released.**	N/A
CR-11.a.2			**The MUD manager should remove all policies** associated with the disconnected IoT device	N/A

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			that had been configured on the MUD PEP router/switch.	
CR-11.b		The MUD-enabled IoT **device's IP address lease shall expire.**		N/A
CR-11.b.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has expired.**	N/A
CR-11.b.2			**The MUD manager should remove all policies** associated with the affected IoT device that had been configured on the MUD PEP router/switch.	N/A
CR-12	The IoT DDoS example implementation shall include a **MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed** for the MUD file indicated by the MUD URL**. The MUD manager should fetch a new**			IoT-10-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	MUD file if the cache-validity time period has already elapsed.			
CR-12.a		The MUD manager shall check if the file associated with the **MUD URL is present in its cache** and shall determine that it is.		IoT-10-v4
CR-12.a.1			The MUD manager shall **check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager shall apply the contents of the cached MUD file.	IoT-10-v4
CR-12.a.2			The MUD manager **shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity**	IoT-10-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			value for this MUD file. If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.	
CR-13	The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the gateway will be configured with **all possible instantiations of that rule,** insofar as **each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the gateway.**			IoT-9-v4
CR-13.a		The MUD file for a device shall contain a rule involving a **domain that can resolve to multiple IP addresses** when queried by the gateway. **Flow rules for permitting access to each of those IP addresses will be inserted into the gateway** for the device in question,		IoT-9-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		and the device will be permitted to communicate with all of those IP addresses.		
CR-13.a.1			IPv4 addressing is used on the network.	IoT-9-v4

4.1.2 Test Cases

This section contains the test cases that were used to verify that Build 3 met the requirements listed in Table 4-1.

4.1.2.1 Test Case IoT-1-v4

Table 4-2: Test Case IoT-1-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL). (CR-2) The IoT DDoS example implementation shall include the capability for the MUD URL to be provided to a MUD manager. (CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server. (CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager. (CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file. (CR-6) The IoT DDoS example implementation shall include a MUD manager that can configure the Micronets Gateway with ACLs that enforce the MUD file rules.

Test Case Field	Description
Testable Requirements	(CR-1.a) The device's MUD file is located by using two items in the device's bootstrapping information (which is encoded in its QR code): the information element and the public bootstrapping key. (CR-1.a.1) The information element identifies a device vendor, and each vendor is assumed to have a well-known location for serving MUD files, so this element identifies the location of the device's MUD file server. The public bootstrapping key of the device identifies the device's MUD file. (CR-2.a) The device bootstrapping information shall be sent to the DPP configurator as part of the device DPP onboarding request. (CR-2.a.1) The bootstrapping information (and in particular the information element and public bootstrapping key) are received at the DPP configurator. (CR-2.b) The DPP configurator shall use the bootstrapping information to look up the MUD URL and send it to the MUD manager. (CR-2.b.1) The MUD manager shall receive the MUD URL. (CR-3.a) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server and can validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.a.1) The MUD file server shall receive the https request from the MUD manager. (CR-4.a) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the certificate had not expired. (CR-5.a) The MUD manager shall successfully validate the signature of the MUD file. (CR-5.a.1) The MUD manager, after validation of the MUD file signature, shall check for an existing MUD file and translate abstractions in the MUD file to gateway configurations. (CR-6.a) The MUD manager shall install ACLs on the Micronets Gateway. (CR-6.a.1) The gateway shall have been configured to enforce the route filter sent by the MUD manager.

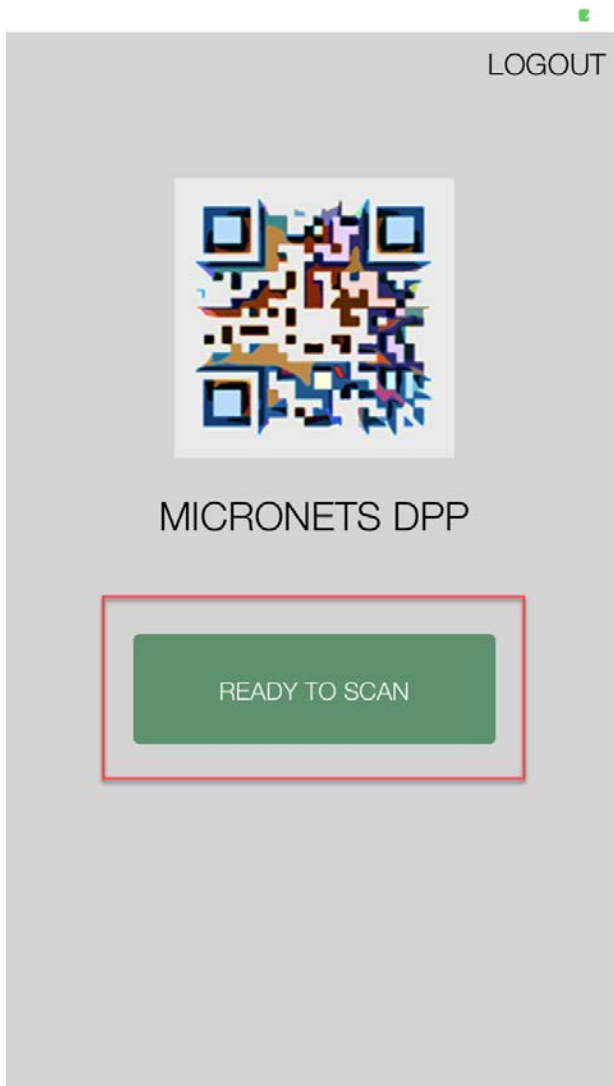
Test Case Field	Description
Description	Shows that when a device that has a MUD file is onboarded to the network using DPP and that device's bootstrapping information includes an information element value to indicate the location of the device's manufacturer and a public bootstrapping key to indicate the device's MUD file, the device will have its gateway automatically configured to enforce the route filtering that is described in the device's MUD file, assuming the MUD file has a valid signature and is served from a MUD file server that has a valid TLS certificate.
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.PT-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_northsouth.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The device's MUD file has a valid signature that was signed by a certificate that had not yet expired, and it is being hosted on a MUD file server that has a valid TLS certificate. 4. The gateway does not yet have any configuration settings pertaining to the IoT device being used in the test. 5. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 4.1.3. 6. The mobile phone onboarding application is installed and logged into the subscriber account that is associated with the gateway.
Procedure	Verify that the gateway for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. 1. Power on the IoT device.

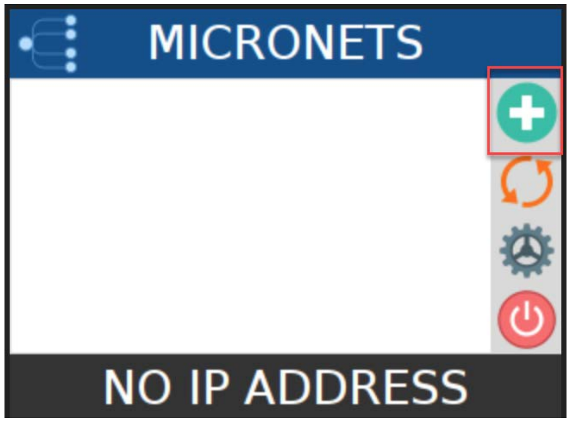
Test Case Field	Description
	2. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages on the frequency indicated by the QR code. 3. Open the onboarding application on the mobile phone and click READY TO SCAN. 4. Position the mobile phone's camera to read the device's QR code. Do this in a timely manner because there is a 60-second countdown for the device to exit DPP onboarding mode. 5. Input additional device-specific information into the mobile onboarding application as requested (must be done within the same 60-second time limit): a. Assign the device to its own unique micronets class (e.g., Generic) to which no other device is or will be assigned. b. Give the device a unique name (e.g., Device 1). c. Click the ONBOARD button on the mobile application. This causes the onboarding application to send the device's bootstrapping information to the DPP configurator on the gateway via the operator's multiple-system operator (MSO) portal and cloud infrastructure. 6. Wait. The following operations are being performed automatically in the operator's cloud infrastructure: a. The Micronets Manager receives the bootstrapping information. b. It looks up the URL of the device's MUD file. c. It provides the MUD file URL to the MUD manager. d. The MUD manager contacts the MUD file server and verifies that it has a valid TLS certificate. e. The MUD manager requests the MUD file and the MUD signature file and validates the MUD file.

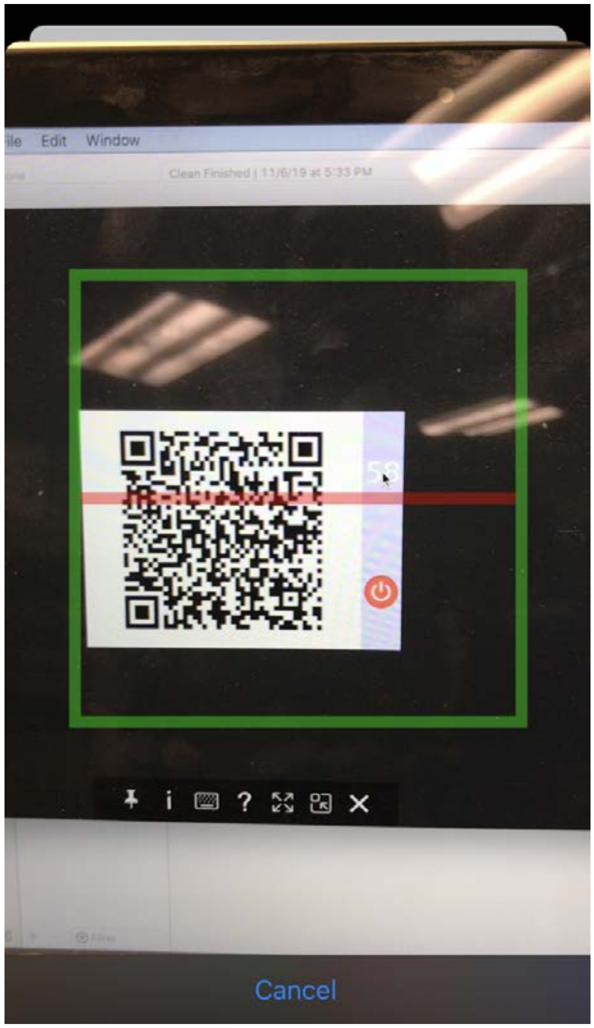
Test Case Field	Description
	f. The MUD manager parses the MUD rules and translates these to ACLs (route filtering rules) that it sends to the Micronets Manager. g. The Micronets Manager provisions the device on the Micronets Gateway and installs MUD ACLs for the device so that the gateway is now configured to enforce the policies specified in the MUD file. h. The gateway briefly switches to the device's frequency and initiates DPP authentication. i. The device switches to the gateway's frequency and receives its network credentials via DPP. j. The device connects to the network. 7. View the logs on the gateway to verify that: a. The bootstrapping information was received at the configurator. b. The authentication phase of DPP onboarding occurred for the device. This is a three-way handshake among the device and the gateway. c. The configuration phase of DPP onboarding occurred for the device (another three-way handshake). 8. Verify that the ACLs that reflect the MUD file rules have been installed on the gateway.
Expected Results	The gateway has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. ACLs are installed on the gateway to reflect MUD filtering rules.
Actual Results	**Onboarding:** **Step 1–sign in to application:**

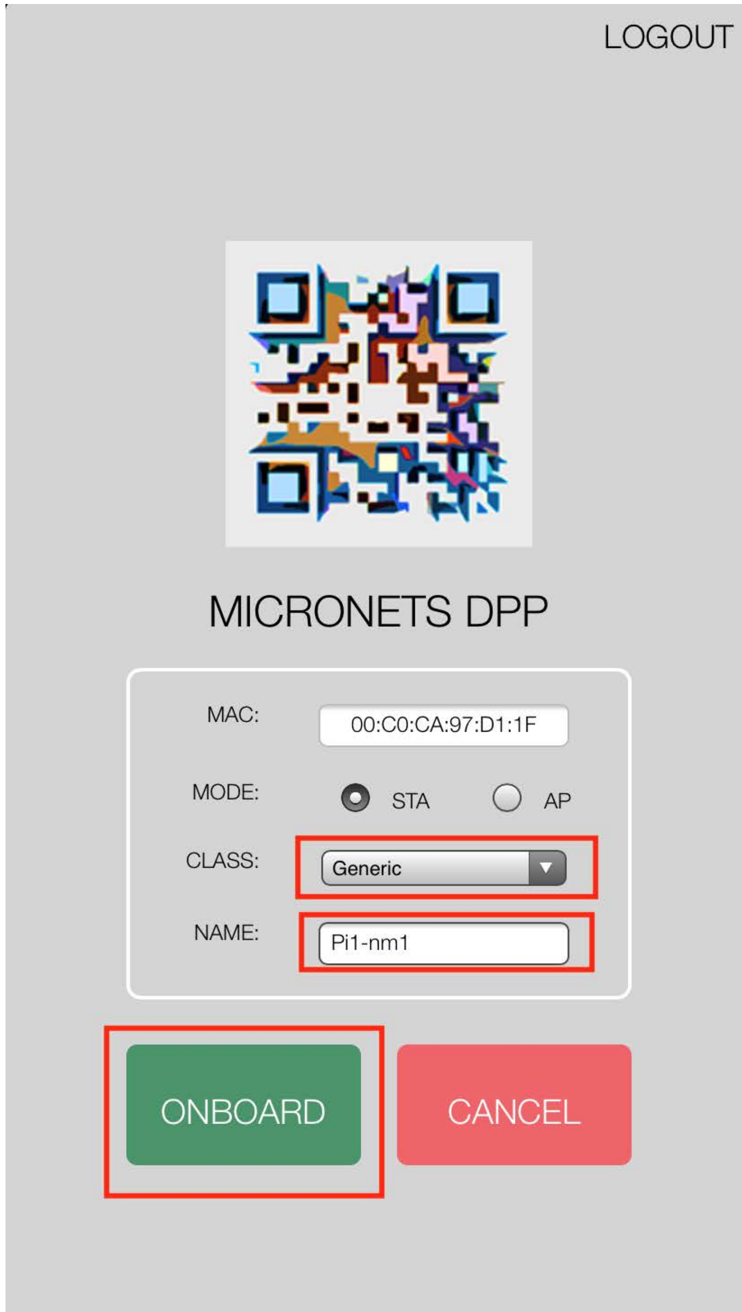
<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>

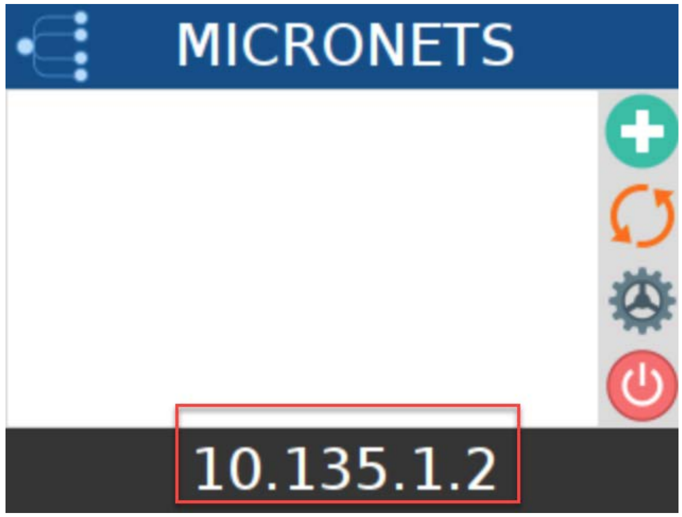
<u>Step 2–click READY TO SCAN on mobile application</u>:</td></tr>
</table>

Test Case Field	Description
	 **<u>Step 3—click plus button on IoT device UI</u>:**

Test Case Field	Description
	 **<u>Step 4–QR code appears on IoT device UI</u>:**  **<u>Step 5–scan QR code from mobile application</u>:**

Test Case Field	Description
	 **Step 6–input device information and click ONBOARD:**

Test Case Field	Description
	 **Step 7–device receives IP address:**

Test Case Field	Description
	 **<u>Verify appropriate micronet created</u>:** `{` `"_id": "5ee7bf78ab3e8358c185e759",` `"id": "subscriber-001",` `"name": "Subscriber 001",` `"ssid": "micronets-gw",` `"gatewayId": "micronets-gw",` `"micronets": [` `{` `"name": "Generic",` `"class": "Generic",` `"micronet-subnet-id": "`**`Generic`**`",` `"trunk-gateway-port": "2",` `"trunk-gateway-ip": "10.36.32.124",` `"dhcp-server-port": "LOCAL",` `"dhcp-zone": "10.135.1.0/24",` `"ovs-bridge-name": "brmn001",` `"ovs-manager-ip": "10.36.32.124",` `"micronet-subnet": "10.135.1.0/24",` `"micronet-gateway-ip": "10.135.1.1",` `"connected-devices": [` `{` `"device-mac": "`**`00:C0:CA:97:D1:1F`**`",` `"device-name": "`**`Pi1-nm1`**`",`

Test Case Field	Description
	"device-id": "**463165abc19725aefffc39def13ce09b17167fba**", "device-openflow-port": "2", "device-ip": "10.135.1.2" }], "micronet-id": "2316794860" }], "createdAt": "2020-06-15T18:35:36.968Z", "updatedAt": "2020-06-16T18:04:06.636Z", "__v": 0 } **<u>View flow rules</u>:** Every 2.0s: sudo ovs-ofctl dump-flows brmn001 --names | /opt/micronets-gw/bin/format-ofctl-dump Tue Jun 16 15:23:00 2020 table=0 priority=500 n_packets=0 dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0 actions=drop table=0 priority=500 n_packets=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop table=0 priority=500 n_packets=0 icmp icmp_code=1 actions=drop table=0 priority=450 n_packets=643 in_port=LOCAL actions=resubmit(200) table=0 priority=400 n_packets=1218 in_port="wlp2s0.2486" actions=resubmit(100) table=0 priority=400 n_packets=18 in_port=wlp2s0 actions=resubmit(100) table=0 priority=0 n_packets=2 actions=output:diagout1 table=100 priority=910 n_packets=0 ct_state=+rel+trk udp actions=LOCAL table=100 priority=910 n_packets=1 ct_state=+est+trk udp actions=LOCAL table=100 priority=910 n_packets=490 ct_state=-trk udp actions=ct(table=100) table=100 priority=905 n_packets=0 ct_state=+est+trk tcp actions=LOCAL table=100 priority=905 n_packets=0 ct_state=+rel+trk tcp actions=LOCAL

Test Case Field	Description
	`table=100 priority=905 n_packets=0 ct_state=-trk tcp actions=ct(table=100)` `table=100 priority=900 n_packets=18 dl_type=0x888e actions=resubmit( 120)` `table=100 priority=850 n_packets=137 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.1.1 actions=resubmit( 120)` `table=100 priority=815 n_packets=0 in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f dl_type=0x888e actions=resubmit( 120)` `table=100 priority=815 n_packets=0 udp in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f tp_dst=67 actions=resubmit( 120)` `table=100 priority=815 n_packets=352 arp in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f actions=resubmit( 120)` `table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.1.1 actions=resubmit( 120)` `table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=104.237.132.42 actions=resubmit( 120)` `table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=198.71.233.87 actions=resubmit( 120)` `table=100 priority=805 n_packets=103 in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f actions=output:diagout1` `table=100 priority=800 n_packets=0 in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f actions=resubmit( 110)` `table=100 priority=460 n_packets=0 in_port=wlp2s0 dl_type=0x888e actions=resubmit( 120)` `table=100 priority=0 n_packets=0 actions=output:diagout1` **[Omitted for length]** **Micronets Gateway and Micronets Manager logs verifying onboarding:** 1. DPP Onboarding Initiated: • Micronets Gateway: “DPPHandler.onboard_device: Issuing DPP onboarding commands for device”

Test Case Field	Description
	2020-06-16 14:03:32,897 micronets-gw-service: INFO DPPHandler.onboard_device: Issuing DPP onboarding commands for device '463165abc19725aefffc39def13ce09b17167fba' in micronet 'generic... 2020-06-16 14:03:32,898 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending: 2020-06-16 14:03:32,899 micronets-gw-service: INFO { "DPPOnboardingStartedEvent": { "deviceId": "463165abc19725aefffc39def13ce09b17167fba", "macAddress": "00:C0:CA:97:D1:1F", "micronetId": "Generic", "reason": "DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")" } } • Micronets Manager: "DPPOnboardingStartedEvent" 2020-06-16T18:03:32.923407831Z Gateway Message : {"body":{"DPPOnboardingStartedEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"}}} EventType : "DPPOnboardingStartedEvent" 2020-06-16T18:03:32.923417691Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:32.923424251Z Event to Post : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"}

Test Case Field	Description
	2020-06-16T18:03:32.923432861Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:32.923483580Z OnBoarding PatchBody : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingStartedEvent","deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"}} 2. DPP Authorization Success: • Micronets Gateway: “DPP-AUTH-SUCCESS” 2020-06-16 14:03:32,921 micronets-gw-service: INFO DPPHandler.handle_hostapd_cli_event(DPP-AUTH-SUCCESS init=1) 2020-06-16 14:03:32,921 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending: 2020-06-16 14:03:32,921 micronets-gw-service: INFO { "DPPOnboardingProgressEvent": { "deviceId": "463165abc19725aefffc39def13ce09b17167fba", "macAddress": "00:C0:CA:97:D1:1F", "micronetId": "Generic", "reason": "DPP Progress (DPP-AUTH-SUCCESS init=1)" } } • Micronets Manager: “DPPOnboardingProgressEvent”/“DPP Progress (DPP-AUTH-SUCCESS init=1)” 2020-06-16T18:03:32.954959234Z Gateway Message : {"body":{"DPPOnboardingProgressEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}}} EventType : "DPPOnboardingProgressEvent" 2020-06-16T18:03:32.955713205Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]:

Test Case Field	Description
	`2020-06-16T18:03:32.955759765Z  Event to Post : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}` `2020-06-16T18:03:32.957158978Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:32.957181208Z OnBoarding PatchBody : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingProgressEvent","deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}}` 3. DPP Configuration Sent: • Micronets Gateway: "DPP-CONF-SENT" `2020-06-16 14:03:33,338 micronets-gw-service: INFO DPPHandler.handle_hostapd_cli_event(DPP-CONF-SENT)` `2020-06-16 14:03:33,338 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending:` `2020-06-16 14:03:33,338 micronets-gw-service: INFO {` ` "DPPOnboardingProgressEvent": {` ` "deviceId": "463165abc19725aefffc39def13ce09b17167fba",` ` "macAddress": "00:C0:CA:97:D1:1F",` ` "micronetId": "Generic",` ` "reason": "DPP Progress (DPP-CONF-SENT)"` ` }` `}` • Micronets Manager: "DPPOnboardingProgressEvent"/"DPP Progress (DPP-CONF-SENT init=1)" `2020-06-16T18:03:33.363367674Z Gateway Message : {"body":{"DPPOnboardingProgressEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}}} EventType : "DPPOnboardingProgressEvent"`

Test Case Field	Description
	`2020-06-16T18:03:33.363573045Z 2020-06-16 18:03:33 ESC[34mdebugESC[39m [index.js]:` `2020-06-16T18:03:33.363584045Z  Event to Post : {"de-viceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","mi-cronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}` `2020-06-16T18:03:33.363785005Z 2020-06-16 18:03:33 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:33.363794825Z OnBoarding PatchBody : {"de-viceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingProgressEv-ent","de-viceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","mi-cronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}}` 4. DPP Onboarding Completed: • Micronets Gateway: "AP-STA-CONNECTED" `2020-06-16 14:03:36,851 micronets-gw-service: INFO DPPHandler.handle_hostapd_cli_event(AP-STA-CONNECTED 00:c0:ca:97:d1:1f)`  `2020-06-16 14:03:36,851 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending:` `2020-06-16 14:03:36,851 micronets-gw-service: INFO {` ` "DPPOnboardingCompleteEvent": {` ` "deviceId": "463165abc19725aefffc39def13ce09b17167fba",` ` "macAddress": "00:C0:CA:97:D1:1F",` ` "micronetId": "Generic",` ` "reason": "DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"` ` }` `}` • Micronets Manager: "DPPOnboardingCompleteEvent"/"DPP Onboarding Complete (AP-STA-CONNECTED" `2020-06-16T18:03:36.882393990Z Gateway Message : {"body":{"DPPOnboardingCompleteEvent":{"de-viceId":"463165abc19725aefffc39def13ce09b17167fba`

Test Case Field	Description
	","macAddress":"00:C0:CA:97:D1:1F","mi- cronetId":"Generic","reason":"DPP Onboarding Com- plete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"}}} EventType : "DPPOnboardingCompleteEvent" 2020-06-16T18:03:36.882403959Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:36.882409589Z Event to Post : {"de- viceId":"463165abc19725aefffc39def13ce09b17167fba ","macAddress":"00:C0:CA:97:D1:1F","mi- cronetId":"Generic","reason":"DPP Onboarding Com- plete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"} 2020-06-16T18:03:36.882415439Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:36.882466150Z OnBoarding PatchBody : {"de- viceId":"463165abc19725aefffc39def13ce09b17167fba ","events":{"type":"DPPOnboardingCom- pleteEvent","de- viceId":"463165abc19725aefffc39def13ce09b17167fba ","macAddress":"00:C0:CA:97:D1:1F","mi- cronetId":"Generic","reason":"DPP Onboarding Com- plete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"}} 2020-06-16T18:03:36.882475160Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]: 2020-06-16T18:03:36.882479660Z Hook Type: before Path: mm/v1/dpp Method: patch 2020-06-16T18:03:36.882486270Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:36.882490280Z 2020-06-16T18:03:36.882493840Z PATCH BEFORE HOOK DPP DATA : {"de- viceId":"463165abc19725aefffc39def13ce09b17167fba ","events":{"type":"DPPOnboardingCom- pleteEvent","de- viceId":"463165abc19725aefffc39def13ce09b17167fba ","macAddress":"00:C0:CA:97:D1:1F","mi- cronetId":"Generic","reason":"DPP Onboarding Com- plete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"}} PARAMS : {} RequestUrl : undefined 2020-06-16T18:03:36.882500760Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]: 2020-06-16T18:03:36.882505420Z Hook Type: before Path: mm/v1/dpp Method: get 2020-06-16T18:03:36.883566612Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]: 2020-06-16T18:03:36.883590111Z Hook Type: after Path: mm/v1/dpp Method: get

Test Case Field	Description
	`2020-06-16T18:03:36.883834742Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]: Hook.result.data : undefined 2020-06-16T18:03:36.884259803Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]:2020-06-16T18:03:36.884279723Z`
Overall Results	Pass

526 IPv6 is not supported in this implementation.

527 *4.1.2.2 Test Case IoT-2-v4*

528 **Table 4-3: Test Case IoT-2-v4**

Test Case Field	Description
Parent Requirement	(CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server.
Testable Requirement	(CR-3.b) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it cannot validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.b.1) The MUD manager shall drop the connection to the MUD file server. (CR-3.b.2) The MUD manager shall send locally defined policy to the gateway that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD manager cannot validate the TLS certificate of a MUD file server when trying to retrieve the MUD file for a specific IoT device, the MUD manager will drop the connection to the MUD file server and configure the gateway according to locally defined policy regarding whether to allow or block traffic to the IoT device in question.
Associated Test Case(s)	IoT-11-v4

Test Case Field	Description
Associated Cybersecurity Framework Subcategory(ies)	PR.AC-7
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_northsouth.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The MUD file server that is hosting the MUD file of the device under test does not have a valid TLS certificate. 4. Local policy has been defined to ensure that if the MUD file for a device is located on a server with an invalid certificate, the gateway will be configured to provision the device and permit it unrestricted communications as if it had not been associated with a MUD file. 5. The gateway for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test. 6. The mobile phone onboarding application is installed and logged into the subscriber account that is associated with the gateway.
Procedure	Verify that the gateway for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. 1. Power on the IoT device. 2. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages. 3. Open the onboarding application on the mobile phone and click READY TO SCAN.

Test Case Field	Description
	4. Position the mobile phone's camera to read the device's QR code. Do this in a timely manner because there is a 60-second countdown for the device to exit DPP onboarding mode. 5. Input additional device-specific information into the mobile onboarding application as requested (must be done within the same 60-second time limit): a. Assign the device to its own unique micronets class (e.g., Security) to which no other device is or will be assigned. b. Give the device a unique name (e.g., Device 1). c. Click the ONBOARD button on the mobile application. This causes the onboarding application to send the device's bootstrapping information to the DPP configurator on the gateway via the operator's MSO portal and cloud infrastructure. 6. Wait. The following operations are being performed automatically in the operator's cloud infrastructure: a. The Micronet's Manager receives the bootstrapping information. b. It looks up the URL of the device's MUD file. c. It provides the MUD file URL to the MUD manager. d. The MUD manager contacts the MUD file server, determines that it does not have a valid TLS certificate, and drops the connection to the MUD file server. e. The Micronets Manager provisions the device on the gateway as if the device had not been associated with a MUD file. In other words, the device does not have any MUD-related restrictions imposed on its communications. (Note that it is a local policy decision as to whether the implementation will fail "closed" and restrict all communications or fail "open" [as this implementation does] and not impose any communications

Test Case Field	Description
	restrictions. In theory, the implementation could assign the device to a more restricted micronet.)
Expected Results	The gateway has had its configuration changed, i.e., it has been configured to permit the device to connect to the network and communicate without any MUD-based restrictions.
Actual Results	`2020-02-20 14:54:42,699 micronets-mud-manager: INFO get-MudInfo called with: {'url': 'https://nccoe-mud-server.micronets.in/micronets-mud/nist-model-fe_samemanufacturer-to.json'}` `2020-02-20 14:54:42,700 micronets-mud-manager: INFO getMUD-File: url: https://nccoe-mud-server.micronets.in/micronets-mud/nist-model-fe_samemanufacturer-to.json` `2020-02-20 14:54:42,703 micronets-mud-manager: INFO getMUD-File: mud filepath for https://nccoe-mud-server.micronets.in/micronets-mud/nist-model-fe_samemanufacturer-to.json: /mud-cache-dir/nccoe-mud-server.micronets.in_micronets-mud_nist-model-fe_samemanufacturer-to.json...` `2020-02-20 14:54:42,705 micronets-mud-manager: INFO getMUD-File: RETRIEVING https://nccoe-mud-server.micronets.in/micronets-mud/nist-model-fe_samemanufacturer-to.json` `[2020-02-20 14:54:42,760]` **`ERROR in app: Exception on request POST /getMudInfo`** **`ssl.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:852)`**
Overall Results	Pass

529 IPv6 is not supported in this implementation.

530 4.1.2.3 Test Case IoT-3-v4

531 **Table 4-4: Test Case IoT-3-v4**

Test Case Field	Description
Parent Requirement	(CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager.

Test Case Field	Description
Testable Requirement	(CR-4.b) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file was valid at the time of signing. It shall determine that the certificate had already expired when it was used to sign the MUD file. (CR-4.b.1) The MUD manager shall cease to process the MUD file. (CR-4.b.2) The MUD manager shall send locally defined policy to the gateway that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD file server serves a MUD file with a signature that was created with an expired certificate, the MUD manager will cease processing the MUD file.
Associated Test Case(s)	IoT-11-v4
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_expiredcert.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The IoT device's MUD file is being hosted on a MUD file server that has a valid TLS certificate, but the MUD file signature was signed by a certificate that had already expired at the time of signature. 4. Local policy has been defined to ensure that if the MUD file for a device has a signature that was signed by a certificate that had already expired at the time of signature, the gateway will provision the device and permit it unrestricted communications as if it had not been associated with a MUD file.

Test Case Field	Description
	5. The gateway does not yet have any configuration settings with respect to the IoT device being used in the test. 6. The mobile phone onboarding application is installed and logged into the subscriber account that is associated with the gateway.
Procedure	Verify that the gateway does not yet have any configuration settings installed with respect to the IoT device being used in the test. Verify that the gateway for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. 1. Power on the IoT device. 2. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages. 3. Open the onboarding application on the mobile phone and click READY TO SCAN. 4. Position the mobile phone's camera to read the device's QR code. Do this in a timely manner because there is a 60-second countdown for the device to exit DPP onboarding mode. 5. Input additional device-specific information into the mobile onboarding application as requested (must be done within the same 60-second time limit): a. Assign the device to its own unique micronets class (e.g., Shared) to which no other device is or will be assigned. b. Give the device a unique name (e.g., Device 1). c. Click the ONBOARD button on the mobile application. This causes the onboarding application to send the de-

Test Case Field	Description
	vice's bootstrapping information to the DPP configurator on the gateway via the operator's MSO portal and cloud infrastructure. 6. Wait. The following operations are being performed automatically in the operator's cloud infrastructure: a. The Micronets Manager receives the bootstrapping information. b. It looks up the URL of the device's MUD file. c. It provides the MUD file URL to the MUD manager. d. The MUD manager contacts the MUD file server, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. e. The MUD file server serves the MUD file and signature to the MUD manager, and the MUD manager detects that the MUD file's signature was created by using a certificate that had already expired at the time of signing. f. The Micronets Manager provisions the device on the gateway as if the device had not been associated with a MUD file. In other words, the device does not have any MUD-related restrictions imposed on its communications. (Note that it is a local policy decision as to whether the implementation will fail "closed" and restrict all communications or fail "open" [as this implementation does] and not impose any communications restrictions. In theory, the implementation could assign the device to a more restricted micronet.)
Expected Results	The gateway has had its configuration changed, i.e., it has been configured to permit the device to connect to the network and communicate without any MUD-based restrictions.
Actual Results	Onboarding occurs as executed in Test Case IoT-1-v4.

Test Case Field	Description
	MUD manager logs: 2020-06-01T19:21:35.145932392Z [2020-06-01 19:21:35,145] 172.17.0.1:57652 POST /getMudInfo 1.0 500 62 4622 2020-06-01T19:21:35.151372716Z 2020-06-01 19:21:35,145 quart.serving: INFO 172.17.0.1:57652 POST /getMudInfo 1.0 500 62 4622 2020-06-01T19:27:14.779094064Z 2020-06-01 19:27:14,778 micronets-mud-manager: INFO getMudInfo called with: {'url': **'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.json'**} 2020-06-01T19:27:14.779344473Z 2020-06-01 19:27:14,779 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.json 2020-06-01T19:27:14.779669434Z 2020-06-01 19:27:14,779 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_expiredcert.json... 2020-06-01T19:27:14.779893264Z 2020-06-01 19:27:14,779 micronets-mud-manager: INFO getMUDFile: RETRIEVING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.json 2020-06-01T19:27:14.812317780Z 2020-06-01 19:27:14,811 micronets-mud-manager: DEBUG Saved MUD https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.json to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_expiredcert.json 2020-06-01T19:27:14.812567930Z 2020-06-01 19:27:14,812 micronets-mud-manager: INFO Attempting to retrieve MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.p7s 2020-06-01T19:27:14.819022355Z 2020-06-01 19:27:14,818 micronets-mud-manager: INFO Successfully retrieved MUD signature https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.p7s 2020-06-01T19:27:14.819639326Z 2020-06-01 19:27:14,819 micronets-mud-manager: INFO Saved MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.p7s to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_expiredcert.p7s 2020-06-01T19:27:14.827058362Z 2020-06-01 19:27:14,826 micronets-mud-manager: DEBUG Signature validation command returned status 4 (Verification failure) 2020-06-01T19:27:14.827369362Z 2020-06-01 19:27:14,827 micronets-mud-manager: INFO MUD signature validation FAILURE (MUD file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_expiredcert.json, sig file /mud-

Test Case Field	Description
	cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_expiredcert.p7s) 2020-06-01T19:27:14.827576822Z 2020-06-01 19:27:14,827 micronets-mud-manager: INFO Signature failure details: 2020-06-01T19:27:14.827595112Z 140195888018560:error:2E099064:CMS routines:cms_signerinfo_verify_cert:certificate verify error:../crypto/cms/cms_smime.c:253:Verify error:**certificate has expired** 2020-06-01T19:27:14.827599552Z 2020-06-01T19:27:14.830093744Z 2020-06-01 19:27:14,829 micronets-mud-manager: INFO Returning status 400 for POST request for /getMudInfo: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.json failed signature validation (via https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_expiredcert.p7s): **Verification failure** 2020-06-01T19:27:14.839997072Z [2020-06-01 19:27:14,839] 172.17.0.1:57716 POST /getMudInfo 1.0 400 248 61267 2020-06-01T19:27:14.840225902Z 2020-06-01 19:27:14,839 quart.serving: INFO 172.17.0.1:57716 POST /getMudInfo 1.0 400 248 61267
Overall Results	Pass

532 IPv6 is not supported in this implementation.

533 *4.1.2.4 Test Case IoT-4-v4*

534 **Table 4-5: Test Case IoT-4-v4**

Test Case Field	Description
Parent Requirement	(CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file.
Testable Requirement	(CR-5.b) The MUD manager shall attempt to validate the signature of the MUD file, but the signature validation fails (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason). (CR-5.b.1) The MUD manager shall cease processing the MUD file. (CR-5.b.2) The MUD manager shall send locally defined policy to the gateway that handles whether to allow or block traffic to and from the MUD-enabled IoT device.

Test Case Field	Description
Description	Shows that if the MUD manager determines that the signature on the MUD file it receives from the MUD file server is invalid, it will cease processing the MUD file and configure the gateway according to locally defined policy regarding whether to allow or block traffic to the IoT device in question.
Associated Test Case(s)	IoT-11-v4
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_invalidsig.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The MUD file that is served from the MUD file server to the MUD manager has a signature that is invalid, even though it was signed by a certificate that had not expired at the time of signing. 4. Local policy has been defined to ensure that if the MUD file for a device has an invalid signature, the gateway will be configured to provision the device and permit it unrestricted communications as if it had not been associated with a MUD file. 5. The gateway does not yet have any configuration settings with respect to the IoT device being used in the test. 6. The mobile phone onboarding application is installed and logged into the subscriber account that is associated with the gateway.
Procedure	Verify that the gateway does not yet have any configuration settings installed with respect to the IoT device being used in the test.

Test Case Field	Description
	Verify that the gateway for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. 1. Power on the IoT device. 2. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages. 3. Open the onboarding application on the mobile phone and click READY TO SCAN. 4. Position the mobile phone's camera to read the device's QR code. Do this in a timely manner because there is a 60-second countdown for the device to exit DPP onboarding mode. 5. Input additional device-specific information into the mobile onboarding application as requested (must be done within the same 60-second time limit): a. Assign the device to its own unique micronets class (e.g., Generic) to which no other device is or will be assigned. b. Give the device a unique name (e.g., Device 1). c. Click the ONBOARD button on the mobile application. This causes the onboarding application to send the device's bootstrapping information to the DPP configurator on the gateway via the operator's MSO portal and cloud infrastructure. 6. Wait. The following operations are being performed automatically in the operator's cloud infrastructure: a. The Micronets Manager receives the bootstrapping information. b. It looks up the URL of the device's MUD file.

Test Case Field	Description
	c. It provides the MUD file URL to the MUD manager. d. The MUD manager contacts the MUD file server, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. e. The MUD file server serves the MUD file and signature file to the MUD manager, and the MUD manager detects that the MUD file's signature is invalid. f. The Micronets Manager provisions the device on the gateway as if the device had not been associated with a MUD file. In other words, the device does not have any MUD-related restrictions imposed on its communications. (Note that it is a local policy decision as to whether the implementation will fail "closed" and restrict all communications or fail "open" [as this implementation does] and not impose any communications restrictions. In theory, the implementation could assign the device to a more restricted micronet.)
Expected Results	The gateway has had its configuration changed, i.e., it has been configured to permit the device to connect to the network and communicate without any MUD-based restrictions.
Actual Results	Onboarding occurs as executed in Test Case IoT-1-v4. **<u>MUD manager logs:</u>** `2020-06-01T19:39:06.642029549Z 2020-06-01 19:39:06,641 micronets-mud-manager: INFO getMudInfo called with: {'url':` **`'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.json`**`'}` `2020-06-01T19:39:06.642269829Z 2020-06-01 19:39:06,642 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.json` `2020-06-01T19:39:06.642629430Z 2020-06-01 19:39:06,642 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-`

Test Case Field	Description
	model-fe_invalidsig.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_invalidsig.json... 2020-06-01T19:39:06.642873149Z 2020-06-01 19:39:06,642 micronets-mud-manager: INFO getMUDFile: RETRIEVING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.json 2020-06-01T19:39:06.649721996Z 2020-06-01 19:39:06,649 micronets-mud-manager: DEBUG Saved MUD https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.json to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_invalidsig.json 2020-06-01T19:39:06.649979886Z 2020-06-01 19:39:06,649 micronets-mud-manager: INFO Attempting to retrieve MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.p7s 2020-06-01T19:39:06.655804960Z 2020-06-01 19:39:06,655 micronets-mud-manager: INFO Successfully retrieved MUD signature https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.p7s 2020-06-01T19:39:06.656470161Z 2020-06-01 19:39:06,656 micronets-mud-manager: INFO Saved MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.p7s to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_invalidsig.p7s 2020-06-01T19:39:06.663617138Z 2020-06-01 19:39:06,663 micronets-mud-manager: DEBUG Signature validation command returned status 4 (Verification failure) 2020-06-01T19:39:06.663920888Z 2020-06-01 19:39:06,663 micronets-mud-manager: INFO MUD signature validation FAILURE (MUD file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_invalidsig.json, sig file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_invalidsig.p7s) 2020-06-01T19:39:06.664095668Z 2020-06-01 19:39:06,663 micronets-mud-manager: INFO Signature failure details: 2020-06-01T19:39:06.664105068Z 139636532962432:error:2E09A09E:CMS routines:CMS_SignerInfo_verify_content:verification failure:../crypto/cms/cms_sd.c:848: 2020-06-01T19:39:06.664108968Z 139636532962432:error:2E09D06D:CMS routines:CMS_verify:content verify error:../crypto/cms/cms_smime.c:393: 2020-06-01T19:39:06.664112498Z 2020-06-01T19:39:06.664799219Z 2020-06-01 19:39:06,664 micronets-mud-manager: INFO Returning status 400 for POST request for /getMudInfo: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.json failed signature validation (via https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_invalidsig.p7s): **Verification failure** 2020-06-01T19:39:06.674001717Z [2020-06-01 19:39:06,673] 172.17.0.1:57802 POST /getMudInfo 1.0 400 246 32530

Test Case Field	Description
	`2020-06-01T19:39:06.674199247Z 2020-06-01 19:39:06,673 quart.serving: INFO 172.17.0.1:57802 POST /getMudInfo 1.0 400 246 32530`
Overall Results	Pass

535 IPv6 is not supported in this implementation.

536 *4.1.2.5 Test Case IoT-5-v4*

537 **Table 4-6: Test Case IoT-5-v4**

Test Case Field	Description
Parent Requirement	(CR-7) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file. (CR-8) The IoT DDoS example implementation shall deny communications from a MUD-enabled IoT device to unapproved internet services (i.e., services that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-7.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to approved internet services. (CR-7.a.1) The gateway shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-7.b) An approved internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-7.b.1) The gateway shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-8.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to unapproved (implicitly denied) internet services. (CR-8.a.1) The gateway shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.b) An unapproved (implicitly denied) internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-8.b.1) The gateway shall receive the attempt and shall deny it based on the filters from the MUD file.

Test Case Field	Description
	(CR-8.c) The MUD-enabled IoT device shall initiate communications to an internet service that is approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device. (CR-8.c.1) The gateway shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.d) An internet service shall initiate communications to a MUD-enabled device that is approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service. (CR-8.d.1) The gateway shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has a gateway that is configured to enforce the route filtering that is described in the device's MUD file with respect to communication with internet services. Further, it shows that the policies that are configured on the gateway with respect to communication with internet services will be enforced as expected, with communications that are configured as denied being blocked and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.IP-1, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_northsouth.json*
Preconditions	Test IoT-1-v4 has run successfully, meaning that the gateway has been configured to enforce the following policies for the IoT device in question (as defined in the MUD file in Section 4.1.3): Note: Preconditions with strike-through are not applicable due to NAT.

Test Case Field	Description
	a) ~~Explicitly permit *https://yes-permit-from.com* to initiate communications with the IoT device.~~ b) Explicitly permit the IoT device to initiate communications with *https://yes-permit-to.com*. c) Implicitly deny all other communications with the internet, including denying: i. ~~the IoT device to initiate communications with *https://yes-permit-from.com*~~ ii. *https://yes-permit-to.com* to initiate communications with the IoT device iii. communication between the IoT device and all other internet locations, such as *https://unnamed-to.com* (by not mentioning this or any other URLs in the MUD file)
Procedure	Note: Procedure steps with strike-through were not tested due to NAT. As stipulated in the preconditions, right before this test, test IoT-1-v4 must have been run successfully. 1. Initiate communications from the IoT device to *https://yes-permit-to.com* and verify that this traffic is received at *https://yes-permit-to.com* (egress). 2. ~~Initiate communications to the IoT device from *https://yes-permit-to.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device (ingress).~~ 3. ~~Initiate communications to the IoT device from *https://yes-permit-from.com* and verify that this traffic is received at the IoT device (ingress).~~ 4. ~~Initiate communications from the IoT device to *https://yes-permit-from.com* and verify that this traffic is received at the gateway, but it is not forwarded by the gateway, nor is it received at *https://yes-permit-from.com* (ingress).~~ 5. Initiate communications from the IoT device to *https://unnamed.com* and verify that this traffic is received at the gateway, but it is not forwarded by the gateway, nor is it received at *https://unnamed.com* (egress). 6. Initiate communications to the IoT device from *https://unnamed.com* and verify that this traffic is received at the MUD PEP,

Test Case Field	Description
	but it is not forwarded by the MUD PEP, nor is it received at the IoT device (ingress).
Expected Results	Each of the results that is listed as needing to be verified in procedure steps above occurs as expected.
Actual Results	**Flow rules:** `Every 2.0s: sudo ovs-ofctl dump-flows brmn001 --names | /opt/micronets-gw/bin/format-ofctl-dump` `Tue Jun  2 11:17:06 2020` `table=0   priority=500 n_packets=0 dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0 actions=drop` `table=0   priority=500 n_packets=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop` `table=0   priority=500 n_packets=0      icmp icmp_code=1 actions=drop` `table=0   priority=450 n_packets=7      in_port=LOCAL actions=resubmit( 200)` `table=0   priority=400 n_packets=2      in_port=wlp2s0 actions=resubmit( 100)` `table=0   priority=400 n_packets=33 in_port="wlp2s0.1861" actions=resubmit( 100)` `table=0   priority=0   n_packets=0      actions=output:diagout1` `table=100 priority=910 n_packets=0      ct_state=+est+trk udp actions=LOCAL` `table=100 priority=910 n_packets=0      ct_state=+rel+trk udp actions=LOCAL` `table=100 priority=910 n_packets=9      ct_state=-trk udp actions=ct(table=100)` `table=100 priority=905 n_packets=0      ct_state=+est+trk tcp actions=LOCAL` `table=100 priority=905 n_packets=0      ct_state=+rel+trk tcp actions=LOCAL` `table=100 priority=905 n_packets=0      ct_state=-trk tcp actions=ct(table=100)` `table=100 priority=900 n_packets=2      dl_type=0x888e actions=resubmit( 120)`

Test Case Field	Description
	table=100 priority=850 n_packets=1 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=10 arp in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 actions=re-submit(120) table=100 priority=815 n_packets=2 udp in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 tp_dst=67 ac-tions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=52.89.85.207 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=54.191.221.118 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=54.201.49.86 actions=resubmit(120) table=100 priority=805 n_packets=20 in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 actions=out-put:diagout1 table=100 priority=800 n_packets=0 in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 actions=re-submit(110) table=100 priority=460 n_packets=0 in_port=wlp2s0 dl_type=0x888e actions=resubmit(120) table=100 priority=0 n_packets=0 actions=output:di-agout1
	Procedure 2: pi@raspberrypi:~ $ wget https://www.cablelabs.com --2020-06-02 09:19:56-- https://www.cablelabs.com/ Resolving www.cablelabs.com (www.cablelabs.com)... 52.89.85.207, 54.201.49.86, 54.191.221.118, ... Connecting to www.cablelabs.com (www.cable-labs.com)|52.89.85.207|:443... connected.
	Procedure 6: pi@raspberrypi:~ $ wget https://www.facebook.com

Test Case Field	Description
	`--2020-06-02 09:55:06--  https://www.facebook.com/` `Resolving www.facebook.com (www.facebook.com)... 31.13.66.35, 2a03:2880:f103:83:face:b00c:0:25de` `Connecting to www.facebook.com (www.facebook.com)|31.13.66.35|:443... failed: Connection timed out.` `Connecting to www.facebook.com (www.facebook.com)|2a03:2880:f103:83:face:b00c:0:25de|:443... failed: Network is unreachable.` **Procedure 7:** `$ ssh pi@10.135.1.2` `ssh: connect to host 10.135.1.2 port 22: Operation timed out`
Overall Results	Pass

IPv6 is not supported in this implementation.

4.1.2.6 Test Case IoT-6-v4

Table 4-7: Test Case IoT-6-v4

Test Case Field	Description
Parent Requirement	(CR-9) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate laterally with devices that are approved in the MUD file. (CR-10) The IoT DDoS example implementation shall deny lateral communications from a MUD-enabled IoT device to devices that are not approved in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-9.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to approved devices. (CR-9.a.1) The gateway shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-9.b) An approved device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-9.b.1) The gateway shall receive the attempt and shall allow it to pass based on the filters from the MUD file.

Test Case Field	Description
	(CR-10.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to unapproved (implicitly denied) devices. (CR-10.a.1) The gateway shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-10.b) An unapproved (implicitly denied) device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-10.b.1) The gateway shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its gateway automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with lateral devices. Further, it shows that the policies that are configured on the gateway with respect to communication with lateral devices will be enforced as expected, with communications that are configured as denied being blocked and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.AC-5, PR.IP-1, PR.PT-3, PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_controller_anyport.json, nist-model-fe_localnetwork_anyport.json, nist-model-fe_manufacturer1.json, nist-model-fe_manufacturer2.json, nist-model-fe_manufacturer-from.json, nist-model-fe_manufacturer-to.json, nist-model-fe_mycontroller.json, nist-model-fe_samemanufacturer.json, nist-model-fe_samemanufacturer-from.json, nist-model-fe_samemanufacturer-to.json*
Preconditions	a) Test IoT-1-v4 has run successfully numerous times to onboard local devices *(anyhost-to*, *anyhost-from*, *unnamed-host*, a device of a specific manufacturer class, and a device of the same manu-

Test Case Field	Description
	facturer class) needed to test enforcement of local communications. These devices have all been onboarded to separate micronets. As a result, the gateway has been configured to enforce the following policies for each IoT device in question with respect to local communications (as defined in the MUD files in Section 4.1.3). (Please note that the cases below that have strike-throughs are untestable for the following reasons: First, Micronets does not yet support port-level flow rules. Second, NAT prevents certain communication attempts, making particular test cases untestable. Third, for devices to be considered on the local network, they must be on the same micronet. Communication within the same micronet will always be allowed and cannot be constrained by MUD rules. b) ~~Local-network class—Explicitly permit **local communication to and from the IoT device and any local hosts** (including the specific local hosts *anyhost-to* and *anyhost-from*) **for specific services,** as specified in the MUD file by source port: any; destination port: 80; and protocol: TCP, and which party initiates the connection.~~ c) Manufacturer class—Explicitly permit **local communication to and from the IoT device and other classes of IoT devices, as identified by their MUD URL *(www.devicetype.com),*** and further constrained by ~~source port: any; destination port: 80; and protocol: TCP.~~ d) Same-manufacturer class—Explicitly permit **local communication to and from IoT devices of the same manufacturer as the IoT device in question (the domain in the MUD URLs [mudfileserver] of the other IoT devices is the same as the domain in the MUD URL [mudfileserver] of the IoT device in question),** and further constrained by ~~source port: any; destination port: 80; and protocol: TCP~~. e) Implicitly deny all other local communication that is not explicitly permitted in the MUD file, including denying i. ***anyhost-to* to initiate communications** with the IoT device ii. ~~the **IoT device to initiate communications** with *anyhost-to* by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**~~

Test Case Field	Description
	iii. the **IoT device to initiate communications with *any-host-from*** iv. ~~***anyhost-from* to initiate communications** with the IoT device by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**~~ v. communications between the IoT device and all lateral hosts (including *unnamed-host)* whose **MUD URLs are not explicitly mentioned** as being permissible in the MUD file vi. ~~communications between the IoT device and all lateral hosts whose **MUD URLs are explicitly mentioned** as being permissible **but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**~~ vii. communications between the IoT device and all lateral hosts that are **not from the same manufacturer** as the IoT device in question viii. ~~communications between the IoT device and a lateral host that **is from the same manufacturer but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**~~
Procedure	Note: Procedure steps with strike-through were not tested in this phase because ingress DACLs are not supported in this implementation. As stipulated in the preconditions, right before this test, test IoT-1-v4 must have been run successfully to onboard the other local devices. Note that when each device is onboarded, the user performing the onboarding must assign each device to its own separate micronet. Local-network (ingress): Initiate communications to the IoT device from *anyhost-from* **for specific permitted service,** and verify that this traffic is received at the IoT device. 1. ~~Local-network (egress): **Initiate communications from the IoT device to *anyhost-from*** for specific permitted service, and verify that this traffic is received at the gateway, but it **is not forwarded** by the gateway, nor is it received at *anyhost-from*.~~ 2. Local-network, controller, my-controller, manufacturer class (egress): Initiate communications from the IoT device to *anyhost-to*

Test Case Field	Description
	for specific permitted service, and verify that this traffic **is received** at *anyhost-to.* 3. ~~Local network~~, controller, my-controller, manufacturer class (ingress): **Initiate communications to the IoT device from *anyhost-to*** for specific permitted service, and verify that this traffic is received at the gateway, but it **is not forwarded** by the gateway, nor is it received at the IoT device. 4. No associated class (egress): Initiate communications from the IoT device to *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted),** and verify that this traffic is received at the gateway, but it **is not forwarded** by the gateway, nor is it received at *unnamed-host*. (Reminder: For this to work, each device must have been manually assigned to its own separate micronet during the onboarding process.) 5. No associated class (ingress): Initiate communications to the IoT device from *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted),** and verify that this traffic is received at the gateway, but it is not forwarded by the gateway, nor is it received at the IoT device. 6. Same-manufacturer class (egress): Initiate communications from the IoT device to *same-manufacturer-host* (where *same-manufacturer-host* is **a host that is from the same manufacturer as the IoT device** in question), and verify that this traffic **is received** at *same-manufacturer-host*. 7. Same-manufacturer class (egress): Initiate communications from the IoT device to *same-manufacturer-host* (where *same-manufacturer-host* is **a host that is from the same manufacturer as the IoT device** in question) **but using a port or protocol that is not specified,** and verify that this traffic is received at the gateway, but it **is not forwarded** by the gateway, nor is it received at *same-manufacturer-host*.
Expected Results	Each of the results that is listed as needing to be verified in the procedure steps above occurs as expected.

Test Case Field	Description
Actual Results	The numbering in this section correlates with the procedure steps above: 2. Local-network (ingress)—allowed: `pi@pi-2:~ $ ssh pi@10.135.2.3` `pi@10.135.2.3's password:` `Last login: Tue Jun  2 10:33:45 2020 from 192.168.30.181` `pi@pi-1:~ $`
	4. Local-network, controller, my-controller, manufacturer class (egress)—allowed: Local-network: `pi@pi-1:~ $ ssh pi@10.135.2.2` `pi@10.135.2.2's password:` `Last login: Tue Jun  2 14:23:16 2020 from 192.168.30.181` `pi@pi-2:~ $` Controller: `pi@pi-2:~ $ wget nccoe-server1.micronets.net` `--2020-06-08 08:47:21--  http://nccoe-server1.micronets.net/` `Resolving nccoe-server1.micronets.net (nccoe-server1.micronets.net)... 104.237.132.42` `Connecting to nccoe-server1.micronets.net (nccoe-server1.micronets.net)|104.237.132.42|:80...` **`connected`**`.` My-controller: `pi@pi-2:~ $ wget nccoe-server1.micronets.net` `--2020-06-08 09:19:49--  http://nccoe-server1.micronets.net/` `Resolving nccoe-server1.micronets.net (nccoe-server1.micronets.net)... 104.237.132.42` `Connecting to nccoe-server1.micronets.net (nccoe-server1.micronets.net)|104.237.132.42|:80...` **`connected`**`.` Manufacturer: `pi@pi-1:~ $ ssh pi@10.135.3.2` `pi@10.135.3.2's password:`

Test Case Field	Description
	`Last login: Thu Jun  4 10:31:17 2020 from 192.168.30.181` `pi@pi-2:~ $`
	5. ~~Local network~~, ~~controller~~, ~~my-controller~~, manufacturer class (in-gress)—blocked: Manufacturer: `pi@pi-1:~ $ ssh pi@10.135.3.2` `ssh: connect to host 10.135.3.2 port 22: Connection timed out`
	6. No associated class (egress)—blocked: `Pi-3 to Pi-2:` `pi@pi-3:~ $ ssh pi@10.135.2.2` `ssh: connect to host 10.135.2.2 port 22: Connection timed out`
	7. No associated class (ingress)—blocked: `Pi-2 to Pi-3:` `pi@pi-2:~ $ ssh pi@10.135.3.2` `ssh: connect to host 10.135.3.2 port 22: Connection timed out`
	8. Same-manufacturer class (egress)—allowed: `Pi-2 to Pi-1:` `pi@pi-2:~ $ ssh pi@10.135.2.2` `pi@10.135.2.2's password:` `Last login: Thu Jun  4 09:56:21 2020 from 192.168.30.181` `pi@pi-1:~ $`
	9. Same-manufacturer class (egress)—blocked: `Pi-1 to Pi-2:` `pi@pi-1:~ $ ssh pi@10.135.3.2` `ssh: connect to host 10.135.3.2 port 22: Connection timed out`
Overall Results	Partial Pass. The gateway was configured to enforce all route filtering that is described in the device's MUD file with respect to communication

Test Case Field	Description
	with lateral devices, with the exception of MUD rules that pertain to specific ports. At the time of this functional demonstration, Micronets did not yet support port-level flow rules. Therefore, the implementation we tested was not able to enforce any port-specific route filtering that is described in the device's MUD file with respect to communication with lateral devices. If a MUD file rule permitted the device to communicate with a lateral host using only a specific port or ports, the Micronets implementation was observed to incorrectly permit the device to communicate to all ports of that permitted host, even though that communication should have been restricted to using only the specific port or ports specified in the MUD file.

541 IPv6 is not supported in this implementation.

542 *4.1.2.7 Test Case IoT-9-v4*

543 **Table 4-8: Test Case IoT-9-v4**

Test Case Field	Description
Parent Requirements	(CR-13) The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the gateway will be configured with all possible instantiations of that rule, insofar as each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the gateway.
Testable Requirements	(CR-13.a) The MUD file for a device shall contain a rule involving a domain that can resolve to multiple IP addresses when queried by the gateway. Flow rules for permitting access to each of those IP addresses will be inserted into the gateway for the device in question, and the device will be permitted to communicate with all of those IP addresses.
Description	Shows that if a domain in a MUD file rule resolves to multiple IP addresses when the address resolution is requested by the gateway, then 1. ACLs instantiating that MUD file rule corresponding to each of these IP addresses will be configured in the gateway for the IoT device associated with the MUD file, and

Test Case Field	Description
	2. The IoT device associated with the MUD file will be permitted to communicate with all the IP addresses to which that domain resolves
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_northsouth.json*
Preconditions	1. The gateway does not yet have any flow rules pertaining to the IoT device being used in the test. 2. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 4.1.3. (Therefore, the MUD file used in the test permits the device to send data to *www.updateserver.com.)* 3. The DNS server that the gateway uses resolves the domain *www.updateserver.com* to only one IP address. 4. The tester has access to a DNS server that will be used by the gateway and can configure it so that it will resolve the domain *www.updateserver.com* to any of these addresses when queried by gateway: x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. 5. A server is running at each of these three IP addresses.
Procedure	1. Verify that the gateway does not yet have any flow rules installed with respect to the IoT device being used in the test. 2. Run test IoT-1-v4. The result should be that the gateway has been configured to explicitly permit the IoT device to initiate communication with *www.updateserver.com*. 3. Attempt to reach *www.updateserver.com* on the device, and see that the gateway is then configured with ACLs that permit the IoT device to send data to IP addresses x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.

Test Case Field	Description
	4. Have the device in question attempt to connect to x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.
Expected Results	The gateway has had its configuration changed, i.e., it has been configured with ACLs that permit the IoT device to send data to multiple IP addresses (i.e., x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1). The IoT device is permitted to send data to each of the servers at these addresses.
Actual Results	**Flow rules:** `Every 2.0s: sudo ovs-ofctl dump-flows brmn001 --names | /opt/micronets-gw/bin/format-ofctl-dump` `Tue Jun  2 11:17:06 2020` `table=0  priority=500 n_packets=0 dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0 actions=drop` `table=0  priority=500 n_packets=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop` `table=0  priority=500 n_packets=0      icmp icmp_code=1 actions=drop` `table=0  priority=450 n_packets=7      in_port=LOCAL actions=resubmit( 200)` `table=0  priority=400 n_packets=2      in_port=wlp2s0 actions=resubmit( 100)` `table=0  priority=400 n_packets=33 in_port="wlp2s0.1861" actions=resubmit( 100)` `table=0  priority=0  n_packets=0      actions=output:diagout1` `table=100 priority=910 n_packets=0      ct_state=+est+trk udp actions=LOCAL` `table=100 priority=910 n_packets=0      ct_state=+rel+trk udp actions=LOCAL` `table=100 priority=910 n_packets=9      ct_state=-trk udp actions=ct(table=100)` `table=100 priority=905 n_packets=0      ct_state=+est+trk tcp actions=LOCAL` `table=100 priority=905 n_packets=0      ct_state=+rel+trk tcp actions=LOCAL`

Test Case Field	Description
	table=100 priority=905 n_packets=0 ct_state=-trk tcp actions=ct(table=100) table=100 priority=900 n_packets=2 dl_type=0x888e actions=resubmit(120) table=100 priority=850 n_packets=1 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=10 arp in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 actions=resubmit(120) table=100 priority=815 n_packets=2 udp in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 tp_dst=67 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=**52.89.85.207** actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=**54.191.221.118** actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 nw_dst=**54.201.49.86** actions=resubmit(120) table=100 priority=805 n_packets=20 in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 actions=output:diagout1 table=100 priority=800 n_packets=0 in_port="wlp2s0.1861" dl_src=00:c0:ca:98:42:37 actions=resubmit(110) table=100 priority=460 n_packets=0 in_port=wlp2s0 dl_type=0x888e actions=resubmit(120) table=100 priority=0 n_packets=0 actions=output:diagout1 **[Remaining flow rules omitted for brevity]** **<u>All IP communication attempts</u>:** pi@raspberrypi:~ $ wget **52.89.85.207** --2020-06-02 10:10:18-- http://52.89.85.207/

Test Case Field	Description
	Connecting to 52.89.85.207:80... **connected**. HTTP request sent, awaiting response... 301 Moved Perma-nently Location: https://52.89.85.207:443/ [following] --2020-06-02 10:10:18-- https://52.89.85.207/ Connecting to 52.89.85.207:443... connected. pi@raspberrypi:~ $ wget **54.201.49.86** --2020-06-02 10:10:39-- http://54.201.49.86/ Connecting to 54.201.49.86:80... **connected**. HTTP request sent, awaiting response... 301 Moved Perma-nently Location: https://54.201.49.86:443/ [following] --2020-06-02 10:10:39-- https://54.201.49.86/ Connecting to 54.201.49.86:443... **connected**. pi@raspberrypi:~ $ wget **54.191.221.118** --2020-06-02 10:10:46-- http://54.191.221.118/ Connecting to 54.191.221.118:80... **connected**. HTTP request sent, awaiting response... 301 Moved Perma-nently Location: https://54.191.221.118:443/ [following] --2020-06-02 10:10:47-- https://54.191.221.118/ Connecting to 54.191.221.118:443... **connected**.
Overall Result	Pass

544 IPv6 is not supported in this implementation.

545 *4.1.2.8 Test Case IoT-10-v4*

546 **Table 4-9: Test Case IoT-10-v4**

Test Case Field	Description
Parent Requirements	(CR-12) The IoT DDoS example implementation shall include a MUD manager that uses a cached MUD file rather than retrieve a new one if

Test Case Field	Description
	the cache-validity time period has not yet elapsed for the MUD file indicated by the MUD URL. The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.
Testable Requirements	(CR-12.a) The MUD manager shall check if the file associated with the MUD URL is present in its cache and shall determine that it is. (CR-12.a.1) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If so, the MUD manager shall apply the contents of the cached MUD file. (CR-12.a.2) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file. If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.
Description	Shows that, upon connection of a MUD-enabled IoT device, the gateway has already been configured to enforce the route filtering that is described in the cached MUD file for that device's MUD URL, assuming that the amount of time that has elapsed since the cached MUD file was retrieved is less than or equal to the number of hours in the file's cache-validity value. If the cache validity has expired for the respective file, the MUD manager should fetch a new MUD file from the MUD file server.
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*nist-model-fe_mycontroller.json*
Preconditions	1. All devices have been configured to use IPv4. 2. The gateway does not yet have any configuration settings pertaining to the IoT device being used in the test.

Test Case Field	Description
	3. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 4.1.3.
Procedure	Verify that the gateway does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Run test IoT-1-v4. 2. Within 24 hours (i.e., within the cache-validity period for the MUD file) of running test IoT-1-v4, a. Verify that the IoT device that was connected during test IoT-1-v4 is still up and running on the network. b. Power on a second IoT device whose bootstrapping information indicates that it will use the same MUD file as the device that was connected during test IoT-1-v4. 3. Power on the IoT device. 4. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages. 5. Open the onboarding application on the mobile phone and click READY TO SCAN. 6. Position the mobile phone's camera to read the device's QR code. Do this in a timely manner because there is a 60-second countdown for the device to exit DPP onboarding mode. 7. Input additional device-specific information into the mobile onboarding application as requested (must be done within the same 60-second time limit): a. Assign the device to its own unique micronets class (e.g., Medical) to which no other device is or will be assigned. b. Give the device a unique name (e.g., Device 1). 8. Click the ONBOARD button on the mobile application. This causes the onboarding application to send the device's bootstrapping information to the DPP configurator on the gateway via the operator's MSO portal and cloud infrastructure. 9. Wait. The following operations are being performed automatically in the operator's cloud infrastructure:

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>a. The Micronets Manager receives the bootstrapping information.
b. It looks up the URL of the device's MUD file.
c. It provides the MUD file URL to the MUD manager.
d. The MUD manager determines that it has this MUD file cached and checks that the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file.
i. If the cache validity has been exceeded, the MUD manager will fetch a new MUD file.
ii. Otherwise the MUD manager will use the cached MUD file.
e. The MUD manager translates the MUD file's contents into appropriate route filtering rules and installs these rules as ACLs onto the gateway for the IoT device in question so that this gateway is now configured to enforce the policies specified in the MUD file.</td></tr>
<tr><td>Expected Results</td><td>The gateway has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. The expected configuration should resemble the following details:

Cache is valid (the MUD manager does NOT retrieve the MUD file from the MUD file server):

Observing the MUD file server logs, notice that only one https Get method request for a MUD file goes out to the MUD file server. Within the next 24 hours, any additional devices onboarded using the same MUD file will not result in the MUD manager sending an https Get method request to the MUD file server to fetch a new MUD file.
Cache is not valid (the MUD manager does retrieve the MUD file from the MUD file server):</td></tr>
</table>

Test Case Field	Description
	Observing the MUD file server logs, notice that the MUD manager fetches a new copy of the MUD file and signature when the cache does not contain the MUD file of interest.
Actual Results	**IoT device initial onboarding event (no cache)**: `2020-06-11T19:37:17.249916385Z 2020-06-11 19:37:17,240 quart.serving: INFO 172.17.0.1:36502 POST /getFlowRules 1.0 200 322 8936` `2020-06-11T19:45:43.446237642Z 2020-06-11 19:45:43,445 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'}` `2020-06-11T19:45:43.446488467Z 2020-06-11 19:45:43,446 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json` `2020-06-11T19:45:43.446804181Z 2020-06-11 19:45:43,446 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json...` `2020-06-11T19:45:43.447009066Z 2020-06-11 19:45:43,446 micronets-mud-manager: INFO getMUDFile:` **`RETRIEVING`** `https://nccoe-server2.micronets.net/micronets-mud/`**`nist-model-fe_mycontroller.json`** **`2020-06-11T19:45:43.518411072Z 2020-06-11 19:45:43,518 micronets-mud-manager: DEBUG Saved MUD https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json`** `2020-06-11T19:45:43.518691567Z 2020-06-11 19:45:43,518 micronets-mud-manager: INFO Attempting to retrieve MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s` `2020-06-11T19:45:43.526955766Z 2020-06-11 19:45:43,526 micronets-mud-manager: INFO` **`Successfully retrieved MUD signature`** `https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s` **`2020-06-11T19:45:43.527737471Z 2020-06-11 19:45:43,527 micronets-mud-manager: INFO Saved MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.p7s`**

Test Case Field	Description
	2020-06-11T19:45:43.536591367Z 2020-06-11 19:45:43,536 micronets-mud-manager: DEBUG Signature validation command returned status 0 (Verification successful) 2020-06-11T19:45:43.536935401Z 2020-06-11 19:45:43,536 micronets-mud-manager: INFO MUD signature validation SUCCESS (MUD file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json, sig file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.p7s) 2020-06-11T19:45:43.537302394Z 2020-06-11 19:45:43,537 micronets-mud-manager: INFO cache-validity for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json is 48 hours 2020-06-11T19:45:43.537601948Z 2020-06-11 19:45:43,537 micronets-mud-manager: INFO expiration for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json is 2020-06-13T19:45:43.537438 2020-06-11T19:45:43.537948152Z 2020-06-11 19:45:43,537 micronets-mud-manager: INFO Dict for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: {'expiration-timestamp': 1592077543.537438} 2020-06-11T19:45:43.538473411Z 2020-06-11 19:45:43,538 micronets-mud-manager: INFO Wrote metadata for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: { 2020-06-11T19:45:43.538485520Z "**expiration-timestamp**": 1592077543.537438 2020-06-11T19:45:43.538490890Z } 2020-06-11T19:45:43.538495320Z 2020-06-11T19:45:43.538779055Z 2020-06-11 19:45:43,538 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-11T19:45:43.546885346Z [2020-06-11 19:45:43,546] 172.17.0.1:36594 POST /getMudInfo 1.0 200 115 101405 2020-06-11T19:45:43.574103085Z 2020-06-11 19:45:43,546 quart.serving: INFO 172.17.0.1:36594 POST /getMudInfo 1.0 200 115 101405 2020-06-11T19:45:43.983935332Z 2020-06-11 19:45:43,983 micronets-mud-manager: INFO getFlowRules called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json', 'version': '1.1', 'ip': '10.135.4.2'} 2020-06-11T19:45:43.984212636Z 2020-06-11 19:45:43,984 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-

Test Case Field	Description
	server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-11T19:45:43.984576320Z 2020-06-11 19:45:43,984 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json... 2020-06-11T19:45:43.985122858Z 2020-06-11 19:45:43,985 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 2020-06-11T19:45:43.985328855Z 2020-06-11 19:45:43,985 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-11T19:45:43.985692867Z 2020-06-11 19:45:43,985 micronets-mud-manager: INFO fromDeviceACL: [{'name': 'cl0-frdev', 'matches': {'ipv4': {'ietf-acldns:dst-dnsname': 'www.osmud.org', 'protocol': 6}, 'tcp': {'ietf-mud:direction-initiated': 'from-device', 'destination-port': {'operator': 'eq', 'port': 443}}}, 'actions': {'forwarding': 'accept'}}, {'name': 'myctl0-frdev', 'matches': {'ietf-mud:mud': {'my-controller': [None]}}, 'actions': {'forwarding': 'accept'}}] 2020-06-11T19:45:43.985885574Z 2020-06-11 19:45:43,985 micronets-mud-manager: INFO Found ietf-mud:mud: {'my-controller': [None]} 2020-06-11T19:45:43.987174428Z 2020-06-11 19:45:43,987 micronets-mud-manager: INFO acls: {'device': {'deviceId': '', 'macAddress': {'eui48': ''}, 'networkAddress': {'ipv4': '10.135.4.2'}, 'allowHosts': ['www.osmud.org', 'my-controller'], 'denyHosts': []}} 2020-06-11T19:45:43.989185189Z fromDeviceACL: dip: www.osmud.org 2020-06-11T19:45:43.989232148Z fromDeviceACL: dip: my-controller 2020-06-11T19:45:43.989236949Z [2020-06-11 19:45:43,988] 172.17.0.1:36620 POST /getFlowRules 1.0 200 296 5824 2020-06-11T19:45:43.990630231Z 2020-06-11 19:45:43,988 quart.serving: INFO 172.17.0.1:36620 POST /getFlowRules 1.0 200 296 5824
	IoT device—second onboarding event: **MUD manager—log file showing cached file in use:** 2020-06-12T14:39:21.769511212Z 2020-06-12 14:39:21,768 micronets-mud-manager: INFO getMudInfo called with: {'url':

Test Case Field	Description
	'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-12T14:39:21.770159883Z 2020-06-12 14:39:21,769 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json **2020-06-12T14:39:21.770708123Z 2020-06-12 14:39:21,770 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json...** 2020-06-12T14:39:21.773076957Z 2020-06-12 14:39:21,772 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 2020-06-12T14:39:21.773351346Z 2020-06-12 14:39:21,773 micronets-mud-manager: INFO getMUDFile: **LOADING** https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-12T14:39:21.774036637Z 2020-06-12 14:39:21,773 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-12T14:39:21.795798112Z [2020-06-12 14:39:21,795] 172.17.0.1:36724 POST /getMudInfo 1.0 200 115 46749 2020-06-12T14:39:21.798249385Z 2020-06-12 14:39:21,795 quart.serving: INFO 172.17.0.1:36724 POST /getMudInfo 1.0 200 115 46749 2020-06-12T14:46:33.851215222Z 2020-06-12 14:46:33,850 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-12T14:46:33.851433703Z 2020-06-12 14:46:33,851 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-12T14:46:33.851736073Z 2020-06-12 14:46:33,851 micronets-mud-manager: INFO **getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json**... 2020-06-12T14:46:33.852175554Z 2020-06-12 14:46:33,852 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 2020-06-12T14:46:33.852385904Z 2020-06-12 14:46:33,852 micronets-mud-manager: INFO getMUDFile: **LOADING** https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json **from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json)**

Test Case Field	Description
	2020-06-12T14:46:33.852709545Z 2020-06-12 14:46:33,852 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-12T14:46:33.855891368Z [2020-06-12 14:46:33,855] 172.17.0.1:36812 POST /getMudInfo 1.0 200 115 5306 2020-06-12T14:46:33.857513729Z 2020-06-12 14:46:33,855 quart.serving: INFO 172.17.0.1:36812 POST /getMudInfo 1.0 200 115 5306 2020-06-12T14:48:43.560538164Z 2020-06-12 14:48:43,560 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-12T14:48:43.560876515Z 2020-06-12 14:48:43,560 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-12T14:48:43.561223856Z 2020-06-12 14:48:43,561 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json... 2020-06-12T14:48:43.561778395Z 2020-06-12 14:48:43,561 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 2020-06-12T14:48:43.562095137Z 2020-06-12 14:48:43,561 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-12T14:48:43.562634237Z 2020-06-12 14:48:43,562 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-12T14:48:43.569593236Z [2020-06-12 14:48:43,569] 172.17.0.1:36864 POST /getMudInfo 1.0 200 115 7932 2020-06-12T14:48:43.571181238Z 2020-06-12 14:48:43,569 quart.serving: INFO 172.17.0.1:36864 POST /getMudInfo 1.0 200 115 7932 2020-06-12T14:53:07.505904799Z 2020-06-12 14:53:07,505 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-12T14:53:07.506221249Z 2020-06-12 14:53:07,506 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-12T14:53:07.506600419Z 2020-06-12 14:53:07,506 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-

Test Case Field	Description
	model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json... 2020-06-12T14:53:07.507296190Z 2020-06-12 14:53:07,507 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 2020-06-12T14:53:07.507898661Z 2020-06-12 14:53:07,507 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-12T14:53:07.508470932Z 2020-06-12 14:53:07,508 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-12T14:53:07.515602561Z [2020-06-12 14:53:07,515] 172.17.0.1:36902 POST /getMudInfo 1.0 200 115 9685 2020-06-12T14:53:07.516735033Z 2020-06-12 14:53:07,515 quart.serving: INFO 172.17.0.1:36902 POST /getMudInfo 1.0 200 115 9685
	Invalid cache: 2020-06-15T14:13:01.654112995Z 2020-06-15 14:13:01,653 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-15T14:13:01.655088176Z 2020-06-15 14:13:01,654 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-15T14:13:01.656192927Z 2020-06-15 14:13:01,655 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json... 2020-06-15T14:13:01.658547789Z 2020-06-15 14:13:01,658 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 **2020-06-15T14:13:01.658875150Z 2020-06-15 14:13:01,658 micronets-mud-manager: INFO getMUDFile: EXPIRING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json)** 2020-06-15T14:13:01.659399130Z 2020-06-15 14:13:01,659 micronets-mud-manager: INFO getMUDFile: RETRIEVING

Test Case Field	Description
	https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json **2020-06-15T14:13:01.699355481Z 2020-06-15 14:13:01,698 micronets-mud-manager: DEBUG Saved MUD https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json** 2020-06-15T14:13:01.699620761Z 2020-06-15 14:13:01,699 micronets-mud-manager: INFO Attempting to retrieve MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s 2020-06-15T14:13:01.706113148Z 2020-06-15 14:13:01,705 micronets-mud-manager: INFO Successfully retrieved MUD signature https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s **2020-06-15T14:13:01.707347299Z 2020-06-15 14:13:01,707 micronets-mud-manager: INFO Saved MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.p7s** 2020-06-15T14:13:01.738890831Z 2020-06-15 14:13:01,738 micronets-mud-manager: DEBUG Signature validation command returned status 0 (Verification successful) 2020-06-15T14:13:01.739395162Z 2020-06-15 14:13:01,739 micronets-mud-manager: INFO MUD signature validation SUCCESS (MUD file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json, sig file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.p7s) 2020-06-15T14:13:01.739940012Z 2020-06-15 14:13:01,739 micronets-mud-manager: INFO cache-validity for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json is 48 hours 2020-06-15T14:13:01.740295383Z 2020-06-15 14:13:01,740 micronets-mud-manager: INFO expiration for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json is 2020-06-17T14:13:01.740045 2020-06-15T14:13:01.740630103Z 2020-06-15 14:13:01,740 micronets-mud-manager: INFO Dict for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: {'expiration-timestamp': 1592403181.740045} 2020-06-15T14:13:01.741795074Z 2020-06-15 14:13:01,741 micronets-mud-manager: INFO Wrote metadata for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: {

Test Case Field	Description
	2020-06-15T14:13:01.741868954Z "expiration-timestamp": 1592403181.740045 2020-06-15T14:13:01.741875624Z } 2020-06-15T14:13:01.741880154Z 2020-06-15T14:13:01.742275394Z 2020-06-15 14:13:01,742 mi-cronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-15T14:13:01.755931658Z [2020-06-15 14:13:01,752] 172.17.0.1:37600 POST /getMudInfo 1.0 200 115 103244 2020-06-15T14:13:01.756955469Z 2020-06-15 14:13:01,752 quart.serving: INFO 172.17.0.1:37600 POST /getMudInfo 1.0 200 115 103244
Overall Results	Pass

IPv6 is not supported in this implementation.

4.1.2.9 Test Case IoT-11-v4

Table 4-10: Test Case IoT-11-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file).
Testable Requirements	(CR-1.a) The device's MUD file is located by using two items in the device's bootstrapping information (which is encoded in its QR code): the information element and the public bootstrapping key. (CR-1.a.1) The information element identifies a device vendor, and each vendor is assumed to have a well-known location for serving MUD files, so this element identifies the location of the device's MUD file server. The public bootstrapping key of the device identifies the device's MUD file.

Test Case Field	Description
Description	Shows that the IoT DDoS example implementation includes IoT devices that are associated with MUD files based on two of the fields in their bootstrapping information (information element and public key), which are encoded in their QR codes. (Note that in future releases, the URL for the MUD file is expected to be provided explicitly, as specified in the latest Wi-Fi Easy Connect protocol specification, so in the future there will be no need to look up the MUD file URL based on other bootstrapping fields.)
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1
IoT Device(s) Under Test	Raspberry Pi 1
MUD File(s) Used	*nist-model-fe_mycontroller.json, nist-model-fe_manufacturer2.json*
Preconditions	1. One device (Device 1) to be used has a QR code with values for its information element and public key fields that indicate the device's MUD file is *mudfile-sensor.json and it is located on the server hosted by the manufacturer indicated by the code in the information element field*. 2. Two other devices (Device 2 and Device 3) to be used each have QR codes with values for their information element and public key fields that indicate the device's MUD file is *nist-model-fe_manufacturer2.json and it is located on the server hosted by the manufacturer indicated by the code in the information element field*. *3.* The appropriate curl command was run to associate the public key of Device 1 with the MUD file *(nist-model-fe_mycontroller.json).* 4. The appropriate curl command was run to associate the public keys of Device 2 and Device 3 (which are different from each other) with the same MUD file *(nist-model-fe_ manufacturer2.json).* 5. The testers have a QR code decoder, i.e., something like https://zxing.org/w/decode.jspx.

Test Case Field	Description
Procedure	1. Do for each of the three devices: a. Power on the IoT device. b. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages. c. Use the QR code decoder to determine the value in the QR code information element and public key fields. 2. If the three devices are supposed to all be from the same manufacturer, verify that they have equivalent information element field values; if one of the devices is supposed to be from a manufacturer different from the other two, verify that its information element field value is different. 3. Verify that all three devices have different public keys. 4. At this point, we have verified that the information in the QR codes is specific to the devices. 5. We also know whether the two MUD files are expected to be on the same server (i.e., if their information element fields are identical) or on different servers (i.e., their information element fields are different). 6. Next, verify that these different QR code values cause the devices to be associated with different MUD files. 1. Verify that the MUD files of the IoT devices to be used are not currently cached at the MUD manager. 2. Run test IoT-1-v4 using Device 1 (the one with a QR code that is different from the QR code that is shared by the other two devices). 3. Verify that the MUD file that was retrieved from the MUD file server when this device was onboarded is *nist-model-fe_mycontroller.json.* 4. Run test IoT-1-v4 using Device 2. 5. Verify that the MUD file that was retrieved from the MUD file server when this device was onboarded is *nist-model-fe_manufacturer2.json* 6. Run test IoT-1-v4 using Device 3. 7. Verify that no MUD file was retrieved but that the ACLs installed on the gateway that apply to this device are identical to the ACLs that

Test Case Field	Description
	were installed on the gateway for the second device (i.e., they enforce the MUD rules specified in *nist-model-fe_manufacturer2.json)*.
Expected Results	Each verification step described in the procedure field can be performed as expected.
Actual Results	**Confirm pub keys**: **Pi-1**: pi@pi-1:~ $ cat micronets-pi3/keys/proto-pi.dpp.pub **MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=** **Pi-2**: pi@pi-2:~ $ cat micronets-pi3/keys/proto-pi.dpp.pub **MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADOqawv+0iCORm2+MoB-tFp9A27HTY3g5bIvFglvJLvXS0=** **Pi-3**: pi@pi-3:~ $ cat micronets-pi3/keys/proto-pi.dpp.pub **MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgAC-cgm5sipeXL5oeF+xpsIFkQkPkPASzQywP2K8Peu010E=** --- **QR code results**: **Pi-1**: DPP:C:81/1;M:00:c0:ca:97:d1:1f;I:TEST;K:**MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=**;; **Pi-2**: DPP:C:81/1;M:00:c0:ca:98:42:37;I:TEST;K:**MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADOqawv+0iCORm2+MoB-tFp9A27HTY3g5bIvFglvJLvXS0=**;; **Pi-3**: DPP:C:81/1;M:00:c0:ca:98:42:2d;I:TEST;K:**MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACcgm5sipeXL5oeF+xpsIFkQkPk-PASzQywP2K8Peu010E=**;;

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<u>Device’s MUD files</u>:

<u>Pi-1</u>:
$ curl -L https://nccoe-server1.micronets.net/mud/v1/mud-url/TEST/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADSOi8J6JCJJ0h4+NmPtARUgfMrQ2mcCazdJNfNdgTkZM=

https://nccoe-server2.micronets.net/micronets-mud/<u>nist-model-fe_mycontroller.json</u>

<u>Pi-2</u>:
$ curl -L https://nccoe-server1.micronets.net/mud/v1/mud-url/TEST/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgA-DOqawv+0iCORm2+MoBtFp9A27HTY3g5bIvFglvJLvXS0=

https://nccoe-server2.micronets.net/micronets-mud/<u>nist-model-fe_mycontroller.json</u>
<u>Pi-3</u>:
$ curl -L https://nccoe-server1.micronets.net/mud/v1/mud-url/TEST/MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgAC-cgm5sipeXL5oeF+xpsIFkQkPkPASzQywP2K8Peu010E=

https://nccoe-server2.micronets.net/micronets-mud/<u>nist-model-fe_manufacturer2.json</u>
<hr>
<u>Check cache file</u>:
micronets-dev@nccoe-server1:~$ ls -1 /var/cache/micronets-mud/
nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer1.json
nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer1.json.md
nccoe-server2.micronets.net_micronets-mud_nist-model-fe_northsouth.json
nccoe-server2.micronets.net_micronets-mud_nist-model-fe_northsouth.json.md
<hr>
<u>MUD manager logs</u>:

<u>Pi-3 onboard</u>:
2020-06-11T19:36:33.733008675Z [2020-06-11 19:36:33,732] 172.17.0.1:36424 POST /getMudInfo 1.0 200 123 52222
2020-06-11T19:36:33.734978384Z 2020-06-11 19:36:33,732 quart.serving: INFO 172.17.0.1:36424 POST /getMudInfo 1.0 200 123 52222
</td></tr>
</table>

Test Case Field	Description
	2020-06-11T19:37:16.917704511Z 2020-06-11 19:37:16,917 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json'} 2020-06-11T19:37:16.918005424Z 2020-06-11 19:37:16,917 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/**nist-model-fe_manufacturer2.json** 2020-06-11T19:37:16.918322588Z 2020-06-11 19:37:16,918 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer2.json... 2020-06-11T19:37:16.918747651Z 2020-06-11 19:37:16,918 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer2.json.md expiration is 2020-06-13T19:36:33.723673 2020-06-11T19:37:16.918957814Z 2020-06-11 19:37:16,918 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer2.json) 2020-06-11T19:37:16.919324757Z 2020-06-11 19:37:16,919 micronets-mud-manager: INFO mud info: {'mfgName': 'www.gmail.com', 'modelName': 'fe-manufacturer2.json', 'mudUrl': 'https://www.gmail.com/fe-manufacturer2.json'} 2020-06-11T19:37:16.922393707Z [2020-06-11 19:37:16,922] 172.17.0.1:36480 POST /getMudInfo 1.0 200 123 5412 2020-06-11T19:37:16.923933922Z 2020-06-11 19:37:16,922 quart.serving: INFO 172.17.0.1:36480 POST /getMudInfo 1.0 200 123 5412 2020-06-11T19:37:17.232818457Z 2020-06-11 19:37:17,232 micronets-mud-manager: INFO getFlowRules called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json', 'version': '1.1', 'ip': '10.135.3.2'} 2020-06-11T19:37:17.233130840Z 2020-06-11 19:37:17,232 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json 2020-06-11T19:37:17.233467433Z 2020-06-11 19:37:17,233 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer2.json... 2020-06-11T19:37:17.234024099Z 2020-06-11 19:37:17,233 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer2.json.md expiration is 2020-06-13T19:36:33.723673

Test Case Field	Description
	2020-06-11T19:37:17.234325612Z 2020-06-11 19:37:17,234 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_manufacturer2.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_manufacturer2.json) 2020-06-11T19:37:17.234895988Z 2020-06-11 19:37:17,234 micronets-mud-manager: INFO fromDeviceACL: [{'name': 'cl0-frdev', 'matches': {'ipv4': {'ietf-acldns:dst-dnsname': 'www.osmud.org', 'protocol': 6}, 'tcp': {'ietf-mud:direction-initiated': 'from-device'}}, 'actions': {'forwarding': 'accept'}}, {'name': 'man0-frdev', 'matches': {'ietf-mud:mud': {'manufacturer': 'mudfiles.nist.getyikes.com'}, 'ipv4': {'protocol': 6}, 'tcp': {'ietf-mud:direction-initiated': 'to-device', 'destination-port': {'operator': 'eq', 'port': 80}}}, 'actions': {'forwarding': 'accept'}}] 2020-06-11T19:37:17.235400092Z 2020-06-11 19:37:17,235 micronets-mud-manager: INFO Found ietf-mud:mud: {'manufacturer': 'mudfiles.nist.getyikes.com'} 2020-06-11T19:37:17.235627615Z 2020-06-11 19:37:17,235 micronets-mud-manager: INFO acls: {'device': {'deviceId': '', 'macAddress': {'eui48': ''}, 'networkAddress': {'ipv4': '10.135.3.2'}, 'allowHosts': ['www.osmud.org', 'manufacturer:mudfiles.nist.getyikes.com'], 'denyHosts': []}} 2020-06-11T19:37:17.241142449Z fromDeviceACL: dip: www.osmud.org 2020-06-11T19:37:17.241164739Z fromDeviceACL: found MUD extension param: mudfiles.nist.getyikes.com 2020-06-11T19:37:17.241168089Z fromDeviceACL: dip: manufacturer:mudfiles.nist.getyikes.com 2020-06-11T19:37:17.241171119Z [2020-06-11 19:37:17,240] 172.17.0.1:36502 POST /getFlowRules 1.0 200 322 8936 2020-06-11T19:37:17.244916385Z 2020-06-11 19:37:17,240 quart.serving: INFO 172.17.0.1:36502 POST /getFlowRules 1.0 200 322 8936 **Pi-1 onboard**: 2020-06-15T14:13:01.654112995Z 2020-06-15 14:13:01,653 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-15T14:13:01.655088176Z 2020-06-15 14:13:01,654 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/**nist-model-fe_mycontroller.json** 2020-06-15T14:13:01.656192927Z 2020-06-15 14:13:01,655 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json...

Test Case Field	Description
	2020-06-15T14:13:01.658547789Z 2020-06-15 14:13:01,658 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-13T19:45:43.537438 2020-06-15T14:13:01.658875150Z 2020-06-15 14:13:01,658 micronets-mud-manager: INFO getMUDFile: EXPIRING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-15T14:13:01.659399130Z 2020-06-15 14:13:01,659 micronets-mud-manager: INFO getMUDFile: RETRIEVING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-15T14:13:01.699355481Z 2020-06-15 14:13:01,698 micronets-mud-manager: DEBUG Saved MUD https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json 2020-06-15T14:13:01.699620761Z 2020-06-15 14:13:01,699 micronets-mud-manager: INFO Attempting to retrieve MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s 2020-06-15T14:13:01.706113148Z 2020-06-15 14:13:01,705 micronets-mud-manager: INFO Successfully retrieved MUD signature https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s 2020-06-15T14:13:01.707347299Z 2020-06-15 14:13:01,707 micronets-mud-manager: INFO Saved MUD signature from https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.p7s to /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.p7s 2020-06-15T14:13:01.738890831Z 2020-06-15 14:13:01,738 micronets-mud-manager: DEBUG Signature validation command returned status 0 (Verification successful) 2020-06-15T14:13:01.739395162Z 2020-06-15 14:13:01,739 micronets-mud-manager: INFO MUD signature validation SUCCESS (MUD file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json, sig file /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.p7s) 2020-06-15T14:13:01.739940012Z 2020-06-15 14:13:01,739 micronets-mud-manager: INFO cache-validity for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json is 48 hours 2020-06-15T14:13:01.740295383Z 2020-06-15 14:13:01,740 micronets-mud-manager: INFO expiration for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json is 2020-06-17T14:13:01.740045

Test Case Field	Description
	2020-06-15T14:13:01.740630103Z 2020-06-15 14:13:01,740 micronets-mud-manager: INFO Dict for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: {'expiration-timestamp': 1592403181.740045} 2020-06-15T14:13:01.741795074Z 2020-06-15 14:13:01,741 micronets-mud-manager: INFO Wrote metadata for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: { 2020-06-15T14:13:01.741868954Z "expiration-timestamp": 1592403181.740045 2020-06-15T14:13:01.741875624Z } 2020-06-15T14:13:01.741880154Z 2020-06-15T14:13:01.742275394Z 2020-06-15 14:13:01,742 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'} 2020-06-15T14:13:01.755931658Z [2020-06-15 14:13:01,752] 172.17.0.1:37600 POST /getMudInfo 1.0 200 115 103244 **Pi-2 onboard:** 2020-06-15T14:13:01.755931658Z [2020-06-15 14:13:01,752] 172.17.0.1:37600 POST /getMudInfo 1.0 200 115 103244 2020-06-15T14:13:01.756955469Z 2020-06-15 14:13:01,752 quart.serving: INFO 172.17.0.1:37600 POST /getMudInfo 1.0 200 115 103244 2020-06-15T18:48:19.422617510Z 2020-06-15 18:48:19,422 micronets-mud-manager: INFO getMudInfo called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json'} 2020-06-15T18:48:19.423262681Z 2020-06-15 18:48:19,423 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/**nist-model-fe_mycontroller.json** 2020-06-15T18:48:19.423891632Z 2020-06-15 18:48:19,423 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json... **2020-06-15T18:48:19.424628272Z 2020-06-15 18:48:19,424 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-17T14:13:01.740045** 2020-06-15T18:48:19.424908472Z 2020-06-15 18:48:19,424 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-15T18:48:19.425380493Z 2020-06-15 18:48:19,425 micronets-mud-manager: INFO mud info: {'mfgName': 'nist', 'modelName': 'fe-mycontroller', 'mudUrl': 'https://mud-files.nist.getyikes.com/fe-mycontroller'}

Test Case Field	Description
	2020-06-15T18:48:19.432904899Z [2020-06-15 18:48:19,432] 172.17.0.1:38052 POST /getMudInfo 1.0 200 115 11251 2020-06-15T18:48:19.435370410Z 2020-06-15 18:48:19,432 quart.serving: INFO 172.17.0.1:38052 POST /getMudInfo 1.0 200 115 11251 2020-06-15T18:48:19.873090877Z 2020-06-15 18:48:19,872 micronets-mud-manager: INFO getFlowRules called with: {'url': 'https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json', 'version': '1.1', 'ip': '10.135.1.2'} 2020-06-15T18:48:19.873446047Z 2020-06-15 18:48:19,873 micronets-mud-manager: INFO getMUDFile: url: https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json 2020-06-15T18:48:19.873952898Z 2020-06-15 18:48:19,873 micronets-mud-manager: INFO getMUDFile: mud filepath for https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json... 2020-06-15T18:48:19.874521568Z 2020-06-15 18:48:19,874 micronets-mud-manager: DEBUG getMUDFile: /mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json.md expiration is 2020-06-17T14:13:01.740045 2020-06-15T18:48:19.875145659Z 2020-06-15 18:48:19,874 micronets-mud-manager: INFO getMUDFile: LOADING https://nccoe-server2.micronets.net/micronets-mud/nist-model-fe_mycontroller.json from CACHE (/mud-cache-dir/nccoe-server2.micronets.net_micronets-mud_nist-model-fe_mycontroller.json) 2020-06-15T18:48:19.875899349Z 2020-06-15 18:48:19,875 micronets-mud-manager: INFO fromDeviceACL: [{'name': 'cl0-frdev', 'matches': {'ipv4': {'ietf-acldns:dst-dnsname': 'www.osmud.org', 'protocol': 6}, 'tcp': {'ietf-mud:direction-initiated': 'from-device', 'destination-port': {'operator': 'eq', 'port': 443}}}, 'actions': {'forwarding': 'accept'}}, {'name': 'myctl0-frdev', 'matches': {'ietf-mud:mud': {'my-controller': [None]}}, 'actions': {'forwarding': 'accept'}}] 2020-06-15T18:48:19.876239609Z 2020-06-15 18:48:19,876 micronets-mud-manager: INFO Found ietf-mud:mud: {'my-controller': [None]} 2020-06-15T18:48:19.876526189Z 2020-06-15 18:48:19,876 micronets-mud-manager: INFO acls: {'device': {'deviceId': '', 'macAddress': {'eui48': ''}, 'networkAddress': {'ipv4': '10.135.1.2'}, 'allowHosts': ['www.osmud.org', 'my-controller'], 'denyHosts': []}} 2020-06-15T18:48:19.885638526Z fromDeviceACL: dip: www.osmud.org 2020-06-15T18:48:19.885670277Z fromDeviceACL: dip: my-controller 2020-06-15T18:48:19.885675247Z [2020-06-15 18:48:19,885] 172.17.0.1:38076 POST /getFlowRules 1.0 200 296 13409

Test Case Field	Description
	2020-06-15T18:48:19.887010138Z 2020-06-15 18:48:19,885 quart.serving: INFO 172.17.0.1:38076 POST /getFlowRules 1.0 200 296 13409 --- **Get micronets:** `{` `"_id": "5ee7bf78ab3e8358c185e759",` `"id": "subscriber-001",` `"name": "Subscriber 001",` `"ssid": "micronets-gw",` `"gatewayId": "micronets-gw",` `"micronets": [` `{` `"name": "Medical",` `"class": "Medical",` `"micronet-subnet-id": "Medical",` `"trunk-gateway-port": "2",` `"trunk-gateway-ip": "10.36.32.124",` `"dhcp-server-port": "LOCAL",` `"dhcp-zone": "10.135.1.0/24",` `"ovs-bridge-name": "brmn001",` `"ovs-manager-ip": "10.36.32.124",` `"micronet-subnet": "10.135.1.0/24",` `"micronet-gateway-ip": "10.135.1.1",` `"connected-devices": [` `{` `"device-mac": "00:C0:CA:98:42:37",` `"device-name": "Pi2-t11",` `"device-id":` `"9f58599efce4680ee0c21efe0b98e27f8a7a8958",` `"device-openflow-port": "2",` `"device-ip": "10.135.1.2"` `}` `],` `"micronet-id": "2309484987"` `},` `{` `"name": "Security",` `"class": "Security",` `"micronet-subnet-id": "Security",` `"trunk-gateway-port": "2",` `"trunk-gateway-ip": "10.36.32.124",` `"dhcp-server-port": "LOCAL",` `"dhcp-zone": "10.135.2.0/24",` `"ovs-bridge-name": "brmn001",` `"ovs-manager-ip": "10.36.32.124",` `"micronet-subnet": "10.135.2.0/24",` `"micronet-gateway-ip": "10.135.2.1",` `"connected-devices": [` `{`

Test Case Field	Description
	"device-mac": "00:C0:CA:97:D1:1F", "device-name": "Pi1-t11", "device-id": "463165abc19725aefffc39def13ce09b17167fba", "device-openflow-port": "2", "device-ip": "10.135.2.2" }], "micronet-id": "2160025251" }, { "name": "Personal", "class": "Personal", "micronet-subnet-id": "Personal", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL", "dhcp-zone": "10.135.3.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.3.0/24", "micronet-gateway-ip": "10.135.3.1", "connected-devices": [{ "device-mac": "00:C0:CA:98:42:2D", "device-name": "Pi3-t11", "device-id": "da34c7219c2c97f0e2c2838e66c725d137f3c097", "device-openflow-port": "2", "device-ip": "10.135.3.2" }], "micronet-id": "2154160396" }], "createdAt": "2020-06-15T18:35:36.968Z", "updatedAt": "2020-06-16T17:18:25.834Z", "__v": 0 } --- **View flow rules:** Every 2.0s: sudo ovs-ofctl dump-flows brmn001 --names | /opt/micronets-gw/bin/format-ofctl-dump Tue Jun 16 13:19:32 2020 table=0 priority=500 n_packets=0 dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0 actions=drop table=0 priority=500 n_packets=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop

Test Case Field	Description
	table=0 priority=500 n_packets=0 icmp icmp_code=1 actions=drop table=0 priority=450 n_packets=25 in_port=LOCAL actions=resubmit(200) table=0 priority=400 n_packets=15 in_port="wlp2s0.3221" actions=resubmit(100) table=0 priority=400 n_packets=18 in_port="wlp2s0.2484" actions=resubmit(100) table=0 priority=400 n_packets=2 in_port=wlp2s0 actions=resubmit(100) table=0 priority=400 n_packets=39 in_port="wlp2s0.3854" actions=resubmit(100) table=0 priority=0 n_packets=0 actions=output:diagout1 table=100 priority=910 n_packets=0 ct_state=+est+trk udp actions=LOCAL table=100 priority=910 n_packets=0 ct_state=+rel+trk udp actions=LOCAL table=100 priority=910 n_packets=38 ct_state=-trk udp actions=ct(table=100) table=100 priority=905 n_packets=0 ct_state=+est+trk tcp actions=LOCAL table=100 priority=905 n_packets=0 ct_state=+rel+trk tcp actions=LOCAL table=100 priority=905 n_packets=0 ct_state=-trk tcp actions=ct(table=100) table=100 priority=900 n_packets=2 dl_type=0x888e actions=resubmit(120) table=100 priority=850 n_packets=3 ip in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=850 n_packets=4 ip in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d nw_dst=10.135.3.1 actions=resubmit(120) table=100 priority=850 n_packets=5 ip in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.2.1 actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 udp in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=0 udp in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 tp_dst=67 actions=resubmit(120)

Test Case Field	Description
	table=100 priority=815 n_packets=2 udp in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=6 arp in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f actions=resubmit(120) table=100 priority=815 n_packets=6 arp in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 actions=resubmit(120) table=100 priority=815 n_packets=8 arp in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.2.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f nw_dst=104.237.132.42 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f nw_dst=198.71.233.87 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 nw_dst=104.237.132.42 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 nw_dst=198.71.233.87 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d nw_dst=10.135.1.2 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d nw_dst=10.135.2.2 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d nw_dst=10.135.3.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d nw_dst=198.71.233.87 actions=resubmit(120) table=100 priority=805 n_packets=25 in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d actions=output:diagout1 table=100 priority=805 n_packets=6 in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 actions=output:diagout1 table=100 priority=805 n_packets=7 in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f actions=output:diagout1

Test Case Field	Description
	`table=100 priority=800 n_packets=0 in_port="wlp2s0.2484" dl_src=00:c0:ca:97:d1:1f actions=re-submit( 110)` `table=100 priority=800 n_packets=0 in_port="wlp2s0.3221" dl_src=00:c0:ca:98:42:37 actions=re-submit( 110)` `table=100 priority=800 n_packets=0 in_port="wlp2s0.3854" dl_src=00:c0:ca:98:42:2d actions=re-submit( 110)` `table=100 priority=460 n_packets=0 in_port=wlp2s0 dl_type=0x888e actions=resubmit( 120)` `table=100 priority=0 n_packets=0 actions=output:di-agout1`
Overall Results	Pass

4.1.3 MUD Files

This section contains the MUD files that were used in the Build 4 functional demonstration.

4.1.3.1 nist-model-fe_northsouth.json

The complete *nist-model-fe_northsouth.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_northsouth.json

4.1.3.2 nist-model-fe_mycontroller.json

The complete *nist-model-fe_mycontroller.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_mycontroller.json

4.1.3.3 nist-model-fe_controller_anyport.json

The complete *nist-model-fe_controller_anyport.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_controller_anyport.json

4.1.3.4 nist-model-fe_expiredcert.json

The complete *nist-model-fe_expiredcert.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_expiredcert.json

4.1.3.5 nist-model-fe_invalidsig.json

The complete *nist-model-fe_invalidsig.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_invalidsig.json

4.1.3.6 nist-model-fe_manufacturer1.json

The complete nist-model-fe_manufacturer1.json MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_manufacturer1.json

4.1.3.7 nist-model-fe_manufacturer2.json

The complete *nist-model-fe_manufacturer2.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_manufacturer2.json

4.1.3.8 nist-model-fe_manufacturer-from.json

The complete *nist-model-fe_manufacturer-from.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_manufacturer-from.json

4.1.3.9 nist-model-fe_manufacturer-to.json

The complete *nist-model-fe_manufacturer-to.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_manufacturer-to.json

4.1.3.10 nist-model-fe_samemanufacturer.json

The complete *nist-model-fe_samemanufacturer.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_samemanufacturer.json

4.1.3.11 nist-model-fe_samemanufacturer-to.json

The complete *nist-model-fe_samemanufacturer-to.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_samemanufacturer-to.json

4.1.3.12 nist-model-fe_samemanufacturer-from.json

The complete *nist-model-fe_samemanufacturer-from.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_samemanufacturer-from.json

4.1.3.13 nist-model-fe_localnetwork_anyport.json

The complete *nist-model-fe_localnetwork_anyport.json* MUD file has been linked to this document. To access this MUD file, please click the link below.

nist-model-fe_localnetwork_anyport.json

4.2 Demonstration of Non-MUD-Related Capabilities

In addition to supporting MUD, Build 3 supports DPP onboarding and provides the capability to place devices onto specific micronets when they are provisioned on the network. Micronets are subnetworks that isolate devices. Devices that are on one Micronet are not able to exchange traffic with devices on other Micronets (unless overridden by their MUD files). Some Micronet classes have been predefined. When a device is onboarded using the DPP onboarding mobile application, the user is asked to input or confirm the class of Micronet to which the device should be assigned.

4.2.1 Non-MUD-Related Functional Capabilities

Table 4-11 lists the non-MUD-related capabilities that were demonstrated for Build 3. We use the letter “M” as a prefix for these functional capability identifiers in the table below because these capabilities are specific to Build 3, which uses Micronets technology. The lowercase “n” after the “M” is shorthand for “non-.” Hence, test MnMUD-1 is the first test to demonstrate the Micronets non-MUD capabilities.

Table 4-11: Non-MUD-Related Functional Capabilities Demonstrated

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
M-1	**DPP onboarding–** The device can be onboarded to the network by using DPP.			MnMUD-1
M-1.a		The IoT device can be put into DPP onboarding mode, i.e., it can display a QR code and listen for DPP messages.	The QR code contains the bootstrapping information for the device.	MnMUD-1
M-1.b		The IoT device's bootstrapping information can be conveyed to the DPP configurator.	The Micronets mobile application can act as the DPP configurator's bootstrapping information reader by scanning the QR code and conveying its content to the configurator.	MnMUD-1
M-1.c		The DPP configurator can support the authentication phase of the DPP onboarding process.	The configurator initiates a three-way protocol exchange to authenticate the device (request, respond, confirm).	MnMUD-1
M-1.d		The DPP configurator can support the configuration phase of the DPP onboarding process.	The configurator initiates a three-way protocol exchange to configure the device (request, respond, result) so that the device is provided with the Service Set Identifier (SSID) and cre-	MnMUD-1

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
			dential it needs to connect to the local network.	
M-2	**Network connection—**the device that has been onboarded with DPP can successfully connect to the network.			MnMUD-1
M-2.a		The device presents its credential to the network with the appropriate SSID.	The device is assigned an IP address on the appropriate network.	MnMUD-1
M-3	**Device Micronet classification–** Upon connection to the network, each device is placed into its intended Micronet class.			MnMUD-2
M-3.a		The Micronet class of each device can be provided as part of the bootstrapping information.	The user specifies the device micronets class by using the onboarding app on the mobile phone (after scanning the QR code).	MnMUD-2
M-3.b		Devices that are in the same Micronet class can communicate with each other		MnMUD-2

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
		(assuming this is not contradicted by the devices' MUD files).		
M-3.c		Devices that are in different Micronet classes cannot communicate with each other (assuming this is not contradicted by the devices' MUD files).		MnMUD-2
M-4	Each device that is onboarded using DPP is assigned a unique credential.			MnMUD-3
M-4.a		The Micronets Gateway can be configured to disconnect a device that has been onboarded using DPP.	The other devices remain connected.	MnMUD-3

4.2.2 Exercises to Demonstrate the Above Non-MUD-Related Capabilities

This section contains the exercises that were performed to verify that Build 3 supports the non-MUD-related capabilities listed in Table 4-11.

4.2.2.1 Exercise MnMUD-1

Table 4-12: Exercise MnMUD-1

Exercise Field	Description
Parent Capability	(M-1) DPP onboarding–The device can be onboarded to the network by using DPP. (M-2) Network connection—The device that has been onboarded with DPP can successfully connect to the network.
Subrequirement(s) of Parent Capability to Be Demonstrated	(M-1.a) The IoT device can be put into DPP onboarding mode, i.e., it can display a QR code and listen for DPP messages. The QR code contains the bootstrapping information for the device. (M-1.b) The IoT device's bootstrapping information can be conveyed to the DPP configurator. The Micronets mobile application can act as the DPP configurator's bootstrapping information reader by scanning the QR code and conveying its content to the configurator. (M-1.c) The DPP configurator can support the authentication phase of the DPP onboarding process. The configurator initiates a three-way protocol exchange to authenticate the device (request, respond conform). (M-1.d) The DPP configurator can support the configuration phase of the DPP onboarding process. The configurator initiates a three-way protocol exchange to configure the device (request, respond, result) so that the device is provided with the SSID and credential it needs to connect to the local network. (M-2.a) The device presents its credential to the network with the appropriate SSID. The device is assigned an IP address on the appropriate network.
Description	Demonstrate that a device can be onboarded using DPP and, once onboarded, the device can successfully connect to the appropriate network by using the credential that was provided to it during onboarding.
Associated Exercises	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, DE.AE-1, DE.CM-1

Exercise Field	Description
IoT Device(s) Used	Raspberry Pi
Policy Used	N/A
Preconditions	1. There are two DPP-capable devices available for use. 2. All devices have been configured to use Ipv4. 3. The gateway does not yet have any configuration settings pertaining to the IoT device being used in the test. 4. The device being onboarded does not have a MUD file (or, if it does have a MUD file, the MUD file will not interfere with the device's ability to communicate with other devices that are on the same micronet or with the device's inability to communicate with devices that are on different micronets). 5. In addition to the access point on the Micronets Gateway that is the correct network to which the device should connect, there is a second access point advertising an SSID of "incorrect network."
Procedure	1. Verify that the gateway for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 2. Power on the IoT device. 3. Wait a minute to verify that the device does not automatically connect to the network. 4. Put the IoT device into DPP onboarding mode by clicking the + button. This will cause it to display a QR code and begin listening for DPP messages. 5. Open the Micronets onboarding application on the mobile phone and click READY TO SCAN. 6. Position the mobile phone's camera to read the device's QR code. Do this in a timely manner because there is a 60-second countdown for the device to exit DPP onboarding mode. 7. Input additional device-specific information into the mobile onboarding application as requested (must be done within the same 60-second time limit): a) Assign the device to a Micronets class (e.g., Generic).

Exercise Field	Description
	b) Give the device a unique name (e.g., Device 1). 8. Click the ONBOARD button on the mobile application. This causes the onboarding application to send the device's bootstrapping information to the DPP configurator on the gateway via the operator's MSO portal and cloud infrastructure. 9. Wait. The following operations are being performed automatically in the operator's cloud infrastructure: a) The Micronets Manager receives the bootstrapping info. b) The Micronets Manager provisions the device on the gateway. c) The device is onboarded via DPP. d) The device connects to the network. 10. View the logs on the gateway to verify that: a) The DPP bootstrapping information was received at the DPP configurator. b) The authentication phase of DPP onboarding occurred for the device. (This is a three-way handshake—request, respond, confirm—between the configurator, which is in the gateway, and the device. The configurator initiates this exchange to authenticate the device and provide the device with a key to use to encrypt further communication. This three-way exchange occurs in the clear.) c) The configuration phase of DPP onboarding occurred for the device. (This is another three-way handshake—request, respond, result—between the configurator and the device. This is an encrypted exchange that the device initiates to learn the SSID of the correct network to which it should connect and its unique network credential.) 11. Verify that the device has been assigned an IP address on the correct network. 12. Repeat all the above steps (1-11) for a second device, but this time call the device Device 2 in step 7b. Note that the second device should be assigned to the same Micronets class as the first device (e.g., Generic). 13. At this point there should be two devices connected to the network, and they should be on the same micronet (micronet Generic). Verify

Exercise Field	Description
	that these two devices can send and receive messages to and from each other.
Demonstrated Results	**Micronets Gateway and Micronets Manager logs verifying onboarding:** **Device 1:** 1. DPP onboarding initiated: • Micronets Gateway: "DPPHandler.onboard_device: Issuing DPP onboarding commands for device" `2020-06-16 14:03:32,897 micronets-gw-service: INFO DPPHandler.onboard_device: Issuing DPP onboarding commands for device '463165abc19725aefffc39def13ce09b17167fba' in micronet 'generic…` `2020-06-16 14:03:32,898 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending:` `2020-06-16 14:03:32,899 micronets-gw-service: INFO {` `"DPPOnboardingStartedEvent": {` `"deviceId": "463165abc19725aefffc39def13ce09b17167fba",` `"macAddress": "00:C0:CA:97:D1:1F",` `"micronetId": "Generic",` `"reason": "DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"` `}` `}` • Micronets Manager: "DPPOnboardingStartedEvent"

Exercise Field	Description
	2020-06-16T18:03:32.923407831Z Gateway Message : {"body":{"DPPOnboardingStartedEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reaso n":"DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"}}} EventType : "DPPOnboardingStartedEvent" 2020-06-16T18:03:32.923417691Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:32.923424251Z Event to Post : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_ini t peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"} 2020-06-16T18:03:32.923432861Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:32.923483580Z OnBoarding PatchBody : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingStartedEvent","deviceId":"463165abc19725aefffc39def13ce09b1716 7fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=7 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=f16c6d6c61bb828f6225738072 f416bd5059f820ac3b06a9218b4a4414c54d7e neg_freq=2412\")"}} 2. DPP authorization success: • Micronets Gateway: "DPP-AUTH-SUCCESS"

Exercise Field	Description
	2020-06-16 14:03:32,921 micronets-gw-service: INFO DPPHandler.handle_hostapd_cli_event(DPP-AUTH-SUCCESS init=1) 2020-06-16 14:03:32,921 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending: 2020-06-16 14:03:32,921 micronets-gw-service: INFO { "DPPOnboardingProgressEvent": { "deviceId": "463165abc19725aefffc39def13ce09b17167fba", "macAddress": "00:C0:CA:97:D1:1F", "micronetId": "Generic", "reason": "DPP Progress (DPP-AUTH-SUCCESS init=1)" } } • Micronets Manager: "DPPOnboardingProgressEvent"/"DPP Progress (DPP-AUTH-SUCCESS init=1)" 2020-06-16T18:03:32.954959234Z Gateway Message : {"body":{"DPPOnboardingProgressEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}}} EventType : "DPPOnboardingProgressEvent" 2020-06-16T18:03:32.955713205Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:32.955759765Z Event to Post : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"} 2020-06-16T18:03:32.957158978Z 2020-06-16 18:03:32 ESC[34mdebugESC[39m [index.js]:

Exercise Field	Description
	`2020-06-16T18:03:32.957181208Z OnBoarding PatchBody : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingProgressEvent","deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}}` 3. DPP configuration sent: • Micronets Gateway: "DPP-CONF-SENT" `2020-06-16 14:03:33,338 micronets-gw-service: INFO DPPHandler.handle_hostapd_cli_event(DPP-CONF-SENT)` `2020-06-16 14:03:33,338 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending:` `2020-06-16 14:03:33,338 micronets-gw-service: INFO {` `"DPPOnboardingProgressEvent": {` `"deviceId": "463165abc19725aefffc39def13ce09b17167fba",` `"macAddress": "00:C0:CA:97:D1:1F",` `"micronetId": "Generic",` `"reason": "DPP Progress (DPP-CONF-SENT)"` `}` `}` • Micronets Manager: "DPPOnboardingProgressEvent"/"DPP Progress (DPP-CONF-SENT init=1)" `2020-06-16T18:03:33.363367674Z Gateway Message : {"body":{"DPPOnboardingProgressEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}}} EventType : "DPPOnboardingProgressEvent"` `2020-06-16T18:03:33.363573045Z 2020-06-16 18:03:33 ESC[34mdebugESC[39m [index.js]:`

Exercise Field	Description
	2020-06-16T18:03:33.363584045Z Event to Post : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"} 2020-06-16T18:03:33.363785005Z 2020-06-16 18:03:33 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:33.363794825Z OnBoarding PatchBody : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingProgressEvent","deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}} 4. DPP onboarding completed: • Micronets Gateway: "AP-STA-CONNECTED" 2020-06-16 14:03:36,851 micronets-gw-service: INFO DPPHandler.handle_hostapd_cli_event(AP-STA-CONNECTED 00:c0:ca:97:d1:1f) 2020-06-16 14:03:36,851 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending: 2020-06-16 14:03:36,851 micronets-gw-service: INFO { "DPPOnboardingCompleteEvent": { "deviceId": "463165abc19725aefffc39def13ce09b17167fba", "macAddress": "00:C0:CA:97:D1:1F", "micronetId": "Generic", "reason": "DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)" } } • Micronets Manager: "DPPOnboardingCompleteEvent"/"DPP Onboarding Complete (AP-STA-CONNECTED"

Exercise Field	Description
	2020-06-16T18:03:36.882393990Z Gateway Message : {"body":{"DPPOnboardingCompleteEvent":{"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"}}} EventType : "DPPOnboardingCompleteEvent" 2020-06-16T18:03:36.882403959Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:36.882409589Z Event to Post : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"} 2020-06-16T18:03:36.882415439Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:36.882466150Z OnBoarding PatchBody : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingCompleteEvent","deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"}} 2020-06-16T18:03:36.882475160Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]: 2020-06-16T18:03:36.882479660Z Hook Type: before Path: mm/v1/dpp Method: patch 2020-06-16T18:03:36.882486270Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:03:36.882490280Z 2020-06-16T18:03:36.882493840Z PATCH BEFORE HOOK DPP DATA : {"deviceId":"463165abc19725aefffc39def13ce09b17167fba","events":{"type":"DPPOnboardingCompleteEvent","deviceId":"463165abc19725aefffc39def13ce09b17167fba","macAddress":"00:C0:CA:97:D1:1F","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:97:d1:1f)"}} PARAMS : {} RequestUrl : undefined

Exercise Field	Description
	`2020-06-16T18:03:36.882500760Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]:` `2020-06-16T18:03:36.882505420Z  Hook Type: before Path: mm/v1/dpp  Method: get` `2020-06-16T18:03:36.883566612Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]:` `2020-06-16T18:03:36.883590111Z  Hook Type: after Path: mm/v1/dpp  Method: get` `2020-06-16T18:03:36.883834742Z 2020-06-16 18:03:36 ESC[32minfoESC[39m [index.js]: Hook.result.data : undefined` `2020-06-16T18:03:36.884259803Z 2020-06-16 18:03:36 ESC[34mdebugESC[39m [index.js]:` `2020-06-16T18:03:36.884279723Z` **Device 2:** 1. DPP onboarding initiated: • Micronets Gateway: “DPPHandler.onboard_device: Issuing DPP onboarding commands for device” `2020-06-16 14:04:08,309 micronets-gw-service: INFO DPPHandler.onboard_device: Issuing DPP onboarding commands for device '9f58599efce4680ee0c21efe0b98e27f8a7a8958' in micronet 'generic...` `2020-06-16 14:04:08,312 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending:` `2020-06-16 14:04:08,312 micronets-gw-service: INFO {` `"DPPOnboardingStartedEvent": {` `"deviceId": "9f58599efce4680ee0c21efe0b98e27f8a7a8958",` `"macAddress": "00:C0:CA:98:42:37",` `"micronetId": "Generic",`

Exercise Field	Description
	"reason": "DPP Started (issuing \"dpp_auth_init peer=8 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=3f95fbf121276caef1e8f468a6cd4904d9309a4cf7c4b30c490bc5f6c089d4e1 neg_freq=2412\")" } } • Micronets Manager: "DPPOnboardingStartedEvent" 2020-06-16T18:04:08.341179747Z Gateway Message : {"body":{"DPPOnboardingStartedEvent":{"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=8 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=3f95fbf121276caef1e8f468a6cd4904d9309a4cf7c4b30c490bc5f6c089d4e1 neg_freq=2412\")"}}} EventType : "DPPOnboardingStartedEvent" 2020-06-16T18:04:08.342059848Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:08.342085778Z Event to Post : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=8 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk psk=3f95fbf121276caef1e8f468a6cd4904d9309a4cf7c4b30c490bc5f6c089d4e1 neg_freq=2412\")"} 2020-06-16T18:04:08.343112830Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:08.343164050Z OnBoarding PatchBody : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","events":{"type":"DPPOnboardingStartedEvent","deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Started (issuing \"dpp_auth_init peer=8 ssid=6d6963726f6e6574732d6777 configurator=2 conf=sta-psk

Exercise Field	Description
	psk=3f95fbf121276caef1e8f468a6cd4904d9309a4cf7c4b30c490bc5f6c089d4e1 neg_freq=2412\")"}} 2. DPP authorization success: • Micronets Gateway: “DPP-AUTH-SUCCESS” 2020-06-16 14:04:08,332 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending: 2020-06-16 14:04:08,333 micronets-gw-service: INFO { "DPPOnboardingProgressEvent": { "deviceId": "9f58599efce4680ee0c21efe0b98e27f8a7a8958", "macAddress": "00:C0:CA:98:42:37", "micronetId": "Generic", "reason": "DPP Progress (DPP-AUTH-SUCCESS init=1)" } } • Micronets Manager: “DPPOnboardingProgressEvent”/“DPP Progress (DPP-AUTH-SUCCESS init=1)” 2020-06-16T18:04:08.363217003Z Gateway Message : {"body":{"DPPOnboardingProgressEvent":{"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}}} EventType : "DPPOnboardingProgressEvent" 2020-06-16T18:04:08.363596564Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:08.363637793Z Event to Post : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}

Exercise Field	Description
	`2020-06-16T18:04:08.363976154Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]:` `2020-06-16T18:04:08.363993024Z OnBoarding PatchBody : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","events":{"type":"DPPOnboardingProgressEvent","deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}}`  `2020-06-16T18:04:08.364503475Z 2020-06-16 18:04:08 ESC[32minfoESC[39m [index.js]:`  `2020-06-16T18:04:08.364537115Z Hook Type: before Path: mm/v1/dpp Method: patch`  `2020-06-16T18:04:08.364807675Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]:`  `2020-06-16T18:04:08.364855145Z`  `2020-06-16T18:04:08.364860535Z PATCH BEFORE HOOK DPP DATA : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","events":{"type":"DPPOnboardingProgressEvent","deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-AUTH-SUCCESS init=1)"}} PARAMS : {} RequestUrl : undefined`  3. DPP configuration sent:  • Micronets Gateway: “DPP-CONF-SENT”  `2020-06-16 14:04:08,743 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending:`  `2020-06-16 14:04:08,743 micronets-gw-service: INFO {`  `"DPPOnboardingProgressEvent": {`  `"deviceId": "9f58599efce4680ee0c21efe0b98e27f8a7a8958",`  `"macAddress": "00:C0:CA:98:42:37",`  `"micronetId": "Generic",`

Exercise Field	Description
	"reason": "DPP Progress (DPP-CONF-SENT)" } } • Micronets Manager: "DPPOnboardingProgressEvent"/"DPP Progress (DPP-CONF-SENT init=1)" 2020-06-16T18:04:08.770279846Z Gateway Message : {"body":{"DPPOnboardingProgressEvent":{"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}}} EventType : "DPPOnboardingProgressEvent" 2020-06-16T18:04:08.770606877Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:08.770621666Z Event to Post : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"} 2020-06-16T18:04:08.770899197Z 2020-06-16 18:04:08 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:08.770945437Z OnBoarding PatchBody : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","events":{"type":"DPPOnboardingProgressEvent","deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Progress (DPP-CONF-SENT)"}} 4. DPP onboarding completed: • Micronets Gateway: "AP-STA-CONNECTED" 2020-06-16 14:04:12,850 micronets-gw-service: INFO DPPHandler.send_dpp_onboard_event: sending: 2020-06-16 14:04:12,851 micronets-gw-service: INFO { "DPPOnboardingCompleteEvent": {

Exercise Field	Description
	"deviceId": "9f58599efce4680ee0c21efe0b98e27f8a7a8958", "macAddress": "00:C0:CA:98:42:37", "micronetId": "Generic", "reason": "DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:98:42:37)" } } • Micronets Manager: "DPPOnboardingCompleteEvent"/"DPP Onboarding Complete (AP-STA-CONNECTED" 2020-06-16T18:04:12.879141075Z Gateway Message : {"body":{"DPPOnboardingCompleteEvent":{"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:98:42:37)"}}} EventType : "DPPOnboardingCompleteEvent" 2020-06-16T18:04:12.879151105Z 2020-06-16 18:04:12 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:12.879156195Z Event to Post : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:98:42:37)"} 2020-06-16T18:04:12.879162795Z 2020-06-16 18:04:12 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:12.879167215Z OnBoarding PatchBody : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","events":{"type":"DPPOnboardingCompleteEvent","deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:98:42:37)"}} 2020-06-16T18:04:12.879174054Z 2020-06-16 18:04:12 ESC[32minfoESC[39m [index.js]:

Exercise Field	Description
	2020-06-16T18:04:12.879178314Z Hook Type: before Path: mm/v1/dpp Method: patch 2020-06-16T18:04:12.879182614Z 2020-06-16 18:04:12 ESC[34mdebugESC[39m [index.js]: 2020-06-16T18:04:12.879207595Z 2020-06-16T18:04:12.879212535Z PATCH BEFORE HOOK DPP DATA : {"deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","events":{"type":"DPPOnboardingCompleteEvent","deviceId":"9f58599efce4680ee0c21efe0b98e27f8a7a8958","macAddress":"00:C0:CA:98:42:37","micronetId":"Generic","reason":"DPP Onboarding Complete (AP-STA-CONNECTED 00:c0:ca:98:42:37)"}} PARAMS : {} RequestUrl : undefined --- **<u>Verify appropriate micronet created and devices added</u>:** { "_id": "5ee7bf78ab3e8358c185e759", "id": "subscriber-001", "name": "Subscriber 001", "ssid": "micronets-gw", "gatewayId": "micronets-gw", "micronets": [{ "name": "Generic", "class": "Generic", "micronet-subnet-id": "Generic", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL",

Exercise Field	Description
	"dhcp-zone": "10.135.1.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.1.0/24", "micronet-gateway-ip": "10.135.1.1", "connected-devices": [{ "device-mac": "00:C0:CA:97:D1:1F", "device-name": "Pi1-nm1", "device-id": "463165abc19725aefffc39def13ce09b17167fba", "device-openflow-port": "2", "device-ip": "10.135.1.2" }, { "device-mac": "00:C0:CA:98:42:37", "device-name": "Pi2-nm1", "device-id": "9f58599efce4680ee0c21efe0b98e27f8a7a8958", "device-openflow-port": "2", "device-ip": "10.135.1.3" }], "micronet-id": "2316794860" }], "createdAt": "2020-06-15T18:35:36.968Z", "updatedAt": "2020-06-16T18:04:06.636Z", "__v": 0

Exercise Field	Description
	}
	View flow rules: `Every 2.0s: sudo ovs-ofctl dump-flows brmn001 --names | /opt/micronets-gw/bin/format-ofctl-dump` `Tue Jun 16 15:23:00 2020` `table=0   priority=500 n_packets=0 dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0 actions=drop` `table=0   priority=500 n_packets=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop` `table=0   priority=500 n_packets=0      icmp icmp_code=1 actions=drop` `table=0   priority=450 n_packets=643     in_port=LOCAL actions=resubmit( 200)` `table=0   priority=400 n_packets=1218 in_port="wlp2s0.2486" actions=resubmit( 100)` `table=0   priority=400 n_packets=18     in_port=wlp2s0 actions=resubmit( 100)` `table=0   priority=0   n_packets=2 actions=output:diagout1` `table=100 priority=910 n_packets=0      ct_state=+rel+trk udp actions=LOCAL` `table=100 priority=910 n_packets=1      ct_state=+est+trk udp actions=LOCAL` `table=100 priority=910 n_packets=490    ct_state=-trk udp actions=ct(table=100)` `table=100 priority=905 n_packets=0      ct_state=+est+trk tcp actions=LOCAL` `table=100 priority=905 n_packets=0      ct_state=+rel+trk tcp actions=LOCAL` `table=100 priority=905 n_packets=0      ct_state=-trk tcp actions=ct(table=100)`

Exercise Field	Description
	table=100 priority=900 n_packets=18 dl_type=0x888e actions=resubmit(120) table=100 priority=850 n_packets=137 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=850 n_packets=137 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 udp in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=2 udp in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=352 arp in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f actions=resubmit(120) table=100 priority=815 n_packets=362 arp in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.1.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=104.237.132.42 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f nw_dst=198.71.233.87 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.1.1 actions=resubmit(120)

Exercise Field	Description
	`table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 nw_dst=104.237.132.42 actions=resubmit( 120)` `table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 nw_dst=198.71.233.87 actions=resubmit( 120)` `table=100 priority=805 n_packets=103 in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f actions=output:diagout1` `table=100 priority=805 n_packets=124 in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 actions=output:diagout1` `table=100 priority=800 n_packets=0 in_port="wlp2s0.2486" dl_src=00:c0:ca:97:d1:1f actions=resubmit( 110)` `table=100 priority=800 n_packets=0 in_port="wlp2s0.2486" dl_src=00:c0:ca:98:42:37 actions=resubmit( 110)` `table=100 priority=460 n_packets=0 in_port=wlp2s0 dl_type=0x888e actions=resubmit( 120)` `table=100 priority=0 n_packets=0 actions=output:diagout1` **Device communication:** `pi@pi-2:~ $ ssh pi@10.135.1.2` `pi@10.135.1.2's password:` `Last login: Tue Jun 16 10:33:01 2020 from 192.168.30.181` `pi@pi-1:~ $` `pi@pi-1:~ $ ssh pi@10.135.1.3` `pi@10.135.1.3's password:` `Last login: Tue Jun 16 09:32:35 2020 from 192.168.30.181` `pi@pi-2:~ $`

Exercise Field	Description

622 ### *4.2.2.2 Exercise MnMUD-2*

623 **Table 4-13: Exercise MnMUD-2**

Exercise Field	Description
Parent Capability	(M-3) Device micronet classification–Upon connection to the network, each device is placed into its intended micronet class.
Subrequirement(s) of Parent Capability to Be Demonstrated	(M-3.a) The micronet class of each device can be provided as part of the bootstrapping information. The user specifies the device micronets class by using the onboarding application on the mobile phone (after scanning the QR code). (M-3.b) Devices that are in the same micronet class can communicate with each other (assuming this is not contradicted by the devices' MUD files). (M-3.c) Devices that are in different micronet classes cannot communicate with each other (assuming this is not contradicted by the devices' MUD files).
Description	Demonstrate that when each device is onboarded, the micronet class to which the device should be assigned can be provided so that when the device connects to the network, it will be located on the specified micronet. Also show that devices that are on the same micronet can communicate with each other, whereas devices that are on different micronets cannot (assuming that the devices do not have MUD files or, if they do have MUD files, the MUD files do not interfere with this behavior.)
Associated Exercises	MnMUD-1

Exercise Field	Description
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, DE.AE-1, DE.CM-1
IoT Device(s) Used	Raspberry Pi
Policy Used	N/A
Preconditions	All the same preconditions as Exercise MnMUD-1, except that for this test, three DPP-capable devices are available for use instead of just two.
Procedure	1. Run Exercise MnMUD-1. 2. At this point, there should be two devices connected to the correct network (Device 1 and Device 2), and they should be on the same micronet (Medical). 3. Perform steps 1-12 of Exercise MnMUD-1 for a third device, but this time assign the device the micronet class Personal in step 7a, and call the device Device 3 in step 7b. 4. Verify that Device 1 and Device 2 (which are both on Medical micronet class) can send and receive messages to and from each other. 5. Verify that neither Device 1 nor Device 2 can send or receive messages to or from Device 3 (which is on Personal micronet class).
Demonstrated Results	{ "_id": "5ee7bf78ab3e8358c185e759", "id": "subscriber-001", "name": "Subscriber 001", "ssid": "micronets-gw", "gatewayId": "micronets-gw", "micronets": [{

Exercise Field	Description
	"name": "Medical", "class": "Medical", "micronet-subnet-id": "Medical", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL", "dhcp-zone": "10.135.4.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.4.0/24", "micronet-gateway-ip": "10.135.4.1", "connected-devices": [{ "device-mac": "00:C0:CA:98:42:37", "device-name": "Pi1-nm2", "device-id": "9f58599efce4680ee0c21efe0b98e27f8a7a8958", "device-openflow-port": "2", "device-ip": "10.135.4.2" }, { "device-mac": "00:C0:CA:97:D1:1F", "device-name": "Pi2-nm2", "device-id": "463165abc19725aefffc39def13ce09b17167fba", "device-openflow-port": "2", "device-ip": "10.135.4.3" }],

Exercise Field	Description
	"micronet-id": "1923653520" }, { "name": "Personal", "class": "Personal", "micronet-subnet-id": "Personal", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL", "dhcp-zone": "10.135.5.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.5.0/24", "micronet-gateway-ip": "10.135.5.1", "connected-devices": [{ "device-mac": "00:C0:CA:98:42:2D", "device-name": "Pi3-nm2", "device-id": "da34c7219c2c97f0e2c2838e66c725d137f3c097", "device-openflow-port": "2", "device-ip": "10.135.5.2" }], "micronet-id": "2340317076" }], "createdAt": "2020-06-15T18:35:36.968Z",

Exercise Field	Description
	`"updatedAt": "2020-06-17T20:55:29.541Z",` `"__v": 0` `}` **<u>Devices' communication</u>:** `pi@pi-2:~ $ ssh pi@10.135.4.3` `pi@10.135.4.3's password:` `Last login: Wed Jun 17 12:07:11 2020 from 192.168.30.181` `pi@pi-1:~ $` `pi@pi-1:~ $ ssh pi@10.135.4.2` `pi@10.135.4.2's password:` `Last login: Wed Jun 17 10:30:58 2020 from 192.168.30.181` `pi@pi-2:~ $` `pi@pi-2:~ $ ssh pi@10.135.5.2` `ssh: connect to host 10.135.5.2 port 22: Connection timed out` `pi@pi-3:~ $ ssh pi@10.135.4.2` `ssh: connect to host 10.135.4.2 port 22: Connection timed out` `pi@pi-3:~ $ ssh pi@10.135.4.3` `ssh: connect to host 10.135.4.3 port 22: Connection timed out` **<u>Flow rules</u>:** `Every 2.0s: sudo ovs-ofctl dump-flows brmn001 --names | /opt/micronets-gw/bin/format-ofctl-dump` `Wed Jun 17 16:57:42 2020`

Exercise Field	Description
	table=0 priority=500 n_packets=0 dl_dst=01:80:c2:00:00:00/ff:ff:ff:ff:ff:f0 actions=drop table=0 priority=500 n_packets=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop table=0 priority=500 n_packets=0 icmp icmp_code=1 actions=drop table=0 priority=450 n_packets=28 in_port=LOCAL actions=resubmit(200) table=0 priority=400 n_packets=20 in_port="wlp2s0.2844" actions=resubmit(100) table=0 priority=400 n_packets=2 in_port=wlp2s0 actions=resubmit(100) table=0 priority=400 n_packets=51 in_port="wlp2s0.2395" actions=resubmit(100) table=0 priority=0 n_packets=0 actions=output:diagout1 table=100 priority=910 n_packets=0 ct_state=+est+trk udp actions=LOCAL table=100 priority=910 n_packets=0 ct_state=+rel+trk udp actions=LOCAL table=100 priority=910 n_packets=26 ct_state=-trk udp actions=ct(table=100) table=100 priority=905 n_packets=0 ct_state=+est+trk tcp actions=LOCAL table=100 priority=905 n_packets=0 ct_state=+rel+trk tcp actions=LOCAL table=100 priority=905 n_packets=0 ct_state=-trk tcp actions=ct(table=100) table=100 priority=900 n_packets=2 dl_type=0x888e actions=resubmit(120) table=100 priority=850 n_packets=2 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.4.1 actions=resubmit(120)

Exercise Field	Description
	table=100 priority=850 n_packets=2 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.4.1 actions=resubmit(120) table=100 priority=850 n_packets=6 ip in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d nw_dst=10.135.5.1 actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 dl_type=0x888e actions=resubmit(120) table=100 priority=815 n_packets=0 udp in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=0 udp in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=16 arp in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d actions=resubmit(120) table=100 priority=815 n_packets=2 udp in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d tp_dst=67 actions=resubmit(120) table=100 priority=815 n_packets=8 arp in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f actions=resubmit(120) table=100 priority=815 n_packets=8 arp in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d nw_dst=10.135.5.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d nw_dst=52.89.85.207 actions=resubmit(120)

Exercise Field	Description
	table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d nw_dst=54.191.221.118 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d nw_dst=54.201.49.86 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f nw_dst=10.135.4.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f nw_dst=104.237.132.42 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f nw_dst=198.71.233.87 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 nw_dst=10.135.4.1 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 nw_dst=104.237.132.42 actions=resubmit(120) table=100 priority=810 n_packets=0 ip in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 nw_dst=198.71.233.87 actions=resubmit(120) table=100 priority=805 n_packets=0 in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f actions=output:diagout1 table=100 priority=805 n_packets=0 in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 actions=output:diagout1 table=100 priority=805 n_packets=27 in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d actions=output:diagout1 table=100 priority=800 n_packets=0 in_port="wlp2s0.2395" dl_src=00:c0:ca:98:42:2d actions=resubmit(110) table=100 priority=800 n_packets=0 in_port="wlp2s0.2844" dl_src=00:c0:ca:97:d1:1f actions=resubmit(110)

Exercise Field	Description
	table=100 priority=800 n_packets=0 in_port="wlp2s0.2844" dl_src=00:c0:ca:98:42:37 actions=resubmit(110) table=100 priority=460 n_packets=0 in_port=wlp2s0 dl_type=0x888e actions=resubmit(120) table=100 priority=0 n_packets=0 actions=output:diagout1

4.2.2.3 Exercise MnMUD-3

Table 4-14: Exercise MnMUD-3

Exercise Field	Description
Parent Capability	(M-4) Each device that is onboarded using DPP is assigned a unique credential.
Subrequirement(s) of Parent Capability to Be Demonstrated	(M-4.a) The Micronets Gateway can be configured to disconnect a device that has been onboarded using DPP. The other devices remain connected.
Description	Demonstrate that if multiple devices have been onboarded, the gateway can be configured to revoke the credential of one of the devices, causing it to be disconnected. But the other devices, which have their own unique credentials, will remain connected.
Associated Exercises	MnMUD-1
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, DE.AE-1, DE.CM-1

Exercise Field	Description
IoT Device(s) Used	Raspberry Pi
Policy Used	N/A
Preconditions	All the same preconditions as Exercise MnMUD-1, except that for this test, three DPP-capable devices are available for use instead of just two.
Procedure	1. Run Exercise MnMUD-1. 2. At this point, there should be two devices connected to the correctnetwork (Device 1 and Device 2), and they should be on the same Micronet (CLASS 1). 3. Perform steps 1-12 of Exercise MnMUD-1 for a third device, assigning the device the same Micronet class (CLASS 1) in step 7a as the other two devices, and call the device Device 3 in step 7b. 4. Verify that Device 1, Device 2, and Device 3 (which are all on Micronet CLASS 1) can send and receive messages to and from one another. 5. Configure the gateway to disconnect Device 2. 6. Verify that Device 2 cannot send messages to or receive messages from Device 1 or Device 3. 7. Verify that Device 1 and Device 3 can send messages to and from each other.
Demonstrated Results	**<u>Get micronets before deleting single device</u>:** { "_id": "5ee7bf78ab3e8358c185e759", "id": "subscriber-001", "name": "Subscriber 001", "ssid": "micronets-gw", "gatewayId": "micronets-gw",

Exercise Field	Description
	"micronets": [{ "name": "Medical", "class": "Medical", "micronet-subnet-id": "Medical", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL", "dhcp-zone": "10.135.2.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.2.0/24", "micronet-gateway-ip": "10.135.2.1", "connected-devices": [{ "device-mac": "00:C0:CA:97:D1:1F", "device-name": "Pi1-nm3", "device-id": "463165abc19725aefffc39def13ce09b17167fba", "device-openflow-port": "2", "device-ip": "10.135.2.2" }, { "device-mac": "00:C0:CA:98:42:37", "device-name": "Pi2-nm3", "device-id": "9f58599efce4680ee0c21efe0b98e27f8a7a8958", "device-openflow-port": "2", "device-ip": "10.135.2.3"

Exercise Field	Description
	}], "micronet-id": "2030552386" }, { "name": "Personal", "class": "Personal", "micronet-subnet-id": "Personal", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL", "dhcp-zone": "10.135.3.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.3.0/24", "micronet-gateway-ip": "10.135.3.1", "connected-devices": [{ "device-mac": "00:C0:CA:98:42:2D", "device-name": "Pi3-nm3", "device-id": "da34c7219c2c97f0e2c2838e66c725d137f3c097", "device-openflow-port": "2", "device-ip": "10.135.3.2" }], "micronet-id": "2136369149" }

Exercise Field	Description
	], "createdAt": "2020-06-15T18:35:36.968Z", "updatedAt": "2020-06-17T19:57:18.274Z", "__v": 0 } --- **After deleting "pi3-nm3":** **Command:** $ curl -X DELETE https://{{micronets-manager-linode-ip}}/sub/{{subscriberId}}/api/mm/v1/subscriber/{{subscriberId}}/micronets/9f58599efce4680ee0c21efe0b98e27f8a7a8958a8958 **Results:** { "_id": "5ee7bf78ab3e8358c185e759", "id": "subscriber-001", "name": "Subscriber 001", "ssid": "micronets-gw", "gatewayId": "micronets-gw", "micronets": [{ "name": "Medical", "class": "Medical", "micronet-subnet-id": "Medical", "trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL",

Exercise Field	Description
	"dhcp-zone": "10.135.2.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.2.0/24", "micronet-gateway-ip": "10.135.2.1", "connected-devices": [{ "device-mac": "00:C0:CA:97:D1:1F", "device-name": "Pi1-nm3", "device-id": "463165abc19725aefffc39def13ce09b17167fba", "device-openflow-port": "2", "device-ip": "10.135.2.2" }, { "device-mac": "00:C0:CA:98:42:37", "device-name": "Pi2-nm3", "device-id": "9f58599efce4680ee0c21efe0b98e27f8a7a8958", "device-openflow-port": "2", "device-ip": "10.135.2.3" }], "micronet-id": "2030552386" }, { "name": "Personal", "class": "Personal", "micronet-subnet-id": "Personal",

Exercise Field	Description
	"trunk-gateway-port": "2", "trunk-gateway-ip": "10.36.32.124", "dhcp-server-port": "LOCAL", "dhcp-zone": "10.135.3.0/24", "ovs-bridge-name": "brmn001", "ovs-manager-ip": "10.36.32.124", "micronet-subnet": "10.135.3.0/24", "micronet-gateway-ip": "10.135.3.1", "connected-devices": [], "micronet-id": "2136369149" }], "createdAt": "2020-06-15T18:35:36.968Z", "updatedAt": "2020-06-17T20:34:15.504Z", "__v": 0 } **Confirming device removal from network:** **Wlan0 not displaying IP address assignment:** pi@pi-3:~ $ ifconfig eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 inet 192.168.30.137 netmask 255.255.255.0 broadcast 192.168.30.255 inet6 fe80::7d50:b23c:eb1f:99dd prefixlen 64 scopeid 0x20<link> ether b8:27:eb:9c:86:af txqueuelen 1000 (Ethernet) RX packets 3584 bytes 301107 (294.0 KiB) RX errors 0 dropped 0 overruns 0 frame 0

Exercise Field	Description
	TX packets 2593 bytes 1964711 (1.8 MiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 inet 127.0.0.1 netmask 255.0.0.0 inet6 ::1 prefixlen 128 scopeid 0x10<host> loop txqueuelen 1000 (Local Loopback) RX packets 4345 bytes 377756 (368.9 KiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 4345 bytes 377756 (368.9 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500 ether 00:c0:ca:98:42:2d txqueuelen 1000 (Ethernet) RX packets 232 bytes 33186 (32.4 KiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 391 bytes 49813 (48.6 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 **Device attempting to communicate to devices on Micronets Gateway:** pi@pi-3:~ $ ssh pi@10.135.2.2 ssh: connect to host 10.135.2.2 port 22: Network is unreachable pi@pi-3:~ $ ssh pi@10.135.2.3 ssh: connect to host 10.135.2.3 port 22: Network is unreachable **Device still has network psk but psk is now invalid:**

Exercise Field	Description
	pi@pi-3:~ $ cat /etc/wpa_supplicant/wpa_supplicant.conf ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev update_config=1 pmf=2 dpp_config_processing=2 network={ ssid="micronets-gw" psk=b10b953e1faef3c4f8c1381533877291b2ec20568fd0b49e19738de690dbf590 key_mgmt=WPA-PSK WPA-PSK-SHA256 ieee80211w=1 }

5 Build 4

Build 4 uses software developed at the NIST Advanced Networking Technologies laboratory. This software provides support for MUD and is intended to serve as a working prototype of the MUD RFC to demonstrate feasibility and scalability.

5.1 Evaluation of MUD-Related Capabilities

The functional evaluation that was conducted to verify that Build 4 conforms to the MUD specification was based on the Build 4-specific requirements listed in Table 5-1.

5.1.1 Requirements

Table 5-1: MUD Use Case Functional Requirements

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-1	The IoT DDoS example implementation shall include a			IoT-1-v4, IoT-11-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled **IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL).**			
CR-1.a		Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one **MUD URL, in https scheme, within the DHCP transaction.**		IoT-1-v4, IoT-11-v4
CR-1.a.1			The DHCP server shall be able to receive **DHCPv4 DISCOVER and REQUEST with IANA code 161** (OPTION_MUD_URL_V4) from the MUD-enabled IoT device.	IoT-1-v4, IoT-11-v4
CR-2	The IoT DDoS example implementation shall include the capability for the extracted MUD URL **to be provided to a MUD manager.**			IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-2.a		The DHCP server shall **assign an IP address lease** to the MUD-enabled IoT device.		IoT-1-v4
CR-2.a.1			The MUD-enabled IoT device shall **receive the IP address.**	IoT-1-v4
CR-2.b		**The MUD manager** shall receive the DHCP message and **extract the MUD URL.**		IoT-1-v4
CR-2.b.1			**The MUD manager shall receive the MUD URL.**	IoT-1-v4
CR-3	The IoT DDoS example implementation shall include a **MUD manager that can request a MUD file and signature from a MUD file server.**			IoT-1-v4
CR-3.a		The MUD manager shall use the GET method (RFC 7231) to **request MUD and signature files** (per RFC 7230) from the MUD file server and can **validate the MUD file server's TLS certificate** by using the rules in RFC 2818.		IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-3.a.1			**The MUD file server shall receive the https request from the MUD manager.**	IoT-1-v4
CR-3.b		**The MUD manager** shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it **cannot validate the MUD file server's TLS certificate** by using the rules in RFC 2818.		IoT-2-v4
CR-3.b.1			**The MUD manager shall drop the connection** to the MUD file server.	IoT-2-v4
CR-3.b.2			**The MUD manager shall send locally defined policy to the router or switch** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-2-v4
CR-4	The IoT DDoS example implementation shall include a **MUD file server that can**			IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	serve a MUD file and signature to the MUD manager.			
CR-4.a		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the **certificate had not expired.**		IoT-1-v4
CR-4.b		**The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file** was valid at the time of signing, i.e., the **certificate had already expired when it was used to sign the MUD file.**		IoT-3-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-4.b.1			The MUD manager shall cease to process the MUD file.	IoT-3-v4
CR-4.b.2			The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-3-v4
CR-5	The IoT DDoS example implementation shall include a **MUD manager** that **can translate local network configurations based on the MUD file.**			IoT-1-v4
CR-5.a		**The MUD manager shall successfully validate the signature of the MUD file.**		IoT-1-v4
CR-5.a.1			The MUD manager, after validation of the MUD file signature, shall **check for an existing MUD file, and translate abstractions in the MUD file to router or switch configurations**.	IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-5.a.2			The MUD manager shall **cache** this newly received MUD file.	IoT-10-v4
CR-5.b		The MUD manager shall attempt to validate the signature of the **MUD file,** but the **signature validation fails** (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason).		IoT-4-v4
CR-5.b.1			**The MUD manager shall cease processing the MUD file.**	IoT-4-v4
CR-5.b.2			**The MUD manager shall send locally defined policy to the router or switch** that handles whether to allow or block traffic to and from the MUD-enabled IoT device.	IoT-4-v4
CR-6	The IoT DDoS example implementation shall include a			IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	MUD manager that can configure the MUD PEP, i.e., the router or switch nearest the MUD-enabled IoT device that emitted the URL.			
CR-6.a		**The MUD manager shall install a router configuration** on the router or switch nearest the MUD-enabled IoT device that emitted the URL.		IoT-1-v4
CR-6.a.1			**The router or switch shall have been configured to enforce the route filter sent by the MUD manager.**	IoT-1-v4
CR-7	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file.**			IoT-5-v4
CR-7.a		The MUD-enabled IoT device shall attempt to **initiate outbound traffic to approved internet services.**		IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-7.a.1			The router or switch shall receive the attempt and shall **allow it to pass** based on the filters from the MUD file.	IoT-5-v4
CR-7.b		An approved **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4
CR-7.b.1			The router or switch shall receive the attempt and shall **allow it to pass** based on the filters from the MUD file.	IoT-5-v4
CR-8	The IoT DDoS example implementation shall **deny communications from a MUD-enabled IoT device to unapproved internet services** (i.e., services that are denied by virtue of not being explicitly approved).			IoT-5-v4
CR-8.a		The MUD-enabled IoT device shall **attempt to initiate outbound traffic to unapproved** (implicitly denied) **internet services.**		IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-8.a.1			**The router or switch shall receive the attempt and** shall **deny it** based on the filters from the MUD file.	IoT-5-v4
CR-8.b		**An unapproved** (implicitly denied) **internet service shall attempt to initiate a connection to the MUD-enabled IoT device.**		IoT-5-v4
CR-8.b.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4
CR-8.c		The MUD-enabled IoT device shall initiate communications to an internet service that is **approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device.**		IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-8.c.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4
CR-8.d		An internet service shall initiate communications to a MUD-enabled device that is **approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service.**		IoT-5-v4
CR-8.d.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-5-v4
CR-9	The IoT DDoS example implementation shall **allow the MUD-enabled IoT device to communicate laterally with devices that are approved** in the MUD file.			IoT-6-v4
CR-9.a		The MUD-enabled IoT device shall **attempt**		IoT-6-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		to initiate lateral traffic to approved devices.		
CR-9.a.1			**The router or switch shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4
CR-9.b		An approved device **shall attempt to initiate a lateral connection to the MUD-enabled IoT device.**		IoT-6-v4
CR-9.b.1			**The router or switch shall receive the attempt and shall allow it to pass** based on the filters from the MUD file.	IoT-6-v4
CR-10	The IoT DDoS example implementation shall **deny lateral communications from a MUD-enabled IoT device to devices that are not approved** in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).			IoT-6-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-10.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to unapproved** (implicitly denied) **devices.**		IoT-6-v4
CR-10.a.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4
CR-10.b		**An unapproved** (implicitly denied) **device shall attempt to initiate a lateral connection** to the MUD-enabled IoT device.		IoT-6-v4
CR-10.b.1			**The router or switch shall receive the attempt and shall deny it** based on the filters from the MUD file.	IoT-6-v4
CR-11	If the IoT DDoS example implementation is such that its DHCP server does not act as a MUD manager and it forwards a MUD URL to a MUD manager, **the DHCP server must notify the MUD manager of any corresponding change to the DHCP state** of			No test needed because the DHCP server does not forward the MUD URL to the

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	the MUD-enabled IoT device, and the MUD manager should **remove the implemented policy configuration in the router/switch pertaining to that MUD-enabled IoT device.**			MUD manager, as intended.
CR-11.a		The MUD-enabled IoT **device shall explicitly release the IP address lease** (i.e., it sends a DHCP release message to the DHCP server).		N/A
CR-11.a.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has been released.**	N/A
CR-11.a.2			**The MUD manager should remove all policies** associated with the disconnected IoT device that had been configured on the MUD PEP router/switch.	N/A
CR-11.b		The MUD-enabled IoT **device's IP address lease shall expire.**		N/A

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-11.b.1			**The DHCP server shall notify the MUD manager that the device's IP address lease has expired.**	N/A
CR-11.b.2			**The MUD manager should remove all policies** associated with the affected IoT device that had been configured on the MUD PEP router/switch.	N/A
CR-12	The IoT DDoS example implementation shall include a **MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed** for the MUD file indicated by the MUD URL**. The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.**			IoT-10-v4
CR-12.a		The MUD manager shall check if the file associated with the **MUD URL is present in its cache** and shall determine that it is.		IoT-10-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
CR-12.a.1			The MUD manager shall **check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager shall apply the contents of the cached MUD file.	IoT-10-v4
CR-12.a.2			The MUD manager **shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file.** If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.	IoT-10-v4
CR-13	The IoT DDoS example implementation shall ensure that for each rule in a MUD file			IoT-9-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	that pertains to an external domain, the MUD PEP router/switch will get configured with **all possible instantiations of that rule,** insofar as **each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the MUD PEP router/switch.**			
CR-13.a		The MUD file for a device shall contain a rule involving a **domain that can resolve to multiple IP addresses** when queried by the MUD PEP router/switch. **Flow rules for permitting access to each of those IP addresses will be inserted into the MUD PEP router/switch** for the device in question, and the device will be permitted to communicate with all of those IP addresses.		IoT-9-v4
CR-13.a.1			IPv4 addressing is used on the network.	IoT-9-v4

5.1.2 Test Cases

This section contains the test cases that were used to verify that Build 4 met the requirements listed in Table 5-1.

The test setup consists of five Raspberry Pis. Two of these are designated as having MUD Uniform Resource Identifiers (URIs) *sensor.nist.local* and one is designated *otherman.nist.local*. MUD files for "sensor" and "otherman" were generated using mudmaker. The Software Defined Network (SDN) enabled wireless router/NAT maps these fake hosts to test servers that are on the public side of the NAT. They are given fake 203.0.113.x addresses for name resolution. One of the Raspberry Pis is designated as a controller, and the last Raspberry Pi is designated as a host on the "local network."

The SDN switch is an unmodified Northbound Networks wireless SDN switch.

The controller host address and the DNS/DHCP host address are configured statically in the SDN controller by using the standard URIs for these entities. The controller URIs for the devices are likewise configured. dhclient is used to issue DHCP requests with MUD URLs embedded for Raspberry Pis 1, 2, and 3. The MUD URIs for 1 and 2 are identical and set to *https://sensor.nist.local/nistmud1*, while the MUD URI for Pi 3 is set to *https://otherman.nist.local/nistmud2*.

The controller host maps the fake host names in these URIs to 127.0.0.1 and runs a manufacturer https server. The server logs access to verify if file caching is properly working on the MUD manager.

Before the tests are conducted, the MUD files are signed using the NCCoE-supplied DigiCert key, and the trusted certificate is installed in the Java virtual machine trust store.

Accessibility testing is done using simple scripts and command line utilities that test whether permissible access works and whether forbidden access is blocked by the MUD-enabled SDN switch. The MUD files have access control entries that enable testing interactions with the hosts and web servers.

5.1.2.1 Test Case IoT-1-v4

Table 5-2: Test Case IoT-1-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL). (CR-2) The IoT DDoS example implementation shall include the capability for the MUD URL to be provided to a MUD manager.

Test Case Field	Description
	(CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server. (CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager. (CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file. (CR-6) The IoT DDoS example implementation shall include a MUD manager that can configure the router or switch nearest the MUD-enabled IoT device that emitted the URL.
Testable Requirements	(CR-1.a) Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one MUD URL, in https scheme, within the DHCP transaction. (CR-1.a.1) The DHCP server shall be able to receive DHCPv4 DISCOVER and REQUEST with IANA code 161 (OPTION_MUD_URL_V4) from the MUD-enabled IoT device. (CR-2.a) The DHCP server shall assign an IP address lease to the MUD-enabled IoT device. (CR-2.a.1) The MUD-enabled IoT device shall receive the IP address. (CR-2.b) The MUD manager shall receive the DHCP message and extract the MUD URL. (CR-2.b.1) The MUD manager shall receive the MUD URL. (CR-3.a) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server and can validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.a.1) The MUD file server shall receive the https request from the MUD manager. (CR-4.a) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file (signed using DER-encoded CMS [RFC 5652]) was valid at the time of signing, i.e., the certificate had not expired. (CR-5.a) The MUD manager shall successfully validate the signature of the MUD file.

Test Case Field	Description
	(CR-5.a.1) The MUD manager, after validation of the MUD file signature, shall check for an existing MUD file and translate abstractions in the MUD file to router or switch configurations. (CR-6.a) The MUD manager shall install a router configuration on the router or switch nearest the MUD-enabled IoT device that emitted the URL. (CR-6.a.1) The router or switch shall have been configured to enforce the route filter sent by the MUD manager.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file, assuming the MUD file has a valid signature and is served from a MUD file server that has a valid TLS certificate
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.PT-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The device's MUD file has a valid signature that was signed by a certificate that had not yet expired, and it is being hosted on a MUD file server that has a valid TLS certificate. 4. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 5. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 5.1.3.

Test Case Field	Description
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. Also verify that the MUD file of the IoT device to be used is not currently cached at the MUD manager. 1. Power on the IoT device and connect it to the test network. 2. On the IoT device, using the dhclient application with appropriate configuration file, manually send a DHCPv4 message containing the device's MUD URL (IANA code 161). 3. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 4. The MUD manager snoops the DHCP request through the switch and extracts the MUD URL from the DHCP request. 5. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, requests and receives the MUD file and signature from the MUD file server, validates the MUD file's signature, and translates the MUD file's contents into appropriate route filtering rules. It then installs these rules onto the MUD PEP for the IoT device in question so that this router/switch is now configured to enforce the policies specified in the MUD file. 6. The DHCP server offers an IP address lease to the newly connected IoT device. 7. The IoT device requests this IP address lease, which the DHCP server acknowledges.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. Flow rules on the switch are updated to reflect MUD filtering rules. The flow rules in the MUD flow rules table should reflect the ACLs in the MUD file.
Actual Results	**<u>Flow rules on router/switch</u>:** As seen below, tables zero and one classify the packets based on source and destination address, and tables two and three implement the MUD

Test Case Field	Description
	rules filtering. Tables four and five are pass and drop tables respectively. Additionally, to simplify, this test is successful when flows other than the default flows are viewed on the MUD PEP router/switch. `OFPST_FLOW reply (OF1.3) (xid=0x2):` ` cookie=0x995ac, duration=38.664s, table=0, n_packets=12, n_bytes=996, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_src=00:13:ef:20:1d:14 actions=write_metadata:0x100300300000000/0x7ffffff00000000,goto_table:1` ` cookie=0x995ac, duration=38.148s, table=0, n_packets=12, n_bytes=996, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_src=00:13:ef:70:47:66 actions=write_metadata:0x100300300000000/0x7ffffff00000000,goto_table:1` ` cookie=0x995ac, duration=37.655s, table=0, n_packets=13, n_bytes=1081, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_src=74:da:38:56:10:66 actions=write_metadata:0x100300300000000/0x7ffffff00000000,goto_table:1` ` cookie=0x995ac, duration=37.149s, table=0, n_packets=16, n_bytes=1324, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_src=b8:27:eb:ac:45:76 actions=write_metadata:0x300300000000/0x7ffffff00000000,goto_table:1` ` cookie=0x995ac, duration=33.630s, table=0, n_packets=58, n_bytes=4806, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_src=70:b3:d5:6c:db:92 actions=write_metadata:0x300300000000/0x7ffffff00000000,goto_table:1` ` cookie=0x995ac, duration=23.550s, table=0, n_packets=8, n_bytes=664, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_src=b8:27:eb:3d:65:78 actions=write_metadata:0x400500000000/0x7ffffff00000000,goto_table:1` ` cookie=0xca8bf, duration=82.206s, table=0, n_packets=25, n_bytes=2073, priority=31,ip actions=CONTROLLER:65535,write_metadata:0x200200000000/0xffffff00000000` ` cookie=0xf6736, duration=88.641s, table=0, n_packets=272, n_bytes=20928, priority=30 actions=write_metadata:0xf6736,goto_table:1` ` cookie=0xe809d, duration=38.641s, table=1, n_packets=60, n_bytes=4976, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_dst=70:b3:d5:6c:db:92 actions=write_metadata:0x3003/0x7ffffff,goto_table:2`

Test Case Field	Description
	cookie=0xe809d, duration=33.105s, table=1, n_packets=10, n_bytes=826, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_dst=00:13:ef:20:1d:14 actions=write_metadata:0x1003003/0x7ffffff,goto_table:2 cookie=0xe809d, duration=32.411s, table=1, n_packets=10, n_bytes=826, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_dst=00:13:ef:70:47:66 actions=write_metadata:0x1003003/0x7ffffff,goto_table:2 cookie=0xe809d, duration=31.916s, table=1, n_packets=12, n_bytes=996, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_dst=74:da:38:56:10:66 actions=write_metadata:0x1003003/0x7ffffff,goto_table:2 cookie=0xe809d, duration=31.417s, table=1, n_packets=15, n_bytes=1239, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_dst=b8:27:eb:ac:45:76 actions=write_metadata:0x3003/0x7ffffff,goto_table:2 cookie=0xe809d, duration=18.337s, table=1, n_packets=7, n_bytes=583, idle_timeout=120, hard_timeout=240, priority=40,ip,dl_dst=b8:27:eb:3d:65:78 actions=write_metadata:0x4005/0x7ffffff,goto_table:2 cookie=0xca8bf, duration=81.689s, table=1, n_packets=11, n_bytes=1324, priority=31,ip actions=CONTROLLER:65535,write_metadata:0x2002/0xffffff cookie=0xf6736, duration=88.335s, table=1, n_packets=272, n_bytes=20928, priority=30 actions=write_metadata:0xf6736,goto_table:2 cookie=0xea237, duration=78.043s, table=2, n_packets=3, n_bytes=1050, priority=55,udp,tp_src=68,tp_dst=67 actions=CONTROLLER:65535,goto_table:4 cookie=0x99f4d, duration=78.043s, table=2, n_packets=3, n_bytes=1031, priority=55,udp,tp_src=67,tp_dst=68 actions=CONTROLLER:65535,goto_table:4 cookie=0x90f01, duration=77.133s, table=2, n_packets=126, n_bytes=10454, priority=55,udp,nw_dst=10.0.41.1,tp_dst=53 actions=CONTROLLER:65535,goto_table:4 cookie=0x90f01, duration=77.132s, table=2, n_packets=0, n_bytes=0, priority=55,tcp,nw_dst=10.0.41.1,tp_dst=53 actions=CONTROLLER:65535,goto_table:4 cookie=0x4d67b, duration=77.133s, table=2, n_packets=117, n_bytes=9693, priority=55,udp,nw_src=10.0.41.1,tp_src=53 actions=CONTROLLER:65535,goto_table:4 cookie=0x4d67b, duration=77.132s, table=2, n_packets=0, n_bytes=0, priority=55,tcp,nw_src=10.0.41.1,tp_src=53 actions=CONTROLLER:65535,goto_table:4 cookie=0xf751b, duration=78.044s, table=2, n_packets=0, n_bytes=0, priority=45,ip,metadata=0x400000000000000/0x400000000000000 actions=goto_table:5

Test Case Field	Description
	cookie=0x6d8f, duration=41.556s, table=2, n_packets=0, n_bytes=0, priority=41,tcp,metadata=0x400001000000/0xfff00001000000,tp_dst=80,tcp_flags=-fin+syn-rst-psh-ack-urg-ece-cwr actions=CONTROLLER:65535,write_metadata:0x400001000000/0xfff00001000000,goto_table:5 cookie=0x6d8f, duration=40.764s, table=2, n_packets=0, n_bytes=0, priority=41,tcp,metadata=0x100000000004000/0x100000000fff000,tp_dst=888,tcp_flags=-fin+syn-rst-psh-ack-urg-ece-cwr actions=CONTROLLER:65535,write_metadata:0x100000000004000/0x100000000fff000,goto_table:5 cookie=0x6d8f, duration=40.627s, table=2, n_packets=0, n_bytes=0, priority=41,tcp,metadata=0x400004000/0xfff00fff000,tp_dst=800,tcp_flags=-fin+syn-rst-psh-ack-urg-ece-cwr actions=CONTROLLER:65535,write_metadata:0x400004000/0xfff00fff000,goto_table:5 cookie=0x6d587, duration=41.634s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400001000000/0xfff00001000000,tp_dst=80 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0x6d587, duration=41.520s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400001000000/0xfff00001000000,tp_dst=888 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0x95d11, duration=41.961s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400000000000/0xfff00000000000,nw_dst=203.0.113.13,tp_dst=443 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0x43f0b, duration=41.889s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400000000000/0xfff00000000000,nw_dst=10.0.41.225,tp_dst=8080 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0xde7f1, duration=41.742s, table=2, n_packets=0, n_bytes=0, priority=40,udp,metadata=0x400000000000/0xfff00000000000,nw_dst=10.0.41.225,tp_dst=4000 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0x6d587, duration=41.676s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400001000000/0xfff00001000000,tp_src=80 actions=write_metadata:0xffffffffffffffff/0,goto_table:3

Test Case Field	Description
	cookie=0x6d587, duration=41.486s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400001000000/0xfff00001000000,tp_src=888 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0xd0bd1, duration=41.415s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400000000004/0xfff00000000fff,tp_src=800 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0xecf6, duration=41.334s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400000000005/0xfff00000000fff,tp_src=8888 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0xd0bd1, duration=41.436s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400000000004/0xfff00000000fff,tp_dst=800 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0xecf6, duration=41.360s, table=2, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400000000005/0xfff00000000fff,tp_dst=8888 actions=write_metadata:0xffffffffffffffff/0,goto_table:3 cookie=0x26ef, duration=42.432s, table=2, n_packets=0, n_bytes=0, priority=35,metadata=0x400000000000/0xfff00000000000 actions=write_metadata:0xffffffffffffffff/0,goto_table:5 cookie=0x29a94, duration=81.184s, table=2, n_packets=282, n_bytes=22446, priority=30 actions=write_metadata:0x29a94,goto_table:3 cookie=0xd5afc, duration=78.045s, table=3, n_packets=0, n_bytes=0, priority=45,ip,metadata=0x4000000/0x4000000 actions=goto_table:5 cookie=0x6d8f, duration=41.094s, table=3, n_packets=0, n_bytes=0, priority=41,tcp,metadata=0x4000/0xfff000,nw_src=203.0.113.13,tp_src=443,tcp_flags=-fin+syn-rst-psh-ack-urg-ece-cwr actions=CONTROLLER:65535,write_metadata:0x4000/0xfff000,goto_table:5 cookie=0x6d8f, duration=41.001s, table=3, n_packets=0, n_bytes=0, priority=41,tcp,metadata=0x4000/0xfff000,nw_src=10.0.41.225,tp_src=8080,tcp_flags=-fin+syn-rst-psh-ack-urg-ece-cwr actions=CONTROLLER:65535,write_metadata:0x4000/0xfff000,goto_table:5 cookie=0x95d11, duration=41.138s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x4000/0xfff000,nw_src=203.0.113.13,tp_src=443 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0x43f0b, duration=41.052s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x4000/0xfff000,nw_src=10.0.41.225,tp_sr

Test Case Field	Description
	c=8080 actions=write_metadata:0xffffffffffffffff/0,goto_ta-ble:4 cookie=0xde7f1, duration=40.921s, table=3, n_packets=0, n_bytes=0, prior-ity=40,udp,metadata=0x4000/0xfff000,nw_src=10.0.41.225,tp_src=4000 actions=write_metadata:0xffffffffffffffff/0,goto_ta-ble:4 cookie=0x6d587, duration=40.896s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_dst=80 actions=write_metadata:0xffffffffffffffff/0,goto_ta-ble:4 cookie=0x6d587, duration=40.799s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_dst=888 actions=write_metadata:0xffffffffffffffff/0,goto_ta-ble:4 cookie=0x6d587, duration=40.852s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_src=80 actions=write_metadata:0xffffffffffffffff/0,goto_ta-ble:4 cookie=0x6d587, duration=40.825s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_src=888 actions=write_metadata:0xffffffffffffffff/0,goto_ta-ble:4 cookie=0xd0bd1, duration=40.729s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x400004000/0xfff00fff000,tp_src=800 ac-tions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0xecf6, duration=40.565s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x500004000/0xfff00fff000,tp_src=8888 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0xd0bd1, duration=40.663s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x400004000/0xfff00fff000,tp_dst=800 ac-tions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0xecf6, duration=40.543s, table=3, n_packets=0, n_bytes=0, prior-ity=40,tcp,metadata=0x500004000/0xfff00fff000,tp_dst=8888 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0x26ef, duration=42.418s, table=3, n_packets=0, n_bytes=0, priority=35,metadata=0x4000/0xfff000 ac-tions=write_metadata:0xffffffffffffffff/0,goto_table:5 cookie=0x29a94, duration=80.685s, table=3, n_packets=282, n_bytes=22446, priority=30 ac-tions=write_metadata:0x29a94,goto_table:4

Test Case Field	Description
	cookie=0x64f19, duration=79.686s, table=4, n_packets=281, n_bytes=24670, priority=41 actions=NORMAL,IN_PORT cookie=0x1c2bd, duration=79.184s, table=5, n_packets=0, n_bytes=0, priority=30 actions=drop **debug-mudtables-sensor.json:** The following maps the flow rules above to the associated MUD file rules. This is for debug purposes only to verify that the MUD rules have been applied appropriately. { "input": { "mud-url": "https://sensor.nist.local/nistmud1", "switch-id": "openflow:123917682138002" } } { "output": { "flow-rule": [{ "flow-id": "https://sensor.nist.local/nist-mud1/NO_FROM_DEV_ACE_MATCH_DROP", "byte-count": 1602, "table-id": 2, "priority": 35, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "metadataMatchGoToTable(5)", "packet-count": 9 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/loc1-frdev/2", "byte-count": 0, "table-id": 2, "dst-local-networks-flag": true, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(proto-col=6,srcPort=888,dstPort=-1,targetTable=3)",

Test Case Field	Description
	"packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4fr/myctl0-frdev*", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "*https://sensor.nist.local/nist-mud1*", "flow-name": "metadataDestIpAndPortMatchGo-ToNext(destIp=10.0.41.225,srcPort=-1,destPort=4000,proto-col=17,sendToController=false)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4fr/myman0-frdev/1*", "dst-manufacturer": "*sensor.nist.local*", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "*https://sensor.nist.local/nist-mud1*", "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=8888,targetTable=3)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4fr/myman0-frdev/2*", "dst-manufacturer": "*sensor.nist.local*", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "*https://sensor.nist.local/nist-mud1*", "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(proto-col=6,srcPort=8888,dstPort=-1,targetTable=3)", "packet-count": 0 },

Test Case Field	Description
	{ "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/loc1-frdev/1", "byte-count": 0, "table-id": 2, "dst-local-networks-flag": true, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=888,targetTable=3)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/ent0-frdev", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "metadataDestIpAndPortMatchGoToNext(destIp=10.0.41.225,srcPort=-1,destPort=8080,protocol=6,sendToController=false)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/man0-frdev/1", "dst-manufacturer": "otherman.nist.local", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=800,targetTable=3)", "packet-count": 0 }, {

Test Case Field	Description
	"flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/cl0-frdev", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "metadataDestIpAndPortMatchGo-ToNext(destIp=203.0.113.13,srcPort=-1,destPort=443,proto-col=6,sendToController=false)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/man0-frdev/2", "dst-manufacturer": "otherman.nist.local", "byte-count": 0, "table-id": 2, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(proto-col=6,srcPort=800,dstPort=-1,targetTable=3)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/loc0-frdev/2", "byte-count": 0, "table-id": 2, "dst-local-networks-flag": true, "priority": 40, "src-model": "https://sensor.nist.local/nist-mud1", "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=80,targetTable=3)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/loc0-frdev/1",

Test Case Field	Description
	"byte-count": 0, "table-id": 2, "dst-local-networks-flag": true, "priority": 40, "src-model": "*https://sensor.nist.local/nist-mud1*", "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=80,dstPort=-1,targetTable=3)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/man0-todev/TCP_DIRECTION_CHECK*", "byte-count": 0, "table-id": 2, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 41, "src-manufacturer": "*otherman.nist.local*", "flow-name": "MetadataTcpSynSrcIpAndPortMatchToToNextTableFlow(srcPort=-1,dstPort=800,targetTable=5)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4fr/loc0-frdev/TCP_DIRECTION_CHECK*", "byte-count": 0, "table-id": 2, "dst-local-networks-flag": true, "priority": 41, "src-model": "*https://sensor.nist.local/nist-mud1*", "flow-name": "MetadataTcpSynSrcIpAndPortMatchToToNextTableFlow(srcPort=-1,dstPort=80,targetTable=5)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/loc1-todev/TCP_DIRECTION_CHECK*", "src-local-networks-flag": true, "byte-count": 0, "table-id": 2,

Test Case Field	Description
	"dst-model": "https://sensor.nist.local/nist-mud1", "priority": 41, "flow-name": "MetadataTcpSynSrcIpAndPortMatch-ToToNextTableFlow(srcPort=-1,dstPort=888,targetTable=5)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/NO_TO_DEV_ACE_MATCH_DROP", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 35, "flow-name": "metadataMatchGoToTable(5)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/myman0-todev/1", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 40, "src-manufacturer": "sensor.nist.local", "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(proto-col=6,srcPort=8888,dstPort=-1,targetTable=4)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/loc1-todev/1", "src-local-networks-flag": true, "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 40,

Test Case Field	Description
	"flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=888,dstPort=-1,targetTable=4)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/man0-todev/1*", "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "src-manufacturer": "*otherman.nist.local*", "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=800,dstPort=-1,targetTable=4)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/cl0-todev*", "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "flow-name": "metadataSrcIpAndPortMatchGoTo(srcAddress =203.0.113.13,srcPort = 443,dstPort -1,protocol=6,targetTable=4)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/myctl0-todev*", "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "flow-name": "metadataSrcIpAndPortMatchGoTo(srcAddress =10.0.41.225,srcPort = 4000,dstPort -1,protocol=17,targetTable=4)",

Test Case Field	Description
	"packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/ent0-todev", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 40, "flow-name": "metadataSrcIpAndPortMatch-GoTo(srcAddress =10.0.41.225,srcPort = 8080,dstPort -1,protocol=6,targetTable=4)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/man0-todev/2", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 40, "src-manufacturer": "otherman.nist.local", "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=800,targetTable=4)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/myman0-todev/2", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 40, "src-manufacturer": "sensor.nist.local", "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=8888,targetTable=4)", "packet-count": 0 },

Test Case Field	Description
	{ "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/loc0-todev/2*", "src-local-networks-flag": true, "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(proto-col=6,srcPort=80,dstPort=-1,targetTable=4)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/loc1-todev/2*", "src-local-networks-flag": true, "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=888,targetTable=4)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/loc0-todev/1*", "src-local-networks-flag": true, "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=80,targetTable=4)", "packet-count": 0 }, {

Test Case Field	Description
	"flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/cl0-todev/TCP_DIRECTION_CHECK", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 41, "flow-name": "MetadataTcpSynSrcIpAndPortMatch-ToToNextTableFlow (srcIp=203.0.113.13,srcPort=443,dstIp=null,dstPort=-1,tar-getTable=5)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4to/ent0-todev/TCP_DIRECTION_CHECK", "byte-count": 0, "table-id": 3, "dst-model": "https://sensor.nist.local/nist-mud1", "priority": 41, "flow-name": "MetadataTcpSynSrcIpAndPortMatch-ToToNextTableFlow (srcIp=10.0.41.225,srcPort=8080,dstIp=null,dstPort=-1,tar-getTable=5)", "packet-count": 0 }] } }
Overall Results	Pass

660 IPv6 is not supported in this implementation.

5.1.2.2 Test Case IoT-2-v4

Table 5-3: Test Case IoT-2-v4

Test Case Field	Description
Parent Requirement	(CR-3) The IoT DDoS example implementation shall include a MUD manager that can request a MUD file and signature from a MUD file server.
Testable Requirement	(CR-3.b) The MUD manager shall use the GET method (RFC 7231) to request MUD and signature files (per RFC 7230) from the MUD file server, but it cannot validate the MUD file server's TLS certificate by using the rules in RFC 2818. (CR-3.b.1) The MUD manager shall drop the connection to the MUD file server. (CR-3.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD manager cannot validate the TLS certificate of a MUD file server when trying to retrieve the MUD file for a specific IoT device, the MUD manager will drop the connection to the MUD file server and configure the router/switch according to locally defined policy regarding whether to allow or block traffic to the IoT device in question.
Associated Test Case(s)	IoT-11-v4
Associated Cybersecurity Framework Subcategory(ies)	PR.AC-7
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager.

Test Case Field	Description
	3. The MUD file server that is hosting the MUD file of the device under test does not have a valid TLS certificate. 4. Local policy has been defined to ensure that if the MUD file for a device is located on a server with an invalid certificate, the router/switch will be configured to deny all communication to and from the IoT device except standard network services (DHCP, DNS, network time protocol [NTP]). 5. The MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Power on the IoT device and connect it to the test network. 2. On the IoT device, using the dhclient application with appropriate configuration file, manually emit a DHCPv4 message containing the device's MUD URL (IANA code 161). 3. The MUD manager snoops the DHCP request through the switch and extracts the MUD URL from the DHCP request. 4. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 5. The DHCP server offers an IP address lease to the newly connected IoT device. 6. The IoT device requests this IP address lease, which the DHCP server acknowledges. 7. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, determines that it does not have a valid TLS certificate, and drops the connection to the MUD file server. 8. The MUD manager configures the router/switch that is closest to the IoT device so that it denies all communications to and from the IoT device except for standard network services (DHCP, DNS, NTP).

Test Case Field	Description
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to local policy for communication to/from the IoT device. Only standard network services are to be allowed (DHCP, DNS, NTP)—this is the standard policy on MUD file verification failures.
Actual Results	**<u>IoT device before DHCP request</u>:** `python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B` `{` `    "input": {` `        "mac-address": "00:13:EF:20:1D:6B"` `    }` `}` `{` `    "output": {` `        "src-local-networks-flag": true,` `        "src-quarantine-flag": false,` **`        "src-blocked-flag": false,`** `        "src-model": "UNCLASSIFIED",` `        "src-manufacturer": "UNCLASSIFIED",` `        "metadata": "100300300000000"` `    }` `}` **<u>MUD manager logs—exception when there is an issue with MUD file</u>:** **`MudfileFetcher: fetchAndInstall : MUD URL =`** ***`https://sen-sor.nist.local/nistmud1`*** **`2019-09-03 14:41:34,114 | ERROR | n-dispatcher-232 | Mud-FileFetcher                | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | Error fetching MUD file -- not installing`** `org.apache.http.conn.HttpHostConnectException: Connect to` *`sensor.nist.local`*`:443 [`*`sensor.nist.local`*`/127.0.0.1] failed: Connection refused (Connection refused)` `        at org.apache.http.impl.conn.DefaultHttpClientConnec-tionOperator.connect(DefaultHttpClientConnectionOpera-tor.java:159)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `        at org.apache.http.impl.conn.PoolingHttpClientConnec-tionManager.connect(PoolingHttpClientConnectionMan-ager.java:373)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `        at org.apache.http.impl.execchain.MainClientExec.es-tablishRoute(MainClien-tExec.java:381)[379:wrap_file__home_mudmanager_nist-`

Test Case Field	Description
	mud_sdnmud-aggregator_karaf_target_assembly_sys- tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient- 4.5.5.jar:0.0.0] at org.apache.http.impl.execchain.MainClientExec.exe- cute(MainClientExec.java:237)[379:wrap_file__home_mudman- ager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys- tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient- 4.5.5.jar:0.0.0] at org.apache.http.impl.execchain.ProtocolExec.exe- cute(ProtocolExec.java:185)[379:wrap_file__home_mudman- ager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys- tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient- 4.5.5.jar:0.0.0] at org.apache.http.impl.execchain.RetryExec.exe- cute(RetryExec.java:89)[379:wrap_file__home_mudmanager_nist- mud_sdnmud-agg **<u>IoT device after DHCP request</u>:** python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B { "input": { "mac-address": "00:13:EF:20:1D:6B" } } { "output": { "src-local-networks-flag": true, "src-quarantine-flag": false, **"src-blocked-flag": true,** "src-model": "UNCLASSIFIED", "src-manufacturer": "UNCLASSIFIED", "metadata": "500300300000000" } }
Overall Results	Pass

663 IPv6 is not supported in this implementation.

5.1.2.3 Test Case IoT-3-v4

Table 5-4: Test Case IoT-3-v4

Test Case Field	Description
Parent Requirement	(CR-4) The IoT DDoS example implementation shall include a MUD file server that can serve a MUD file and signature to the MUD manager.
Testable Requirement	(CR-4.b) The MUD file server shall serve the file and signature to the MUD manager, and the MUD manager shall check to determine whether the certificate used to sign the MUD file was valid at the time of signing, i.e., the certificate had already expired when it was used to sign the MUD file. (CR-4.b.1) The MUD manager shall cease to process the MUD file. (CR-4.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if a MUD file server serves a MUD file with a signature that was created with an expired certificate, the MUD manager will cease processing the MUD file.
Associated Test Case(s)	IoT-11-v4
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The IoT device's MUD file is being hosted on a MUD file server that has a valid TLS certificate, but the MUD file signature was signed by a certificate that had already expired at the time of signature.

Test Case Field	Description
	4. Local policy has been defined to ensure that if the MUD file for a device has a signature that was signed by a certificate that had already expired at the time of signature, the device's MUD PEP router/switch will be configured to deny all communication to/from the device. 5. The MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Power on the IoT device and connect it to the test network. 2. On the IoT device, using the dhclient application with appropriate configuration file, manually emit a DHCPv4 message containing the device's MUD URL (IANA code 161). 3. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 4. The DHCP server offers an IP address lease to the newly connected IoT device. 5. The IoT device requests this IP address lease, which the DHCP server acknowledges. 6. The DHCP server sends the MUD URL to the MUD manager. 7. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. 8. The MUD file server serves the MUD file and signature to the MUD manager, and the MUD manager detects that the MUD file's signature was created by using a certificate that had already expired at the time of signing. 9. The MUD manager configures the router/switch that is closest to the IoT device so that it denies all communications to and from the IoT device.

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td>Expected Results</td><td>The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to local policy for communication to/from the IoT device. Only standard network services are to be allowed (DHCP, DNS, NTP)—this is the standard policy on MUD file verification failures.</td></tr>
<tr><td>Actual Results</td><td><u>IoT device before DHCP request</u>:
<pre>
python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B
{
    "input": {
        "mac-address": "00:13:EF:20:1D:6B"
    }
}
{
    "output": {
        "src-local-networks-flag": true,
        "src-quarantine-flag": false,
        "src-blocked-flag": false,
        "src-model": "UNCLASSIFIED",
        "src-manufacturer": "UNCLASSIFIED",
        "metadata": "100300300000000"
    }
}
</pre>
<u>MUD manager logs—exception when there is an issue with MUD file</u>:
<pre>
MudfileFetcher: fetchAndInstall : MUD URL = https://sensor.nist.local/nistmud1
2019-09-03 14:41:34,114 | ERROR | n-dispatcher-232 | MudFileFetcher | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | Error fetching MUD file -- not installing
org.apache.http.conn.HttpHostConnectException: Connect to sensor.nist.local:443 [sensor.nist.local/127.0.0.1] failed: Connection refused (Connection refused)
       at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnectionOperator.java:159)[379:wrap_file__home_mudmanager_nistmud_sdnmud-aggregator_karaf_target_assembly_system_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]
       at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.connect(PoolingHttpClientConnectionManager.java:373)[379:wrap_file__home_mudmanager_nistmud_sdnmud-aggregator_karaf_target_assembly_system_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]
</pre></td></tr>
</table>

Test Case Field	Description
	`at org.apache.http.impl.execchain.MainClientExec.es-tablishRoute(MainClien-tExec.java:381)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `at org.apache.http.impl.execchain.MainClientExec.exe-cute(MainClientExec.java:237)[379:wrap_file__home_mudman-ager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `at org.apache.http.impl.execchain.ProtocolExec.exe-cute(ProtocolExec.java:185)[379:wrap_file__home_mudman-ager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `at org.apache.http.impl.execchain.RetryExec.exe-cute(RetryExec.java:89)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-agg` **IoT device after DHCP request:** `python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B` `{` `"input": {` `"mac-address": "00:13:EF:20:1D:6B"` `}` `}` `{` `"output": {` `"src-local-networks-flag": true,` `"src-quarantine-flag": false,` **`"src-blocked-flag": true,`** `"src-model": "UNCLASSIFIED",` `"src-manufacturer": "UNCLASSIFIED",` `"metadata": "500300300000000"` `}` `}`
Overall Results	Pass

666 IPv6 is not supported in this implementation.

667 5.1.2.4 Test Case IoT-4-v4

668 **Table 5-5: Test Case IoT-4-v4**

Test Case Field	Description
Parent Requirement	(CR-5) The IoT DDoS example implementation shall include a MUD manager that can translate local network configurations based on the MUD file.
Testable Requirement	(CR-5.b) The MUD manager shall attempt to validate the signature of the MUD file, but the signature validation fails (even though the certificate that had been used to create the signature had not been expired at the time of signing, i.e., the signature is invalid for a different reason). (CR-5.b.1) The MUD manager shall cease processing the MUD file. (CR-5.b.2) The MUD manager shall send locally defined policy to the router or switch that handles whether to allow or block traffic to and from the MUD-enabled IoT device.
Description	Shows that if the MUD manager determines that the signature on the MUD file it receives from the MUD file server is invalid, it will cease processing the MUD file and configure the router/switch according to locally defined policy regarding whether to allow or block traffic to the IoT device in question.
Associated Test Case(s)	IoT-11-v4
Associated Cybersecurity Framework Subcategory(ies)	PR.DS-6
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	1. All devices have been configured to use IPv4. 2. This MUD file is not currently cached at the MUD manager. 3. The MUD file that is served from the MUD file server to the MUD manager has a signature that is invalid, even though it was signed by a certificate that had not expired at the time of signing. 4. Local policy has been defined to ensure that if the MUD file for a device has an invalid signature, the device's MUD PEP router/switch

Test Case Field	Description
	will be configured to deny all communications to/from the device except for standard network services (DHCP, DNS, NTP). 5. The MUD PEP router/switch does not yet have any configuration settings with respect to the IoT device being used in the test.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Power on the IoT device and connect it to the test network. 2. On the IoT device, using the dhclient application with appropriate configuration file, manually emit a DHCPv4 message containing the device's MUD URL (IANA code 161). 3. The MUD manager snoops the DHCP request through the switch and extracts the MUD URL from the DHCP request. 4. The DHCP server receives the DHCP message containing the IoT device's MUD URL. 5. The DHCP server offers an IP address lease to the newly connected IoT device. 6. The IoT device requests this IP address lease, which the DHCP server acknowledges. 7. The MUD manager automatically contacts the MUD file server that is located by using the MUD URL, verifies that it has a valid TLS certificate, and requests the MUD file and signature from the MUD file server. 8. The MUD file server sends the MUD file, and the MUD manager detects that the MUD file's signature is invalid. 9. The MUD manager configures the router/switch that is closest to the IoT device so that it denies all communications to and from the IoT device except standard network services (DHCP, DNS, NTP).
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to local policy for communication to/from the IoT device. Only standard network services are to be

Test Case Field	Description
	allowed (DHCP, DNS, NTP)—this is the standard policy on MUD file verification failures.
Actual Results	**<u>IoT device before DHCP request</u>:** `python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B` `{` `    "input": {` `        "mac-address": "00:13:EF:20:1D:6B"` `    }` `}` `{` `    "output": {` `        "src-local-networks-flag": true,` `        "src-quarantine-flag": false,` **`        "src-blocked-flag": false,`** `        "src-model": "UNCLASSIFIED",` `        "src-manufacturer": "UNCLASSIFIED",` `        "metadata": "100300300000000"` `    }` `}` **<u>MUD manager logs—exception when there is an issue with MUD file</u>:** **`MudfileFetcher: fetchAndInstall : MUD URL = `*`https://sen-sor.nist.local/nistmud1`*** **`2019-09-03 14:41:34,114 | ERROR | n-dispatcher-232 | Mud-FileFetcher | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | Error fetching MUD file -- not installing`** `org.apache.http.conn.HttpHostConnectException: Connect to `*`sensor.nist.local`*`:443 [`*`sensor.nist.local`*`/127.0.0.1] failed: Connection refused (Connection refused)` `        at org.apache.http.impl.conn.DefaultHttpClientConnec-tionOperator.connect(DefaultHttpClientConnectionOpera-tor.java:159)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `        at org.apache.http.impl.conn.PoolingHttpClientConnec-tionManager.connect(PoolingHttpClientConnectionMan-ager.java:373)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]` `        at org.apache.http.impl.execchain.MainClientExec.es-tablishRoute(MainClien-tExec.java:381)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-4.5.5.jar:0.0.0]`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>   at org.apache.http.impl.execchain.MainClientExec.exe-
cute(MainClientExec.java:237)[379:wrap_file__home_mudman-
ager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-
tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-
4.5.5.jar:0.0.0]
        at org.apache.http.impl.execchain.ProtocolExec.exe-
cute(ProtocolExec.java:185)[379:wrap_file__home_mudman-
ager_nist-mud_sdnmud-aggregator_karaf_target_assembly_sys-
tem_org_apache_httpcomponents_httpclient_4.5.5_httpclient-
4.5.5.jar:0.0.0]
        at org.apache.http.impl.execchain.RetryExec.exe-
cute(RetryExec.java:89)[379:wrap_file__home_mudmanager_nist-
mud_sdnmud-agg</pre>
<u>IoT device after DHCP request</u>:
<pre>python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B
{
    "input": {
        "mac-address": "00:13:EF:20:1D:6B"
    }
}
{
    "output": {
        "src-local-networks-flag": true,
        "src-quarantine-flag": false,
        "src-blocked-flag": true,
        "src-model": "UNCLASSIFIED",
        "src-manufacturer": "UNCLASSIFIED",
        "metadata": "500300300000000"
    }
}</pre></td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

669 IPv6 is not supported in this implementation.

670 *5.1.2.5 Test Case IoT-5-v4*

671 **Table 5-6: Test Case IoT-5-v4**

Test Case Field	Description
Parent Requirement	(CR-7) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate with approved internet services in the MUD file. (CR-8) The IoT DDoS example implementation shall deny communications from a MUD-enabled IoT device to unapproved internet services (i.e., services that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-7.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to approved internet services. (CR-7.a.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-7.b) An approved internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-7.b.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-8.a) The MUD-enabled IoT device shall attempt to initiate outbound traffic to unapproved (implicitly denied) internet services. (CR-8.a.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.b) An unapproved (implicitly denied) internet service shall attempt to initiate a connection to the MUD-enabled IoT device. (CR-8.b.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.c) The MUD-enabled IoT device shall initiate communications to an internet service that is approved to initiate communications with the MUD-enabled device but not approved to receive communications initiated by the MUD-enabled device. (CR-8.c.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-8.d) An internet service shall initiate communications to a MUD-enabled device that is approved to initiate communications with the internet service but that is not approved to receive communications initiated by the internet service. (CR-8.d.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.

Test Case Field	Description
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with internet services. Further shows that the policies that are configured on the MUD PEP router/switch with respect to communication with internet services will be enforced as expected, with communications that are configured as denied being blocked, and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.IP-1, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json, mudfile-otherman.json*
Preconditions	Test IoT-1-v4 has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the following policies for the IoT device in question (as defined in the MUD file in Section 5.1.3): a) Explicitly permit *https://yes-permit-from.com* to initiate communications with the IoT device. b) Explicitly permit the IoT device to initiate communications with *https://yes-permit-to.com*. c) Implicitly deny all other communications with the internet, including denying: i) the IoT device to initiate communications with *https://yes-permit-from.com* ii) *https://yes-permit-to.com* to initiate communications with the IoT device iii) communication between the IoT device and all other internet locations, such as *https://unnamed-to.com* (by not mentioning this or any other URLs in the MUD file)

Test Case Field	Description
Procedure	Note: Procedure steps with strike-through were not tested due to NAT. 1. As stipulated in the preconditions, right before this test, test IoT-1-v4 must have been run successfully. 2. Initiate communications from the IoT device to *https://yes-permit-to.com* and verify that this traffic is received at *https://yes-permit-to.com*. (egress) ~~3. Initiate communications to the IoT device from *https://yes-permit-to.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress)~~ ~~4. Initiate communications to the IoT device from *https://yes-permit-from.com* and verify that this traffic is received at the IoT device. (ingress)~~ 5. Initiate communications from the IoT device to *https://yes-permit-from.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at *https://yes-permit-from.com*. (ingress) 6. Initiate communications from the IoT device to *https://unnamed.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at *https://unnamed.com*. (egress) ~~7. Initiate communications to the IoT device from *https://unnamed.com* and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress)~~
Expected Results	Each of the results that is listed as needing to be verified in procedure steps above occurs as expected.
Actual Results	Procedure 2: Connection to approved server (*www.nist.local* port 443) successfully initiated by IoT device: `sensor ] wget www.nist.local:443` `--2019-07-04 05:09:29--  http://www.nist.local:443/` `Resolving www.nist.local (www.nist.local)... 203.0.113.13` `Connecting to www.nist.local (www.nist.local)|203.0.113.13|:443... connected.`

Test Case Field	Description
	`HTTP request sent, awaiting response... 200 OK` `Length: 116855 (114K) [text/html]` `Saving to: 'index.html.51'` `index.html.51` `100%[=======================================================================================>] 114.12K   414KB/s    in 0.3s` `2019-07-04 05:09:30 (414 KB/s) - 'index.html.51' saved [116855/116855]`
	Procedure 5: Connection from device (another manufacturer) to server (*www.nist.local* port 443) fails: `anotherman ]` **`wget www.nist.local:443 --timeout 30 --tries 2`** `--2019-05-02 12:14:32--  http://www.nist.local:443/` `Resolving www.nist.local (www.nist.local)... 203.0.113.13` `Connecting to www.nist.local (www.nist.local)|203.0.113.13|:443... failed: Connection timed out.` `Retrying.` `--2019-05-02 12:15:03--  (try: 2)  http://www.nist.local:443/` `Connecting to www.nist.local (www.nist.local)|203.0.113.13|:443... failed: Connection timed out.` `Giving up.`
	Procedure 6: IoT device failed to connect to unapproved server (*www.antd.local* any port): `sensor ]` **`wget www.antd.local --timeout 30 --tries 2`** `--2019-07-04 05:14:57--  http://www.antd.local/` `Resolving www.antd.local (www.antd.local)... 203.0.113.14` `Connecting to www.antd.local (www.antd.local)|203.0.113.14|:80... failed: Connection timed out.` `Retrying.` `--2019-07-04 05:15:28--  (try: 2)  http://www.antd.local/` `Connecting to www.antd.local (www.antd.local)|203.0.113.14|:80... failed: Connection timed out.` `Giving up.`

Test Case Field	Description
Overall Results	Pass

672 IPv6 is not supported in this implementation.

673 5.1.2.6 Test Case IoT-6-v4

674 **Table 5-7: Test Case IoT-6-v4**

Test Case Field	Description
Parent Requirement	(CR-9) The IoT DDoS example implementation shall allow the MUD-enabled IoT device to communicate laterally with devices that are approved in the MUD file. (CR-10) The IoT DDoS example implementation shall deny lateral communications from a MUD-enabled IoT device to devices that are not approved in the MUD file (i.e., devices that are implicitly denied by virtue of not being explicitly approved).
Testable Requirement	(CR-9.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to approved devices. (CR-9.a.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-9.b) An approved device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-9.b.1) The router or switch shall receive the attempt and shall allow it to pass based on the filters from the MUD file. (CR-10.a) The MUD-enabled IoT device shall attempt to initiate lateral traffic to unapproved (implicitly denied) devices. (CR-10.a.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file. (CR-10.b) An unapproved (implicitly denied) device shall attempt to initiate a lateral connection to the MUD-enabled IoT device. (CR-10.b.1) The router or switch shall receive the attempt and shall deny it based on the filters from the MUD file.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP

Test Case Field	Description
	router/switch automatically configured to enforce the route filtering that is described in the device's MUD file with respect to communication with lateral devices. Further shows that the policies that are configured on the MUD PEP router/switch with respect to communication with lateral devices will be enforced as expected, with communications that are configured as denied being blocked and communications that are configured as permitted being allowed.
Associated Test Case(s)	IoT-1-v4
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, PR.DS-5, PR.AC-5, PR.IP-1, PR.PT-3, PR.IP-3, PR.DS-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	Test IoT-1-v4 has run successfully, meaning that the MUD PEP router/switch has been configured to enforce the following policies for the IoT device in question with respect to local communications (as defined in the MUD files in Section 5.1.3): a) Local-network class—Explicitly permit **local communication to and from the IoT device and any local hosts** (including the specific local hosts *anyhost-to* and *anyhost-from*) **for specific services,** as specified in the MUD file by source port: any; destination port: 80; and protocol: TCP, and which party initiates the connection. b) Manufacturer class—Explicitly permit **local communication to and from the IoT device and other classes of IoT devices, as identified by their MUD URL (*www.devicetype.com*), and further constrained** by source port: any; destination port: 80; and protocol: TCP. c) Same-manufacturer class—Explicitly permit **local communication to and from IoT devices of the same manufacturer as the IoT device in question (the domain in the MUD URLs [mudfileserver] of the other IoT devices is the same as the domain in**

Test Case Field	Description
	the MUD URL [mudfileserver] of the IoT device in question), and further constrained by source port: any; destination port: 80; and protocol: TCP. d) Implicitly deny all other local communication that is not explicitly permitted in the MUD file, including denying i) ***anyhost-to* to initiate communications** with the IoT device ii) the **IoT device to initiate communications** with *anyhost-to* by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** iii) the **IoT device to initiate communications with *anyhost-from*** iv) ***anyhost-from* to initiate communications** with the IoT device by **using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** v) communications between the IoT device and all lateral hosts (including *unnamed-host*) whose **MUD URLs are not explicitly mentioned** as being permissible in the MUD file vi) communications between the IoT device and all lateral hosts whose **MUD URLs are explicitly mentioned** as being permissible **but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted** vii) communications between the IoT device and all lateral hosts that are **not from the same manufacturer** as the IoT device in question viii) communications between the IoT device and a lateral host that **is from the same manufacturer but using a source port, destination port, or protocol (TCP or UDP) that is not explicitly permitted**
Procedure	1. As stipulated in the preconditions, right before this test, test IoT-1-v4 must have been run successfully. 2. Local-network (ingress): Initiate communications to the IoT device from *anyhost-from* **for specific permitted service,** and verify that this traffic is received at the IoT device. 3. Local-network (egress): **Initiate communications from the IoT device to *anyhost-from*** for specific permitted service, and verify that

Test Case Field	Description
	this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *anyhost-from*. 4. Local-network, controller, my-controller, manufacturer class (egress): Initiate communications from the IoT device to *anyhost-to* **for specific permitted service,** and verify that this traffic **is received** at *anyhost-to.* 5. Local-network, controller, my-controller, manufacturer class (ingress): **Initiate communications to the IoT device from *anyhost-to*** for specific permitted service, and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at the IoT device. 6. No associated class (egress): Initiate communications from the IoT device to *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted),** and verify that this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *unnamed-host*. 7. No associated class (ingress): Initiate communications to the IoT device from *unnamed-host* (where *unnamed-host* is a host that is not from the same manufacturer as the IoT device in question and whose **MUD URL is not explicitly mentioned in the MUD file as being permitted)**, and verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at the IoT device. 8. Same-manufacturer class (egress): Initiate communications from the IoT device to *same-manufacturer-host* (where *same-manufacturer-host* is **a host that is from the same manufacturer as the IoT device** in question), and verify that this traffic **is received** at *same-manufacturer-host*. 9. Same-manufacturer class (egress): Initiate communications from the IoT device to *same-manufacturer-host* (where *same-manufacturer-host* is **a host that is from the same manufacturer as the IoT device** in question) **but using a port or protocol that is not specified,** and

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>verify that this traffic is received at the MUD PEP, but it is not forwarded by the MUD PEP, nor is it received at same-manufacturer-host.</td></tr>
<tr><td>Expected Results</td><td>Each of the results that is listed as needing to be verified in the procedure steps above occurs as expected.</td></tr>
<tr><td>Actual Results</td><td>2. Local-network (ingress)—allowed:<pre>
laptop] wget sensor:80
--2019-05-07 10:21:03-- http://sensor/
Resolving sensor (sensor)... 10.0.41.190
Connecting to sensor (sensor)|10.0.41.190|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 116344 (114K) [text/html]
Saving to: ‘index.html.3’

index.html.3
100%[==
===================================>] 113.62K
389KB/s   in 0.3s

2019-05-07 10:21:04 (389 KB/s) - ‘index.html.3’ saved
[116344/116344]
</pre><hr>3. Local-network (egress)—blocked:<pre>
sensor] wget laptop:80 --tries 2 --timeout 30
--2019-07-14 03:24:07-- http://laptop/
Resolving laptop (laptop)... 10.0.41.135
Connecting to laptop (laptop)|10.0.41.135|:80...
failed: Connection timed out.
Retrying.

--2019-07-14 03:24:38-- (try: 2) http://laptop/
Connecting to laptop (laptop)|10.0.41.135|:80...
failed: Connection timed out.
Giving up.
</pre><hr>4. Local-network, controller, my-controller, manufacturer class (egress)—allowed:
Local-network:</td></tr>
</table>

Test Case Field	Description
	sensor] **wget laptop:888** --2019-07-17 00:45:37-- http://laptop:888/ Resolving laptop (laptop)... 10.0.41.135 Connecting to laptop (laptop)|10.0.41.135|:888... connected. HTTP request sent, awaiting response... 200 OK Length: 116344 (114K) [text/html] Saving to: 'index.html.7' index.html.7 100%[===>] 113.62K 703KB/s in 0.2s 2019-07-17 00:45:38 (703 KB/s) - 'index.html.7' saved [116344/116344] Controller: sensor] **wget laptop2:8080** --2019-07-14 03:27:43-- http://laptop2:8080/ Resolving laptop2 (laptop2)... 10.0.41.225 Connecting to laptop2 (laptop2)|10.0.41.225|:8080... connected. HTTP request sent, awaiting response... 200 OK Length: 116344 (114K) [text/html] Saving to: 'index.html.53' index.html.53 100%[===>] 113.62K 548KB/s in 0.2s 2019-07-14 03:27:43 (548 KB/s) - 'index.html.53' saved [116344/116344] My-controller: sensor] **python udpping.py --client --npings 6 --host laptop2 --port 4000** start ... Namespace(bind=False, client=True, host='laptop2', npings=6, port=4000, quiet=False, server=False, timeout=False) PING 1 03:31:59 RTT = 1.24670505524

Test Case Field	Description
	PING 2 03:32:00 RTT = 0.812637805939 PING 3 03:32:01 RTT = 0.652308940887 PING 4 03:32:02 RTT = 0.784868001938 PING 5 03:32:02 RTT = 0.573136806488 PING 6 03:32:03 RTT = 0.481912136078 [rc=6] Manufacturer: sensor] **wget anotherman:800** --2019-07-21 05:23:07-- http://anotherman:800/ Resolving anotherman (anotherman)... 10.0.41.245 Connecting to anotherman (another-man)|10.0.41.245|:800... connected. HTTP request sent, awaiting response... 200 OK Length: 116855 (114K) [text/html] Saving to: ‘index.html.1’ index.html.1 100%[===>] 114.12K --.-KB/s in 0.1s 2019-07-21 05:23:08 (816 KB/s) - ‘index.html.1’ saved [116855/116855] 5. Local-network, controller, my-controller, manufacturer class (in-gress)—blocked: Local-network: laptop] **wget sensor:888** --2019-05-10 07:47:18-- http://sensor:888/ Resolving sensor (sensor)... 10.0.41.190 Connecting to sensor (sensor)|10.0.41.190|:888... ^C laptop] wget sensor:888 --timeout 30 --tries 2 --2019-05-10 07:47:29-- http://sensor:888/ Resolving sensor (sensor)... 10.0.41.190 Connecting to sensor (sensor)|10.0.41.190|:888... failed: Connection timed out. Retrying.

Test Case Field	Description
	`--2019-05-10 07:48:00--  (try: 2)  http://sensor:888/` `Connecting to sensor (sensor)|10.0.41.190|:888... failed: Connection timed out.` `Giving up.`
	Controller: `laptop2 ]` **`wget sensor:8080 --tries 2 --timeout 30`** `--2019-07-13 18:42:31--  http://sensor:8080/` `Resolving sensor (sensor)... 10.0.41.190` `Connecting to sensor (sensor)|10.0.41.190|:8080... failed: Connection timed out.` `Retrying.` `--2019-07-13 18:43:02--  (try: 2)  http://sensor:8080/` `Connecting to sensor (sensor)|10.0.41.190|:8080... failed: Connection timed out.` `Giving up.`
	My-controller: `laptop2 ]` **`python udpping.py --client --npings 6 --host sensor --port 4000`** `start ...` `Namespace(bind=False, client=True, host='sensor', npings=10, port=4000, quiet=False, server=False, timeout=False)` `PING 1 18:43:49` `UDPPING FAILED` `PING 2 18:43:50` `UDPPING FAILED` `PING 3 18:43:51` `UDPPING FAILED` `PING 4 18:43:52` `UDPPING FAILED` `PING 5 18:43:53` `UDPPING FAILED` `PING 6 18:43:54` `[rc=0]`
	Manufacturer: `anotherman ]`**`wget sensor:800 --timeout 30 --tries 2`** `--2019-05-20 05:55:48--  http://sensor:800/`

Test Case Field	Description
	`Resolving sensor (sensor)... 10.0.41.190` `Connecting to sensor (sensor)|10.0.41.190|:800... failed: Connection timed out.` `Retrying.` `--2019-05-20 05:56:19--  (try: 2)  http://sensor:800/` `Connecting to sensor (sensor)|10.0.41.190|:800... failed: Connection timed out.` `Giving up.` 6. No associated class (egress)—blocked: `sensor ]` **`ping laptop -c 10`** `PING laptop (10.0.41.135) 56(84) bytes of data.` `--- laptop ping statistics ---` `10 packets transmitted, 0 received, 100% packet loss, time 9355ms` 7. No associated class (ingress)—blocked: `laptop ]` **`ping sensor -c 10`** `PING sensor (10.0.41.190) 56(84) bytes of data.` `--- sensor ping statistics ---` `10 packets transmitted, 0 received, 100% packet loss, time 9337ms` 8. Same-manufacturer class (egress)—allowed: `sensor ]` **`wget sameman:8888`** `--2019-07-17 01:19:08--  http://sameman:8888/` `Resolving sameman (sameman)... 10.0.41.220` `Connecting to sameman (sameman)|10.0.41.220|:8888... connected.` `HTTP request sent, awaiting response... 200 OK` `Length: 116855 (114K) [text/html]` `Saving to: ‘index.html.8’` `index.html.8` `100%[=======================================================================================>] 114.12K   705KB/s in 0.2s` `2019-07-17 01:19:08 (705 KB/s) - ‘index.html.8’ saved [116855/116855]`

Test Case Field	Description
	9. Same-manufacturer class (egress)—blocked: `sensor ] ping sameman -c 10` `PING sameman (10.0.41.220) 56(84) bytes of data.` `--- sameman ping statistics ---` `10 packets transmitted, 0 received, 100% packet loss, time 9383ms`
Overall Results	Pass

675 IPv6 is not supported in this implementation.

676 ### *5.1.2.7 Test Case IoT-9-v4*

677 **Table 5-8: Test Case IoT-9-v4**

Test Case Field	Description
Parent Requirements	(CR-13) The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the MUD PEP router/switch will get configured with all possible instantiations of that rule, insofar as each instantiation contains one of the IP addresses to which the domain in that MUD file rule may be resolved when queried by the SDN-capable switch.
Testable Requirements	(CR-13.a) The MUD file for a device shall contain a rule involving a domain that can resolve to multiple IP addresses when queried by the SDN-capable switch. Flow rules for permitting access to each of those IP addresses will be inserted into the SDN-capable switch, for the device in question, and the device will be permitted to communicate with all of those IP addresses.
Description	Shows that if a domain in a MUD file rule resolves to multiple IP addresses when the address resolution is requested by the router/switch, then

Test Case Field	Description
	1. flow rules instantiating that MUD file rule corresponding to each of these IP addresses will be configured in the switch for the IoT device associated with the MUD file, and 2. the IoT device associated with the MUD file will be permitted to communicate with all the IP addresses to which that domain resolves
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	1. The SDN-capable switch on the home/small-business network does not yet have any flow rules pertaining to the IoT device being used in the test. 2. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 5.1.3. (Therefore, the MUD file used in the test permits the device to send data to *www.updateserver.com*.) 3. The DNS server that the switch uses resolves the domain *www.updateserver.com* to only one IP address. 4. The tester has access to a DNS server that will be used by the SDN-capable switch and can configure it so that it will resolve the domain *www.updateserver.com* to any of these addresses when queried by the SDN-capable switch: x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. 5. There is a server running at each of these three IP addresses.
Procedure	1. Verify that the SDN-capable switch on the home/small-business network does not yet have any flow rules installed with respect to the IoT device being used in the test.

Test Case Field	Description
	2. Run test IoT-1-v4. The result should be that the SDN-capable switch on the home/small-business network has been configured to explicitly permit the IoT device to initiate communication with *www.updateserver.com*. 3. Attempt to reach *www.updateserver.com* on the device, and see that the SDN-capable switch is then configured with flow rules that permit the IoT device to send data to IP addresses x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1. 4. Have the device in question attempt to connect to x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1.
Expected Results	The SDN-capable switch has had its configuration changed, i.e., it has been configured with flow rules that permit the IoT device to send data to multiple IP addresses (i.e., x1.x1.x1.x1, y1.y1.y1.y1, and z1.z1.z1.z1). The IoT device is permitted to send data to each of the servers at these addresses.
Actual Results	In this test, *www.nist.local* (an allowed internet interaction) resolved to two addresses (203.0.113.13 and 203.0.113.15). When the device attempted to reach *www.nist.local*, both IP addresses were allowed by the flows as intended. The flow rules relating to this interaction are shown below: `cookie=0x95d11, duration=365.237s, table=2, n_packets=1, n_bytes=74, priority=40,tcp,metadata=0x400000000000/0xfff00000000000,nw_dst=203.0.113.13,tp_dst=443 actions=wr` `cookie=0x95d11, duration=365.141s, table=2, n_packets=6, n_bytes=493, priority=40,tcp,metadata=0x400000000000/0xfff00000000000,nw_dst=203.0.113.15,tp_dst=443 actions=w` `cookie=0x95d11, duration=365.220s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x4000/0xfff000,nw_src=203.0.113.13,tp_src=443 actions=write_metadata:0xff`

Test Case Field	Description
	`cookie=0x95d11, duration=365.125s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x4000/0xfff000,nw_src=203.0.113.15, tp_src=443 actions=write_metadata:0xff`
Overall Result	Pass

IPv6 is not supported in this implementation.

5.1.2.8 Test Case IoT-10-v4

Table 5-9: Test Case IoT-10-v4

Test Case Field	Description
Parent Requirements	(CR-12) The IoT DDoS example implementation shall include a MUD manager that uses a cached MUD file rather than retrieve a new one if the cache-validity time period has not yet elapsed for the MUD file indicated by the MUD URL. The MUD manager should fetch a new MUD file if the cache-validity time period has already elapsed.
Testable Requirements	(CR-12.a) The MUD manager shall check if the file associated with the MUD URL is present in its cache and shall determine that it is. (CR-12.a.1) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If so, the MUD manager shall apply the contents of the cached MUD file. (CR-12.a.2) The MUD manager shall check whether the amount of time that has elapsed since the cached file was retrieved is greater than the number of hours in the cache-validity value for this MUD file. If so, the MUD manager may (but does not have to) fetch a new file by using the MUD URL received.
Description	Shows that, upon connection to the network, a MUD-enabled IoT device used in the IoT DDoS example implementation has its MUD PEP router/switch automatically configured to enforce the route filtering that is described in the cached MUD file for that device's MUD URL, assuming that the amount of time that has elapsed since the cached MUD file was retrieved is less than or equal to the number of hours in the

Test Case Field	Description
	file's cache-validity value. If the cache validity has expired for the respective file, the MUD manager should fetch a new MUD file from the MUD file server.
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1, ID.AM-2, ID.AM-3, PR.DS-5, DE.AE-1, PR.AC-4, PR.AC-5, PR.IP-1, PR.IP-3, PR.DS-2, PR.PT-3
IoT Device(s) Under Test	Raspberry Pi
MUD File(s) Used	*mudfile-sensor.json*
Preconditions	1. All devices have been configured to use IPv4. 2. The MUD PEP router/switch does not yet have any configuration settings pertaining to the IoT device being used in the test. 3. The MUD file for the IoT device being used in the test is identical to the MUD file provided in Section 5.1.3.
Procedure	Verify that the MUD PEP router/switch for the IoT device to be used in the test does not yet have any configuration settings installed with respect to the IoT device being used in the test. 1. Run test IoT-1-v4. 2. Within 24 hours (i.e., within the cache-validity period for the MUD file) of running test IoT-1-v4, verify that the IoT device that was connected during test IoT-1-v4 is still up and running on the network. Power on a second IoT device that has been configured to emit the same MUD URL as the device that was connected during test IoT-1-v4, and connect it to the test network. 3. On the IoT device, emit a DHCPv4 message containing the device's MUD URL (IANA code 161). 4. The MUD manager snoops the DHCP request through the switch and extracts the MUD URL from the DHCP request.

Test Case Field	Description
	5. The DHCP server receives the DHCPv4 message containing the IoT device's MUD URL. 6. The DHCP server offers an IP address lease to the newly connected IoT device. 7. The IoT device requests this IP address lease, which the DHCP server acknowledges. 8. The MUD manager determines that it has this MUD file cached and checks that the amount of time that has elapsed since the cached file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If the cache validity has been exceeded, the MUD manager will fetch a new MUD file. 9. The MUD manager translates the MUD file's contents into appropriate route filtering rules and installs these rules onto the MUD PEP for the IoT device in question so that this router/switch is now configured to enforce the policies specified in the MUD file.
Expected Results	The MUD PEP router/switch for the IoT device has had its configuration changed, i.e., it has been configured to enforce the policies specified in the IoT device's MUD file. The expected configuration should resemble the following details: **Cache is valid** (the MUD manager does NOT retrieve the MUD file from the MUD file server)**:** Observing the MUD file server logs, notice that only the first DHCP request for a device goes out to the MUD file server. Within the next 24 hours, any additional DHCP requests will not go to the MUD file server to fetch a new MUD file. **Cache is not valid** (the MUD manager does retrieve the MUD file from the MUD file server)**:** Observing the MUD file server logs, notice that the MUD manager fetches a new copy of the MUD file and signature when the cache does not contain the MUD file of interest.
Actual Results	**<u>IoT device initial DHCP event</u>:** `For the first DHClient request:`

Test Case Field	Description
	sensor] date Tue Sep 3 15:01:16 EDT 2019 sensor] alias dhc alias dhc='sudo rm /var/lib/dhcp/dhclient.leases; sudo ifconfig wlan0 0.0.0.0; sudo dhclient -v wlan0 -cf /etc/dhcp/dhclient.conf.toaster' sensor] dhc Internet Systems Consortium DHCP Client 4.3.5 Copyright 2004-2016 Internet Systems Consortium. All rights reserved. For info, please visit https://www.isc.org/software/dhcp/ Listening on LPF/wlan0/00:13:ef:20:1d:6b Sending on LPF/wlan0/00:13:ef:20:1d:6b Sending on Socket/fallback DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 6 DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 7 DHCPREQUEST of 10.0.41.182 on wlan0 to 255.255.255.255 port 67 DHCPOFFER of 10.0.41.182 from 10.0.41.1 DHCPACK of 10.0.41.182 from 10.0.41.1 bound to 10.0.41.182 -- renewal in 17153 seconds. **MUD file server—log of initial fetch:** sudo -E python mudfile-server.py DoGET /nistmud1 127.0.0.1 - - **[03/Sep/2019 15:02:53] "GET /nistmud1 HTTP/1.1" 200 -** Read 9548 chars DoGET /nistmud1/mudfile-sensor.p7s 127.0.0.1 - - [03/Sep/2019 15:02:55] "GET /nistmud1/mudfile-sensor.p7s HTTP/1.1" 200 - Read 3494 chars **MUD manager log file showing MUD file caching:** 2019-09-03 15:02:56,702 | INFO | on-dispatcher-99 | Mud-FileFetcher | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | verification success 2019-09-03 15:02:56,709 | INFO | on-dispatcher-99 | Mud-FileFetcher | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | Write to Cache here **2019-09-03 15:02:56,738 | INFO | on-dispatcher-99 | Mud-CacheDataStoreListener | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | Writing MUD Cache {"mud-cache-entries":[{"cache-timeout":48,"cached-mudfile-name":"*sensor.nist.local*_nistmud1","retrieval-**

Test Case Field	Description
	time":1567537376711,"mud-url":"*https://sensor.nist.local/nistmud1*"}]} 2019-09-03 15:02:56,739 | INFO | on-dispatcher-99 | DatastoreUpdater | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | jsonData = {"mud-cache-entries":[{"cache-timeout":48,"cached-mudfile-name":"*sensor.nist.local*_nistmud1","retrieval-time":1567537376711,"mud-url":"*https://sensor.nist.local/nistmud1*"}]} **<u>IoT device—second DHCP request</u>:** sensor] date **Tue Sep 3 15:03:10 EDT 2019** sensor] dhc Internet Systems Consortium DHCP Client 4.3.5 Copyright 2004-2016 Internet Systems Consortium. All rights reserved. For info, please visit https://www.isc.org/software/dhcp/ Listening on LPF/wlan0/00:13:ef:20:1d:6b Sending on LPF/wlan0/00:13:ef:20:1d:6b Sending on Socket/fallback DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 8 DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 19 DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 12 DHCPREQUEST of 10.0.41.182 on wlan0 to 255.255.255.255 port 67 DHCPOFFER of 10.0.41.182 from 10.0.41.1 DHCPACK of 10.0.41.182 from 10.0.41.1 bound to 10.0.41.182 -- renewal in 17132 seconds. **<u>MUD manager—log file showing cached file in use</u>:** **2019-09-03 15:03:51**,666 | INFO | on-dispatcher-99 | MudFileFetcher | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | **Found file in mud cache length = 9548** 2019-09-03 15:03:51,666 | INFO | on-dispatcher-99 | MudFileFetcher | 93 - gov.nist.antd.sdnmud-impl - 0.1.0 | read 9548 characters **<u>MUD file server—log after second fetch (no change in output)</u>:** sudo -E python mudfile-server.py DoGET /nistmud1 127.0.0.1 - - [03/Sep/2019 15:02:53] "GET /nistmud1 HTTP/1.1" 200 - Read 9548 chars DoGET /nistmud1/mudfile-sensor.p7s

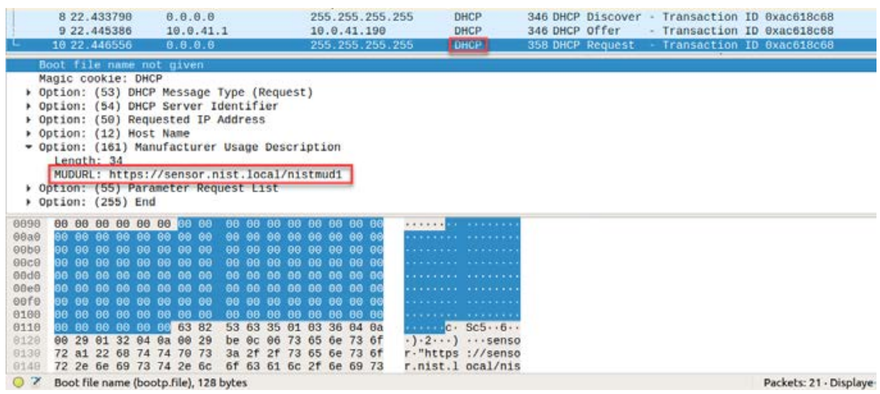
Test Case Field	Description
	`127.0.0.1 - - [03/Sep/2019 15:02:55] "GET /nistmud1/mudfile-sensor.p7s HTTP/1.1" 200 -` `Read 3494 chars`
Overall Results	Pass

IPv6 is not supported in this implementation.

5.1.2.9 Test Case IoT-11-v4

Table 5-10: Test Case IoT-11-v4

Test Case Field	Description
Parent Requirements	(CR-1) The IoT DDoS example implementation shall include a mechanism for associating a device with a MUD file URL (e.g., by having the MUD-enabled IoT device emit a MUD file URL via DHCP, LLDP, or X.509 or by using some other mechanism to enable the network to associate a device with a MUD file URL).
Testable Requirements	(CR-1.a) Upon initialization, the MUD-enabled IoT device shall broadcast a DHCP message on the network, including at most one MUD URL, in https scheme, within the DHCP transaction. (CR-1.a.1) The DHCP server shall be able to receive DHCPv4 DISCOVER and REQUEST with IANA code 161 (OPTION_MUD_URL_V4) from the MUD-enabled IoT device.
Description	Shows that the IoT DDoS example implementation includes IoT devices that can emit a MUD URL via DHCP.
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-1
IoT Device(s) Under Test	Raspberry Pi 1

Test Case Field	Description
MUD File(s) Used	*nistmud1.json*
Preconditions	Device has been developed to emit MUD URL in DHCP transaction.
Procedure	1. Power on a device and connect it to the network. 2. Verify that the device emits a MUD URL in a DHCP transaction. (Use Wireshark to capture the DHCP transaction with options present.)
Expected Results	DHCP transaction with MUD option 161 enabled and MUD URL included
Actual Results	
Overall Results	Pass

5.1.3 MUD Files

This section contains the MUD files that were used in the Build 4 functional demonstration.

5.1.3.1 mudfile-sensor.json

The complete mudfile-sensor.json MUD file has been linked to this document. To access this MUD file please click the link below.

mudfile-sensor.json

5.1.3.2 mudfile-otherman.json

The complete mudfile-otherman.json MUD file has been linked to this document. To access this MUD file please click the link below.

693 mudfile-otherman.json