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

November 2019

PRELIMINARY 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

Executive Summary

The demand for internet-connected "smart" home and small-business devices is growing rapidly, but so too are concerns regarding potential subversion of these devices. 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)] architecture to frustrate subversion of connected devices. The goal of MUD is that Internet of Things (IoT) devices behave only as intended by their manufacturers. MUD provides a standard way for manufacturers to specify the network communications that a device requires to perform its intended function. MUD enables networks to automatically permit each IoT device to send and receive only the traffic it requires to perform as intended and to prohibit all other communication with the device.

- This NCCoE project demonstrates 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.
- Prohibiting unauthorized traffic to and from a device reduces the opportunity for the device to be compromised by a network-based attack and reduces the ability of compromised devices to participate in network-based attacks such as distributed denial of service (DDoS) campaigns.
- 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.
- A DDoS attack can significantly harm an organization that is dependent on the internet to conduct its business. A DDoS attack uses multiple devices in disparate locations to send repeated requests to network servers to overload them and render them inaccessible.
- Recently, IoT devices have been exploited to launch DDoS attacks. IoT devices are often recruited by attackers because the devices may have unpatched or easily discoverable software flaws, and many have minimal security, are unprotected, or are difficult to secure.
- A DDoS attack may result in 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

This National Institute of Standards and Technology (NIST) Cybersecurity Practice Guide demonstrates how to use MUD to reduce the vulnerability of IoT devices to 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 and other tools to satisfy IoT users' security.

CHALLENGE

The term *IoT* is often applied to the aggregate of single-purpose, internet-connected devices, like thermostats, security monitors, and lighting control systems. The IoT is undergoing hypergrowth.

Gartner predicts there will be 20.4 billion IoT devices by 2020 and that the total will reach 25 billion by 2021. Full-featured devices, such as laptops and phones, are protected from most known threats by state-of-the-art security software, but many IoT devices are challenging to secure because they are designed to be inexpensive and to perform a single function. These factors result in processing, timing, memory, and power constraints. Users often do not know what devices are on their networks and lack means for controlling access to them over their life cycles. However, the consequences of not addressing security concerns of IoT devices can be catastrophic. For instance, in typical networking environments, adversaries can detect and attack an IoT device within minutes of it being connected. If it has a known vulnerability, this weakness can be exploited at scale, enabling them to commandeer sets of compromised devices, called *botnets*, to launch large-scale DDoS and other network-based attacks.

SOLUTION

This project demonstrates how MUD strengthens security for IoT devices on home and small-business networks by helping prevent them from being both victims and perpetrators of network-based attacks. This practice guide describes four MUD implementations, three of which are complete:

- 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, still under development, uses equipment supplied by CableLabs to support MUD. It will leverage the Wi-Fi Alliance Easy Connect specification to securely onboard devices to the network. It will also use software-defined networking to create separate trust zones (e.g., network segments) to which devices are assigned according to their intended network function.
- Build 4 uses DigiCert certificates and software developed by the NIST Advanced Networking Technologies Division as a working prototype that demonstrates feasibility and scalability of the MUD specification.

While the NCCoE used a suite of commercial products to address this challenge, this guide does not endorse these 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.

BENEFITS

The NCCoE's practice guide to securing small-business and home IoT devices can help:

- organizations that rely on the internet 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 easily exploited to support DDoS or other network-based attacks
- service providers benefit from reduction of the IoT devices that can be easily 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

SHARE YOUR FEEDBACK

You can view or download the guide 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, easily 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

Darshak Thakore
Mark Walker
CableLabs

Dean Coclin
Clint Wilson
DigiCert

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

Eliot Lear
Brian Weis
Cisco

Tim Jones
Forescout

Adnan Baykal
Global Cyber Alliance

Jaideep Singh
Molex

November 2019

PRELIMINARY 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, 169 pages, (November 2019), 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: November 21, 2019 through January 21, 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. 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

botnets; Internet of Things; IoT; Manufacturer Usage Description; MUD; router; server; software update server; threat signaling.

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
Ashwini Kadam	CableLabs
Craig Pratt	CableLabs
Tao Wan	CableLabs
Russ Gyurek	Cisco
Peter Romness	Cisco
Rob Cantu	CTIA
Katherine Gronberg	Forescout
Rae'-Mar Horne	MasterPeace Solutions
Nate Lesser	MasterPeace Solutions
Tom Martz	MasterPeace Solutions
Daniel Weller	MasterPeace Solutions
Mo Alhroub	Molex
Bill Haag	NIST

Name	Organization
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
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
John Bambenek	ThreatSTOP
Paul Watrobski	University of Maryland

Name	Organization
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 Service provider server Partner and service provider server Prototype medical devices–Raspberry Pi
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 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

Technology Partner/Collaborator	Build Involvement
MasterPeace Solutions	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 is focused on the use of IoT devices in home and small-business environments. Its objective is to show how MUD can be used practically and effectively to reduce the vulnerability of IoT devices to network-based threats, and how MUD can be used to 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. Three implementations of the reference design are demonstrated. A fourth implementation is under development. These implementations are referred to as *builds*, and this volume describes three 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. Because it is still under development, it is not described in detail in this version of the practice guide.
- 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; 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. Non-MUD-capable devices used include laptops, phones, and IoT devices that cannot emit 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 smart television sets. The IoT is experiencing what some might describe as hypergrowth. Gartner forecasts that there will be 20.4 billion IoT devices by 2020 and that the total will reach 25 billion by 2021, while Forbes forecasts the market to be $457 billion by 2020 (a 28.5 percent compounded annual growth rate). As IoT devices become more commonplace in homes and businesses, security concerns are also increasing. IoT devices may have unpatched or easily discoverable software flaws, and many have minimal security, are unprotected, or are difficult to secure. The full-featured devices such as web servers, personal or business computers, and mobile devices with which users are familiar often have state-of-the-art security software protecting them from most known threats. Conversely, many IoT devices are challenging to secure because they are designed to be inexpensive and to perform a single function—resulting in processing, timing, memory, and power constraints. Nevertheless, the consequences of not addressing security concerns of IoT devices can be catastrophic. For instance, in typical networking environments, malicious actors can detect an IoT device within minutes of it being connected and then, unbeknownst to the user, launch an attack on that device. They can also commandeer a group of compromised devices, called a *botnet,* that can be used to launch large-scale attacks. One example of such an attack is a distributed denial of service (DDoS) attack, which 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

1.2 Solution

This project demonstrates how to use MUD to strengthen security while 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 three implementations of the MUD specification that support MUD-capable IoT devices. It describes how one build (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 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 of the number of IoT devices that can be easily used by malicious actors 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 to their businesses 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:

- NIST SP 1800-15A: *Executive Summary*
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics*—what we built and why **(you are here)**
- NIST SP 1800-15C: *How-To Guides*—instructions for building the example solutions

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 solution built at the NCCoE
- benefits of adopting the demonstrated approach

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 example 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.

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 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 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 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 pathnames; 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`**

Typeface/ Symbol	Meaning	Example
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). Use of 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 reducing the ability for these devices to be used in a DDoS or other network-based attack. In addition, in one build (Build 2), network-wide access controls based on threat signaling are provided to protect legacy IoT devices, MUD-capable IoT devices, and fully featured devices (e.g., personal computers). 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 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. The build teams documented the architecture and design of each build. As each build progressed, the team documented the steps taken to install and configure each component of the build. The team 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 will be permitted to send and receive. We verified that attempts to perform prohibited communications would be blocked. The team conducted a risk assessment and a security characteristics 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, home and small-business network users are not intended 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 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 smart lighting controllers, cameras, smartphones, printers, baby monitors, digital video recorders, and smart 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.

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 are 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

It is assumed that:

- 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 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, a MUD manager, a MUD-capable gateway/router/switch, and a threat-signaling-capable gateway/router/switch
    - 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 *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Federal Information Systems and Organizations* (NIST SP 800-53) 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 whitelisting device capabilities and blacklisting 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 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 to 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

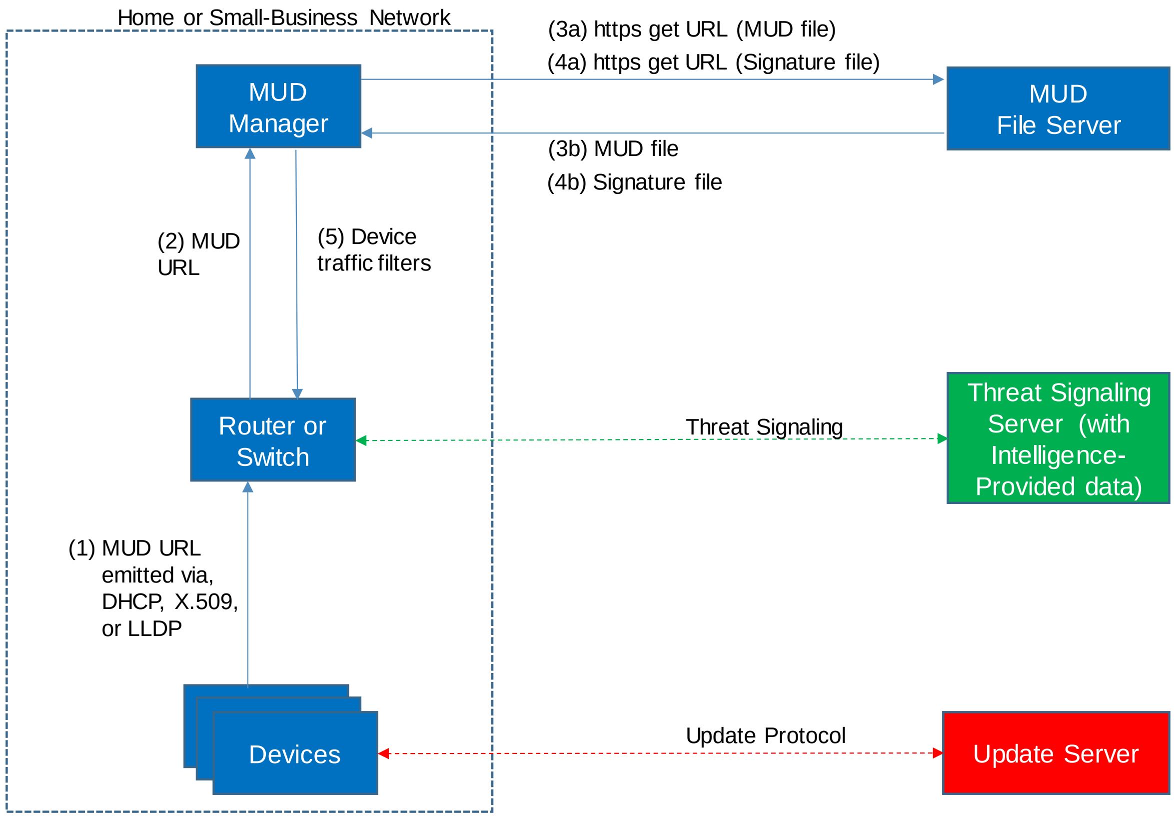
The project architecture is intended for home and small-business networks that are composed of both IoT components and fully featured devices (e.g., personal computers). The architecture is designed to provide three forms of protection:

- 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), 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 (TEAP). 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.

Figure 4-1 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.

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 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 standalone 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 provide support for MUD in its internet gateway equipment and infrastructure, such 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 three builds support MUD and the ability to receive faux updates from a notional update server, only Build 2 currently supports threat signaling. In addition, 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.

The four builds of the reference architecture that have been undertaken, three of which are complete and have been demonstrated, are as follows:

- Build 1 uses products from Cisco Systems, DigiCert, Forescout, and Molex. The Cisco MUD manager is used to support MUD, and the Forescout virtual appliances and enterprise manager are used to perform non-MUD-related device discovery on the network. Molex Power over Ethernet (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., 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, which is still under development, uses products supplied by CableLabs to support MUD. It will leverage the Wi-Fi Alliance Easy Connect specification to securely onboard devices to the network. It will also use software-defined networking to create separate trust zones (e.g., network segments) to which devices are assigned according to their intended network function. Although limited functionality of a preliminary version of this build was demonstrated as part of this project, Build 3 is not yet complete. Therefore, it has not yet been subjected to functional evaluation or demonstration. A brief preview of the architecture and functional elements planned for Build 3 is provided in this practice guide. Full documentation of Build 3 is planned for inclusion in the next phase of this project.
- 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 RFC to demonstrate feasibility and scalability. Certificates from DigiCert are also used.

The logical architectures and detailed descriptions of Builds 1, 2, and 4 can be found in Section 6 (Build 1), Section 7 (Build 2), and Section 9 (Build 4). Build 3 is described briefly in Section 8.

4.2 Physical Architecture

Figure 4-2 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.

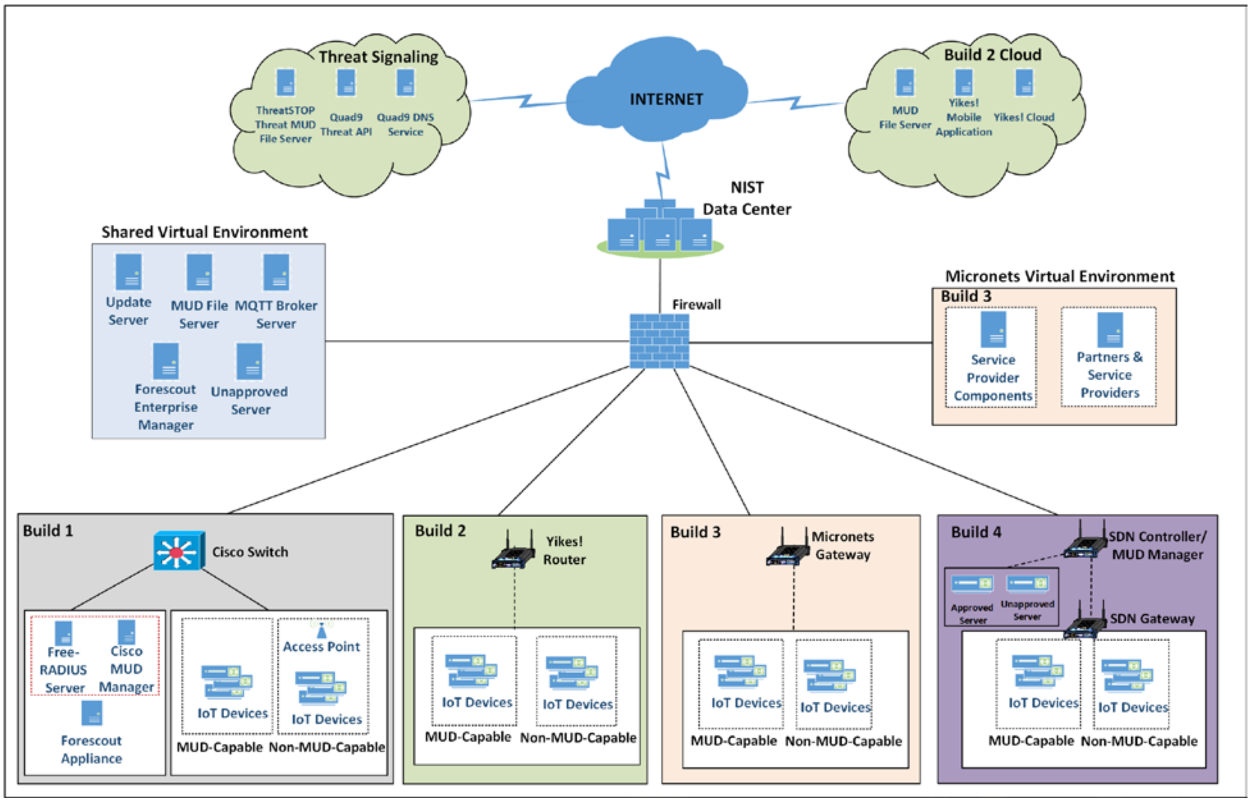
All four builds, as depicted in the diagram, have been implemented, but only three are complete:

- 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 is still under development and is expected to be completed by the next phase of this project. As of this writing, this build’s 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.
- 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. 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, smartphones, lighting devices, a smart 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), and Section 9 (Build 4). Build 3 is described briefly in Section 8.

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 the security characteristic analysis involved assessing how well the reference design addresses the security characteristics that it was intended to support. The NIST Cybersecurity Framework Subcategories were used to provide structure to the security assessment. We consulted the specific sections of each standard that are cited in reference to a Subcategory. The cited sections provide validation points that the example implementations 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 characteristics 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 Special Publication [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 security platform directly addresses NIST SP 800-53 controls AC-3, AC-18, CM-7, 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, seven of the NISTIR 8228 expectations are addressed by the example implementation. 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., power on) (Build 1, Build 2, and Build 4) • MUD URL is provided in LLDP (Build 1) • MUD URL is included in X.509 certificate (Build 3) However, there may be other means for a MUD URL to be learned by a network, and the MUD specification is designed to allow flexibility in this regard.	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.
The MUD file URL, which identifies the device type, among other things, is passed to the MUD manager, which retrieves a MUD file 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)	Device can interface with enterprise asset management systems.	Provides AC-3 Access Enforcement AC-18 Wireless Access CM-7 Least Functionality SC-5 Denial of Service Protection SC-7 Boundary Protection	Provides PR.PT-3 The principle of least functionality is incorporated by configuring systems to provide only essential capabilities. Supports ID.AM-1 Physical devices and systems within the organization are inventoried.

Applicable Project Description Element That Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
		Supports AC-4 Information Flow Enforcement AC-6 Least Privilege AC-24 Access Control Decisions CM-8 System Component Inventory PM-5 System Inventory	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., 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

Applicable Project Description Element That Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
			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 scanners or provides built-in vulnerability identification and reporting capabilities.	Supports AC-24 Access Control Decisions RA-5 Vulnerability Scanning SI-5 Security Alerts, Advisories, and Directives	Supports ID.RA-2 Cyber threat intelligence is received 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.

Applicable Project Description Element That Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
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 control information for enforcement by the router or switch. (all builds)	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 be used to 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)	Device can prevent unauthorized access to all sensitive data transmitted from it over networks.	Provides SC-23 Session Authenticity Supports AC-18 Wireless Access SC-8 Transmission Confidentiality and Integrity	Provides PR.PT-3 The principle of least functionality 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

Applicable Project Description Element That Addresses the Expectation	Applicable NISTIR 8228 Expectations	NIST SP 800-53 Controls Supported	Cybersecurity Framework Subcategories Supported
			mechanisms are used to verify software, firmware, and information integrity.
There exists some mechanism for associating each device with a URL that can be used to 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

Table 5-2 details Cybersecurity Framework Identify, Protect, and Detect Categories and Subcategories that the example implementations directly address or for which the example implementations may serve a supporting role. Those Subcategories 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 cybersecurity risk to organizational operations (including mission, functions, image, or reputation), organizational assets, and individuals.	**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, 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

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
		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, 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

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
		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-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

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
		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, 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

Cybersecurity Framework Category	Cybersecurity Framework Subcategory	Informative References
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

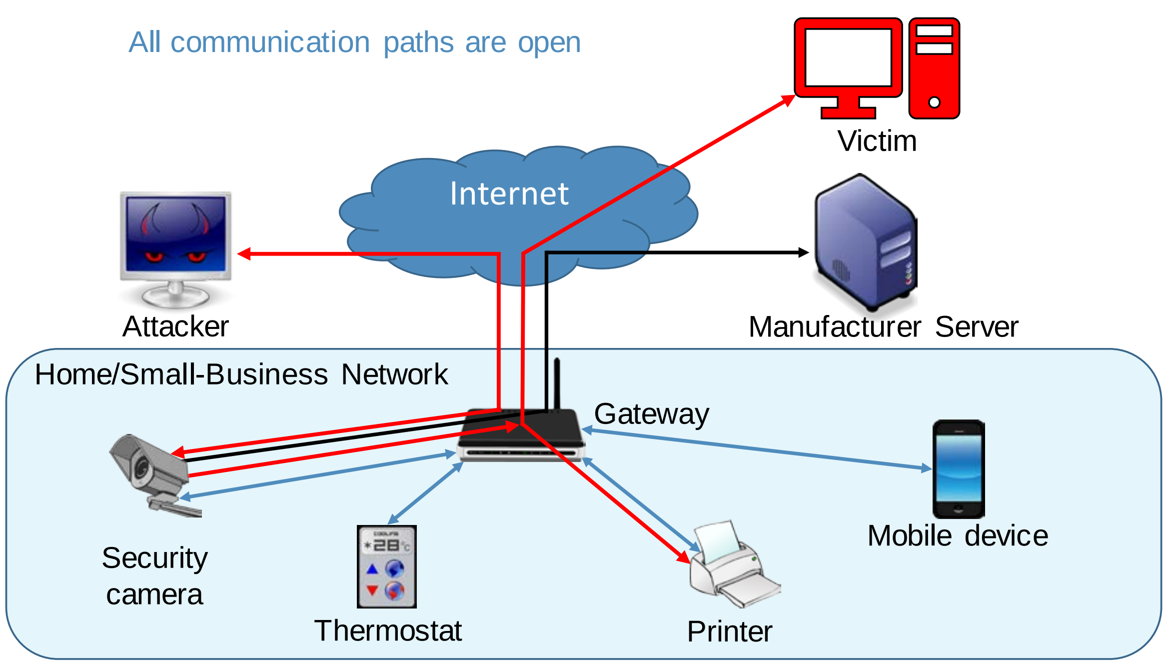
This section presents two scenarios involving home and small-business networks that have IoT devices. In the first scenario, MUD is not deployed on the network, so IoT devices are vulnerable to being port scanned and are not restricted from exchanging traffic with either external sites or other devices 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 the IoT devices are permitted to exchange traffic with only external domains and internal devices that are explicitly specified in their MUD file. Use of 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

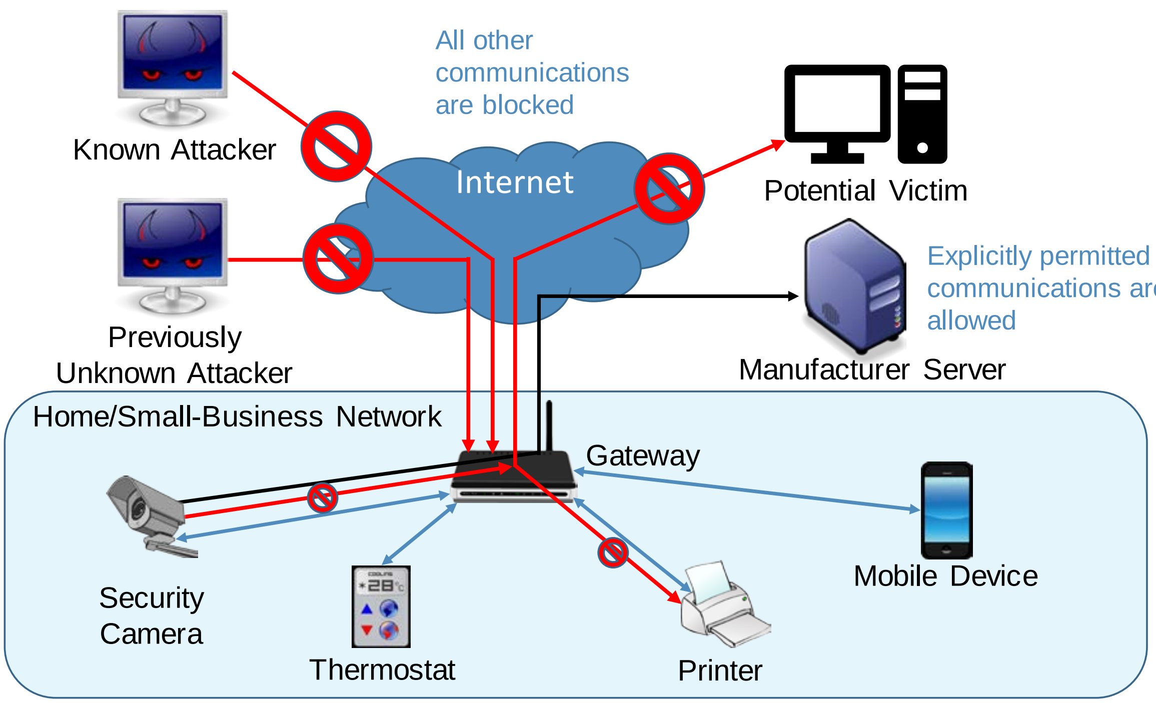


All communication paths are open. The IoT devices on the network can be port scanned (and perhaps hijacked) by an attacker on the internet. IoT devices are permitted to communicate to and from intended services, such as a manufacturer update server as desired. However, the IoT devices are also reachable 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 device’s 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 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 malicious actor.

6 Build 1

The Build 1 implementation uses products from Cisco Systems, DigiCert, Forescout, and Molex. Cisco equipment is used to support 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 a MUD-capable IoT device. 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 is being undertaken within Cisco's Enterprise Central Software Group with an eye toward improving the product offering over time. Cisco has 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 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 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 continuous posture assessment. As of June 30, 2019, 3400 customers in more than 85 countries rely on Forescout's infrastructure-agnostic solution to reduce the risk of business disruption from security incidents or breaches, demonstrate security compliance, and 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 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 control(s) that the product provides. 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 6-1 Products and Technologies

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

Component	Product	Function	Cybersecurity Framework Subcategories
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.mudmaker.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 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	Cisco 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

Component	Product	Function	Cybersecurity Framework Subcategories
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; 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) 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

Component	Product	Function	Cybersecurity Framework Subcategories
Non-MUD-capable IoT device	Camera Smartphones Smart lighting devices Smart 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

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, the Cisco MUD manager was used to support IoT devices that emit their MUD URLs via DHCP messages and 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 (MAB) 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., 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: 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 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 will be detailed 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 be used to 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 (SAML), 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 devkits were modified by the NCCoE 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

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

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 smart 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 smart 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, smartphones, lighting, a smart 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 smartphone.

6.2.9.2.2 Smartphones

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

6.2.9.2.3 Lighting

Two types of smart lighting devices are used in the laboratory implementation. These smart lighting components are controlled and managed by a smartphone.

6.2.9.2.4 Smart Assistant

A smart assistant is utilized in the laboratory implementation. The device is used to demonstrate and test the wide range of network traffic generated by a smart assistant.

6.2.9.2.5 Printer

A smart printer is connected to the laboratory network wirelessly to demonstrate smart 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 smartphone.

6.2.9.2.7 Wireless Access Point

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

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 a manual process, with the firmware update taking place over the CoAP, UDP, and trivial file transfer protocol protocols. 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, communication to and from the unapproved server and the MUD-capable IoT device was tested.

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.

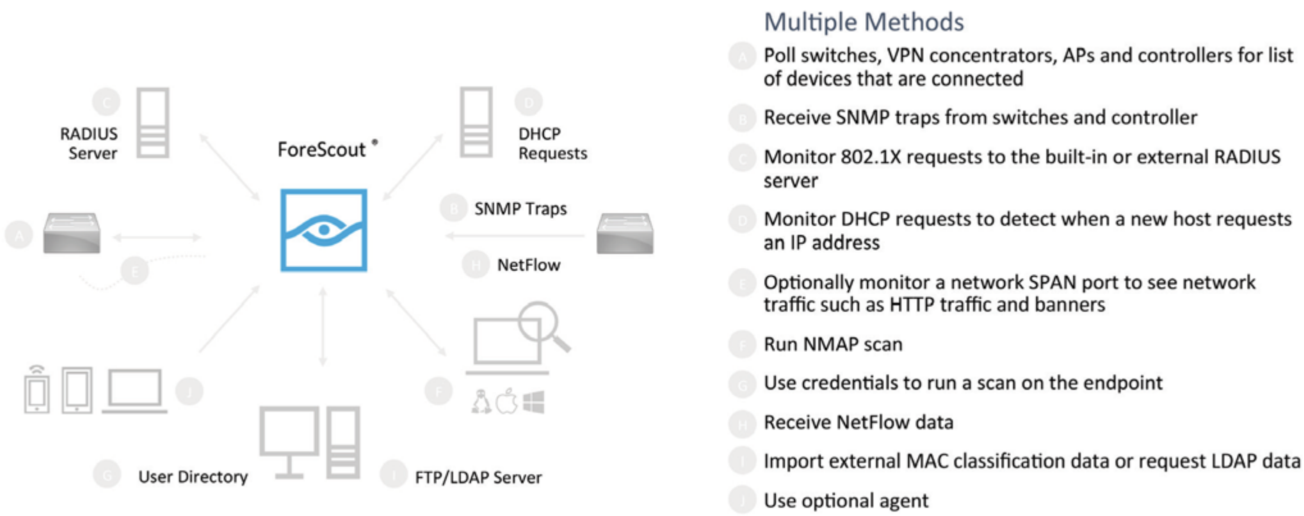
6.2.13 IoT Device Discovery

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 included 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.

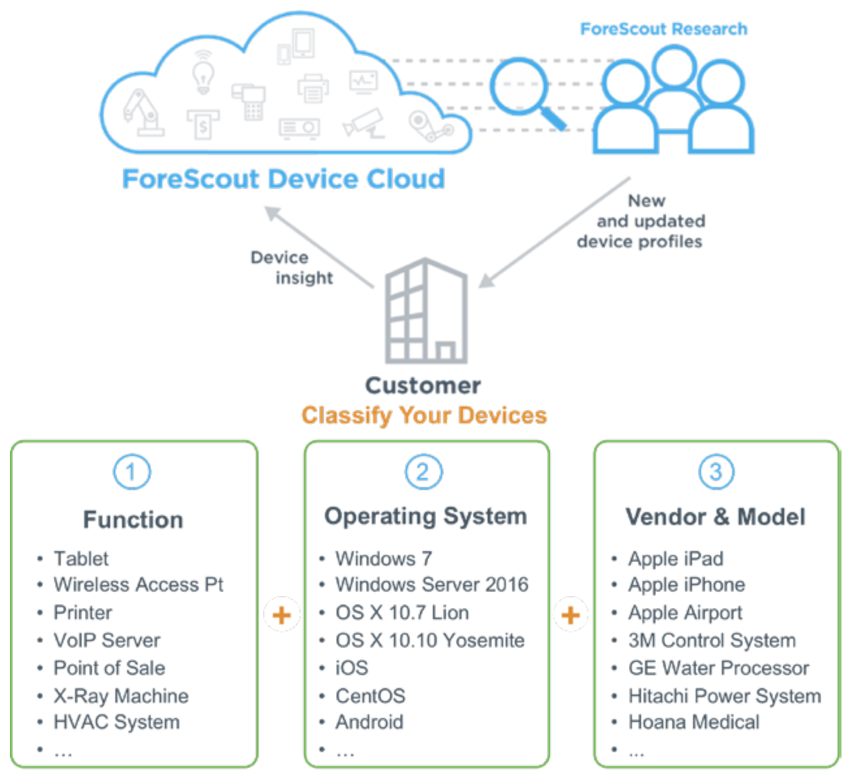
Figure 6-1 Methods the Forescout Platform Can Use to Discover and Classify IP-Connected Devices



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 the time of 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

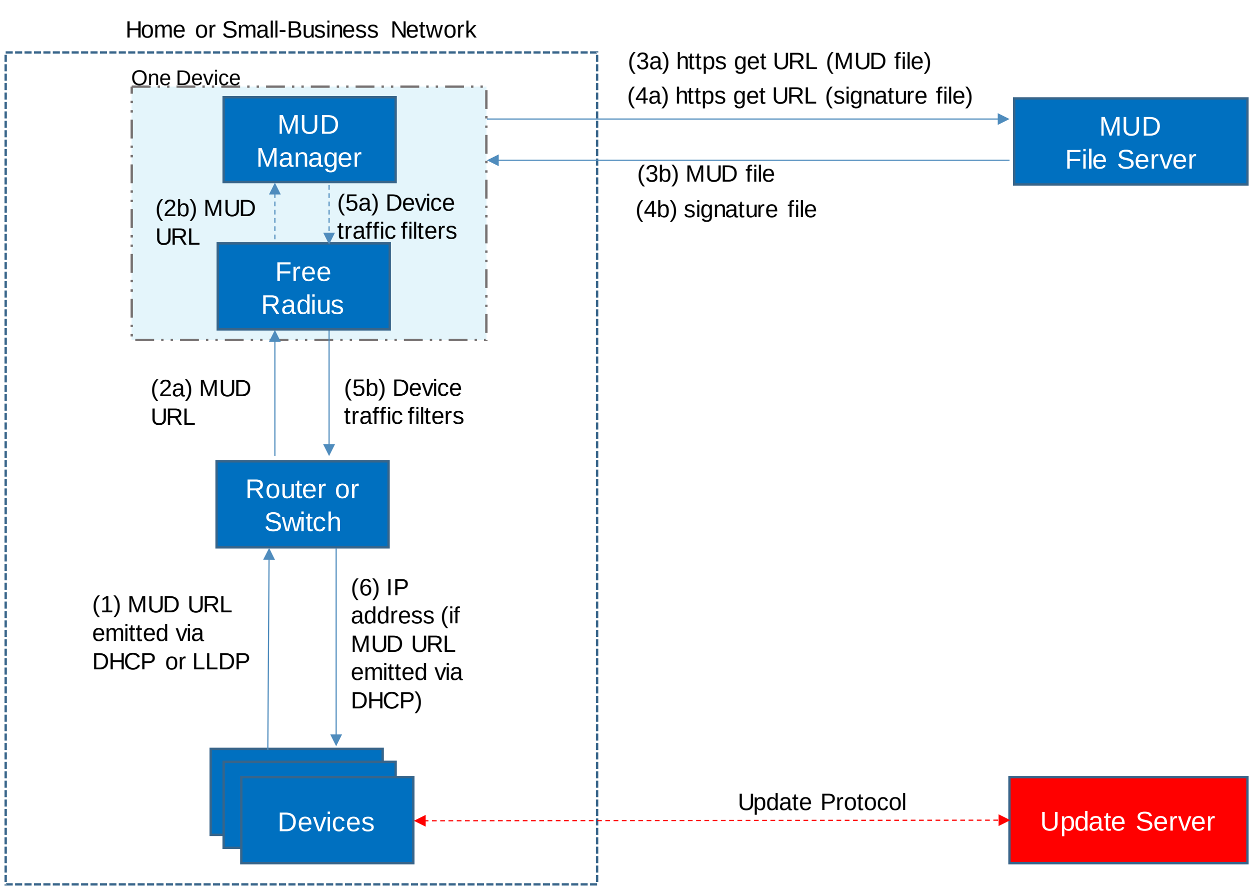
Figure 6-3 depicts the logical architecture of Build 1. Build 1 is designed with a single device serving as the MUD manager and FreeRADIUS server that interfaces with the Catalyst 3850-S switch over TCP/IP. It supports two mechanisms for MUD URL emission: DHCP and LLDP. Only the steps performed when using DHCP emission are depicted in Figure 6-3. 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 fetches 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 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

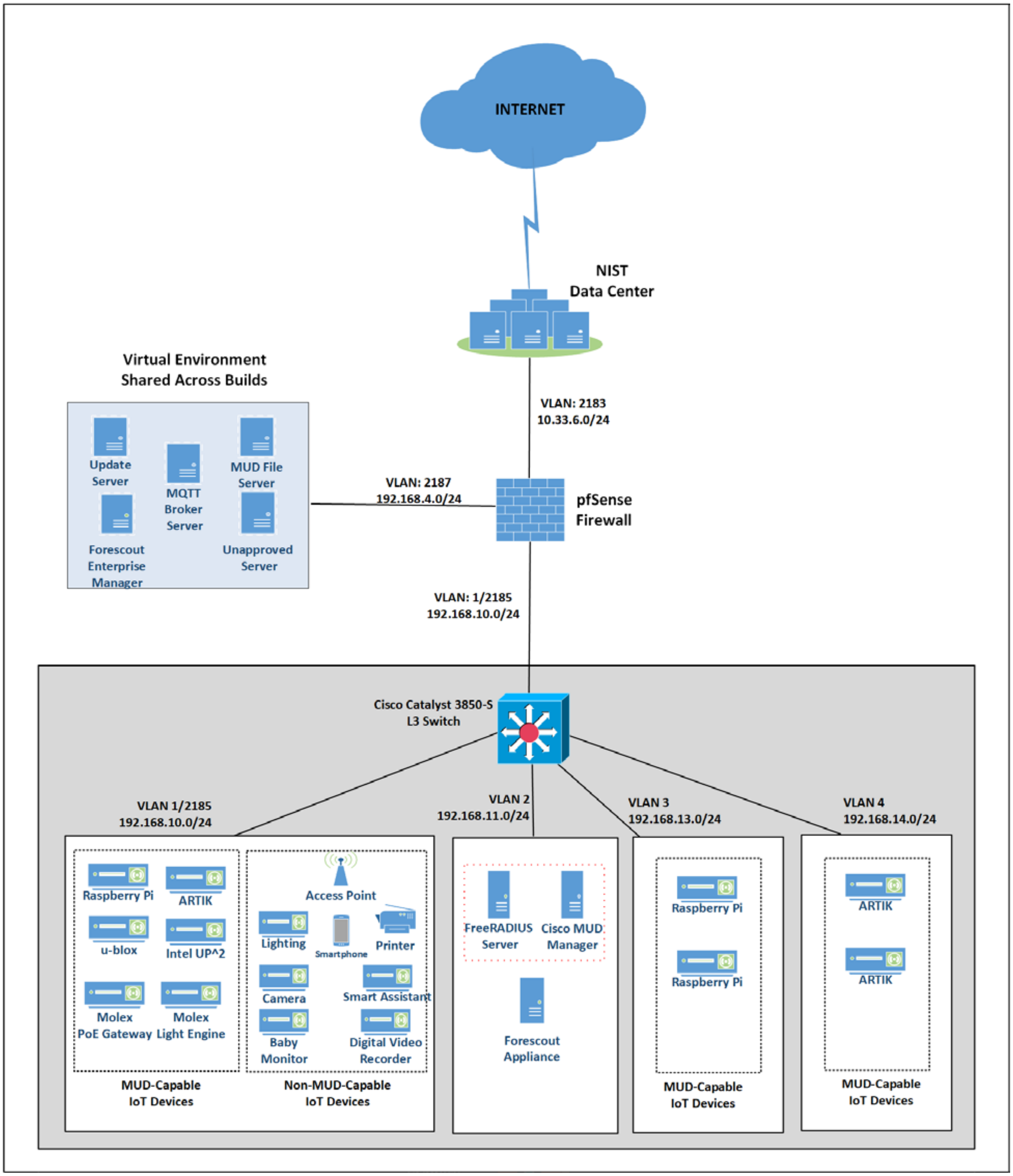


6.3.2 Physical Architecture

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.

Figure 6-4 Physical Architecture–Build 1



A full description of Cisco's proof-of-concept MUD manager implementation can be found 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.

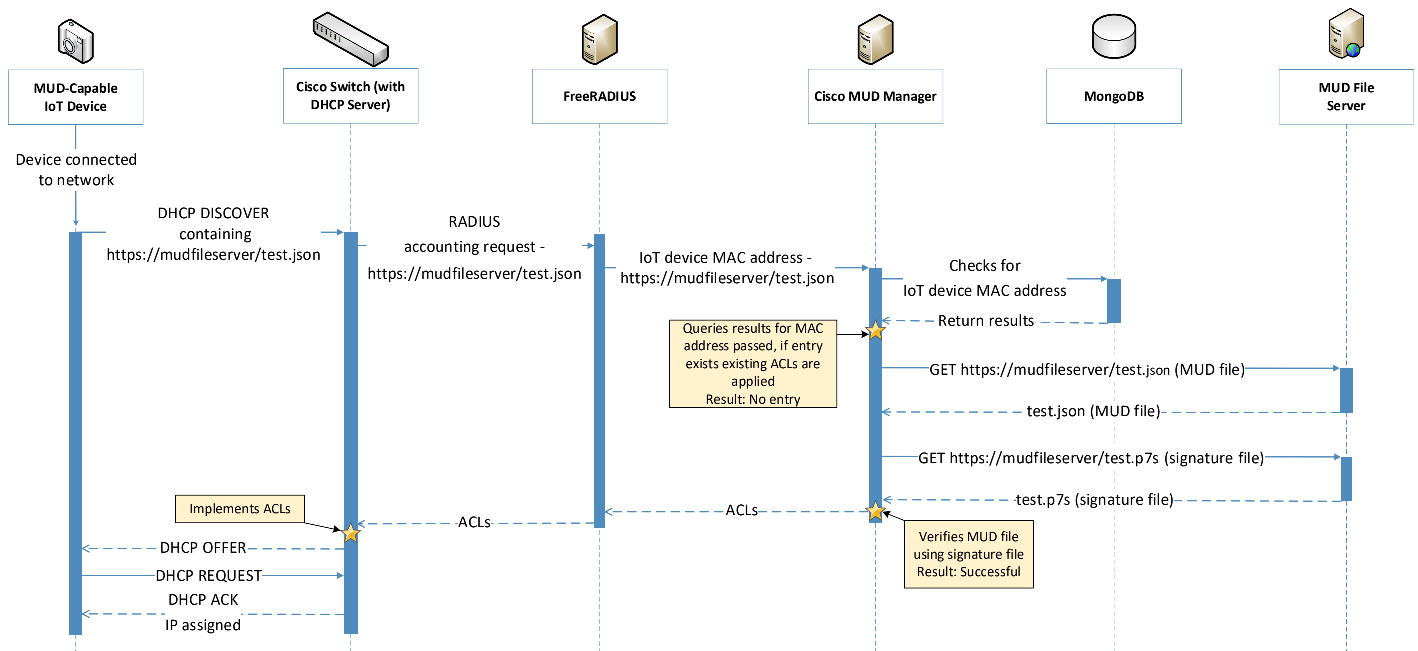
6.3.3 Message Flow

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

6.3.3.1 *Onboarding MUD-Capable Devices*

Figure 6-5 shows the message flow of the process of onboarding a MUD-capable IoT device that emits a MUD URL via DHCPv4.

Figure 6-5 MUD-Capable IoT Device Onboarding 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.

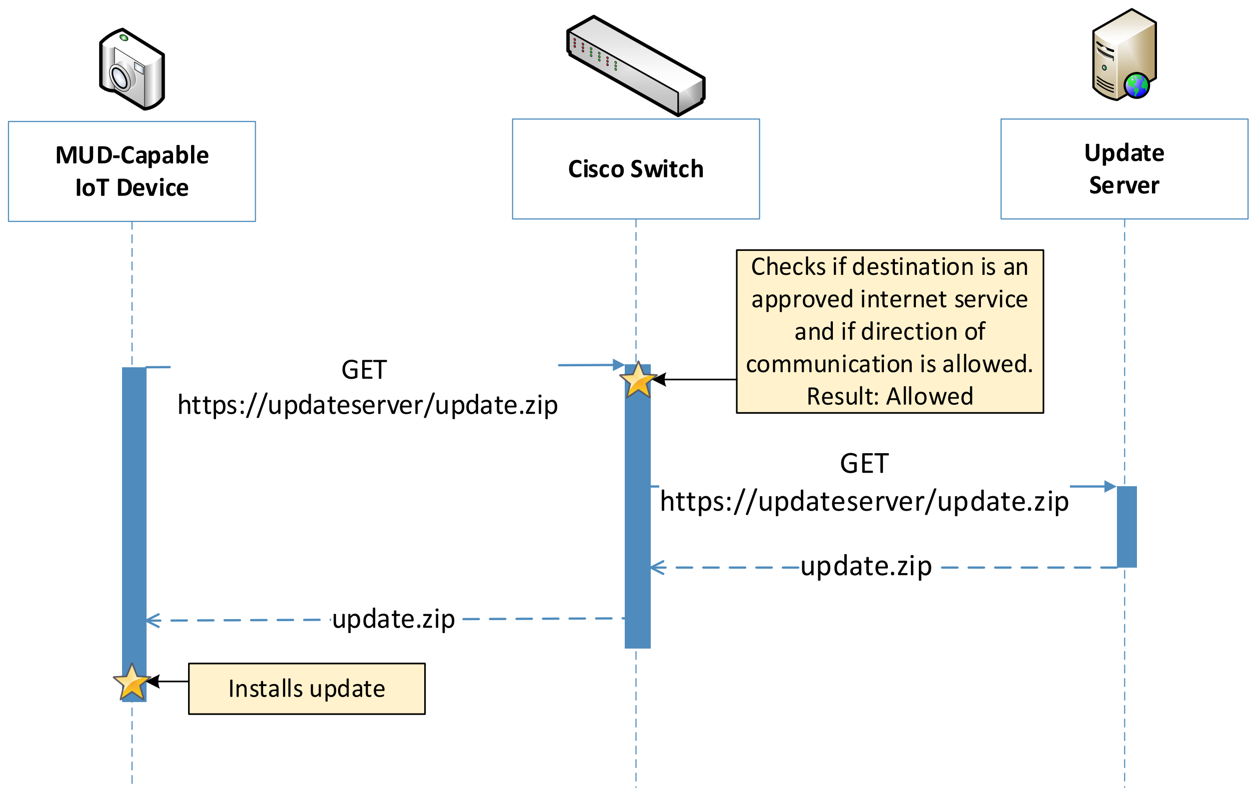
- The DHCP server on the Cisco switch recognizes that option and extracts the MUD URL from the DHCP message, which is sent from the switch to the FreeRADIUS server in the associated accounting request. From this point, the FreeRADIUS server sends the MAC address and MUD URL for the newly onboarded device 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.

1266 **Figure 6-6 Update Process Message Flow–Build 1**



1267

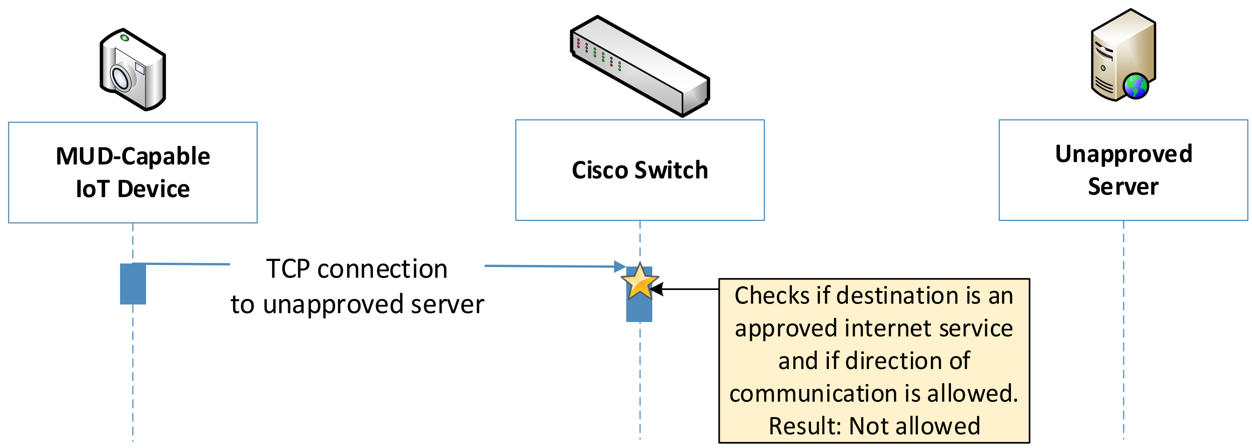
1268 As shown in Figure 6-6 Update Process Message Flow–Build 1, the message flow is as follows:

1269 - A MUD-capable IoT device initiates an https request to the update server.
1270 - The Cisco switch checks its ACLs to determine if the destination and direction of
1271 communication should be allowed for the IoT device and allows the request after verification.
1272 - The update server completes the process by sending the requested update package to the IoT
1273 device.

1274 6.3.3.3 *Prohibited Traffic*

1275 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:

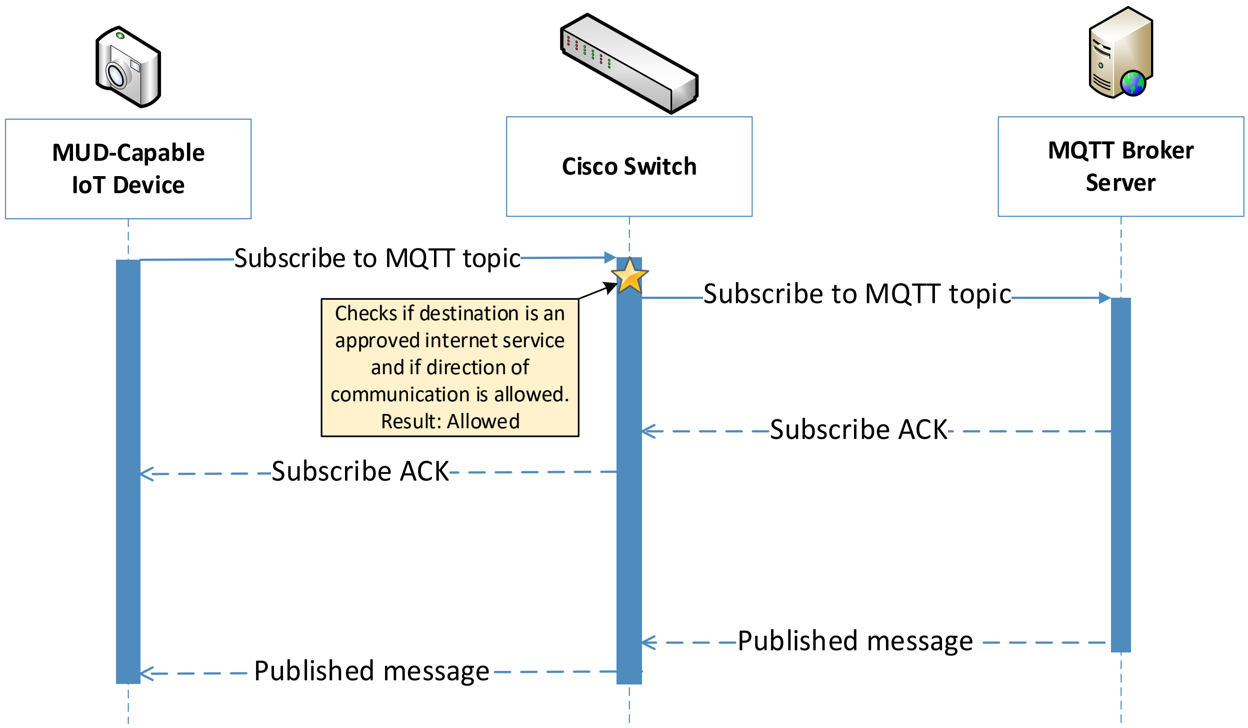
- The MUD-capable IoT device initiates a TCP request to an unapproved server.
- The Cisco switch checks its ACLs to determine if the destination and direction of communication should be allowed for the IoT device and blocks the unapproved communication.

At the time of publication, 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 can be found 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 allows the Subscribe message after verification.
- The MQTT broker server sends a Subscribe 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. Build 1 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 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 were performed to evaluate Build 1's MUD-related capabilities, and Table 6-3 summarizes the exercises that were 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 is designed to verify support. The tests and exercises 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 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	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**	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
	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.	**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.**		

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
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-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 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-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-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 the time of 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**	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 the time of signing. According to local	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		router/switch will be configured to deny all communication to/from the device.	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-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	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**	When the MUD-capable IoT device is connected to the network, its MUD PEP **router/switch will be configured to enforce the route filter-**	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
	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	**to/from all other internet locations.**	**ing 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.	
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).	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
	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 **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	Test IoT-1 has run successfully, meaning that the MUD PEP **router/switch has been configured based on the MUD**	When the MUD-capable **IoT device's IP address lease expires,** the MUD-related configuration for	Failed (not supported)

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. **NIST SP 800-53 Rev. 4** CM-8, MP-6	**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.**	that IoT device will be removed from its 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.	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
	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-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	A MUD-capable IoT device is configured to emit a MUD URL. Upon being connected to the network, its MUD file is retrieved,	Upon reconnection of the IoT device to the network, **the MUD manager does not contact**	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	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 control systems is created and maintained, incorporating	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.	**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.	

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.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 is capable of emitting 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

In addition to supporting MUD, Build 1 demonstrates capabilities with respect to device discovery, 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 the time of testing. 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 the time of testing. 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 are used 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.

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 be used to 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

The 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 control(s) that the product provides. 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 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. MUD file maker (see previous row) can be used to create	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
	MUD files. Each MUD file is also associated with a separate MUD signature file.		
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 Smartphones Smart lighting devices Smart assistant Printer Digital video recorder	Acts as typical IoT devices on a network; creates network connections to cloud services	ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
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 Quad9 threat API regarding whether the	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		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 requesting device. For domains that have been flagged as dangerous,	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		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 of the domain that the router had tried, un-	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		successfully, 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 sites to and from which traffic should be	ID.RA-1 ID.RA-2 ID.RA-3

Component	Product	Function	Cybersecurity Framework Subcategories
		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.	

1401 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 project 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 each other. 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 for provisioning and managing publicly trusted X.509 certificates, can be found 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 devkits were modified by the NCCoE 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, smartphones, smart lighting, a smart 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 smartphone.

7.2.8.2.2 Smartphones

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

7.2.8.2.3 Lighting

Two types of smart lighting devices are used in the laboratory implementation. These smart lighting components are controlled and managed by a smartphone.

7.2.8.2.4 Smart Assistant

A smart assistant is utilized in the laboratory implementation. The device is used to demonstrate and test the wide range of network traffic generated by a smart assistant.

7.2.8.2.5 Printer

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

7.2.8.2.6 Digital Video Recorder

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

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 can be found 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, smart watch, wearable, car
- home and office: computer, laptop, printer, IP phone, scanner
- smart home: IP camera, smart device, smart 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 the time of publication 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 the 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 may be found 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, which 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

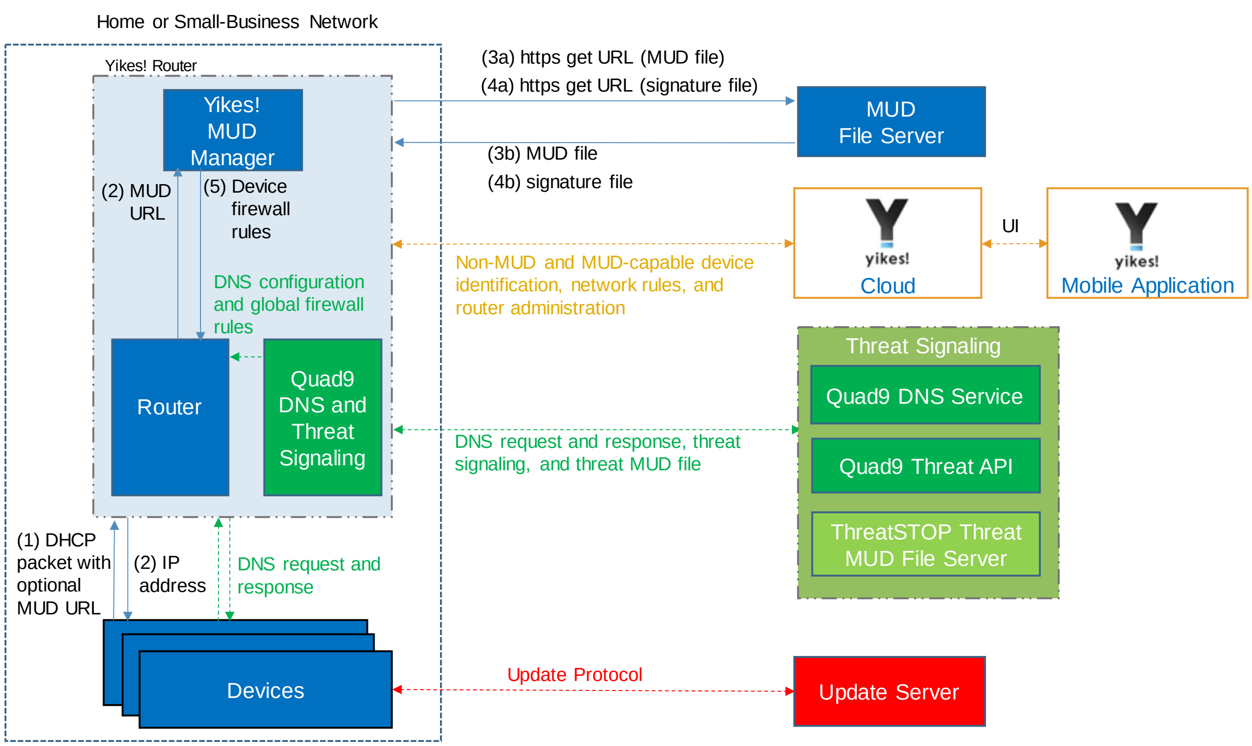
In this section we present the logical architecture of Build 2 relative to how it instantiates the reference architecture depicted in Figure 4-1. We also describe Build 2's physical architecture and present message flow diagrams for some of its processes.

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 onboarding 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 onboarded.

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 onboarding 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.

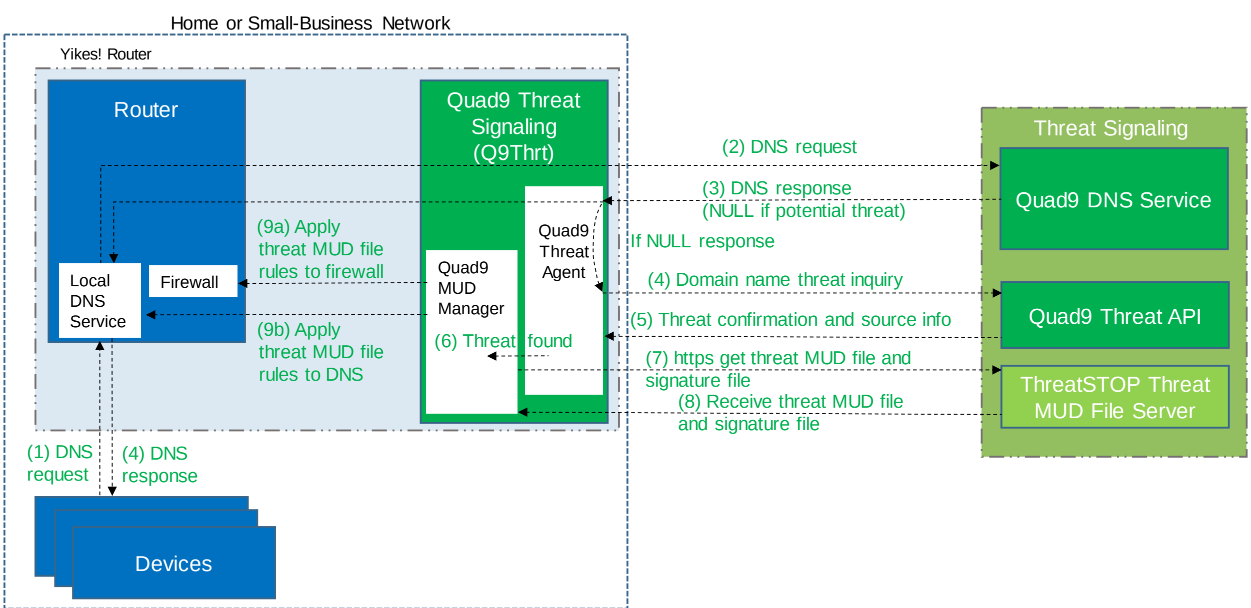
It is possible that both Yikes! cloud traffic policies and MUD file traffic policies could both apply to any given device in the network. For any given device, if these policies conflict, MUD file policies are given precedence over Yikes! traffic policies. If the policies do not conflict, they are both 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.

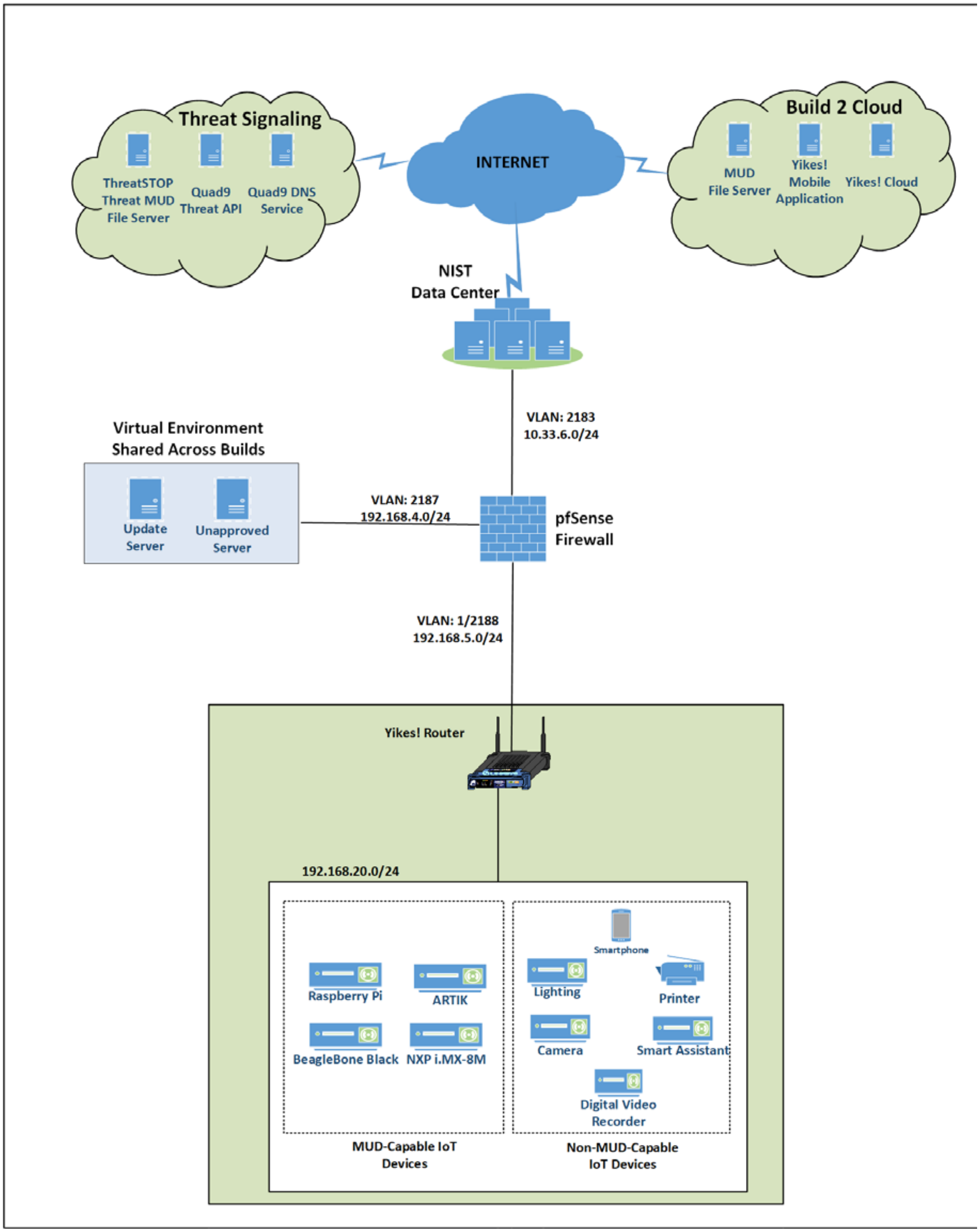
7.3.2 Physical Architecture

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.

1815 **Figure 7-3 Physical Architecture—Build 2**



1816

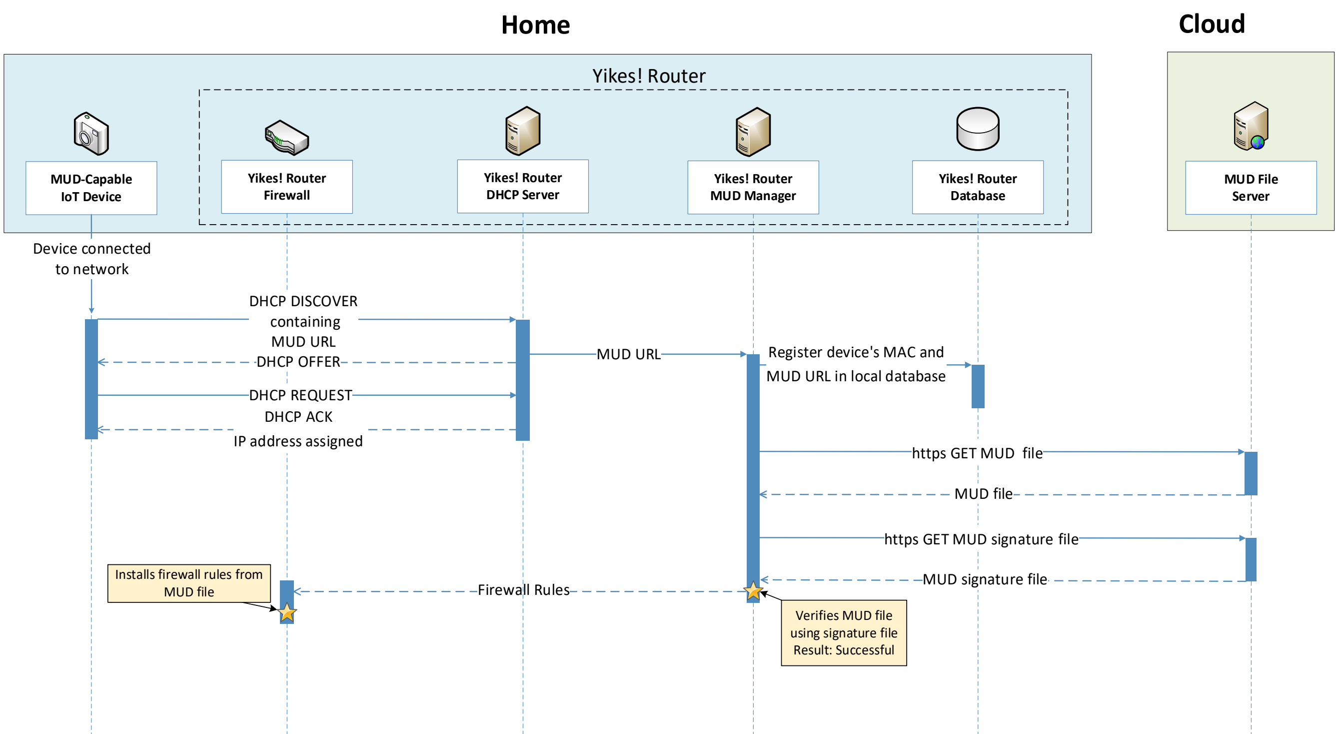
7.3.3 Message Flow

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

7.3.3.1 *Onboarding MUD-Capable Devices*

Figure 7-4 MUD-Capable IoT Device Onboarding Message Flow - Build 2 depicts the message flows involved in the process of onboarding a MUD-capable IoT device in Build 2.

Figure 7-4 MUD-Capable IoT Device Onboarding 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 MUD-Capable IoT Device Onboarding Message Flow—Build 2 because it is not involved in onboarding 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 MUD-Capable IoT Device Onboarding Message Flow—Build 2 because it is the only cloud component that is involved in onboarding the MUD-capable device.)

As shown in Figure 7-4 MUD-Capable IoT Device Onboarding Message Flow—Build 2, 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.

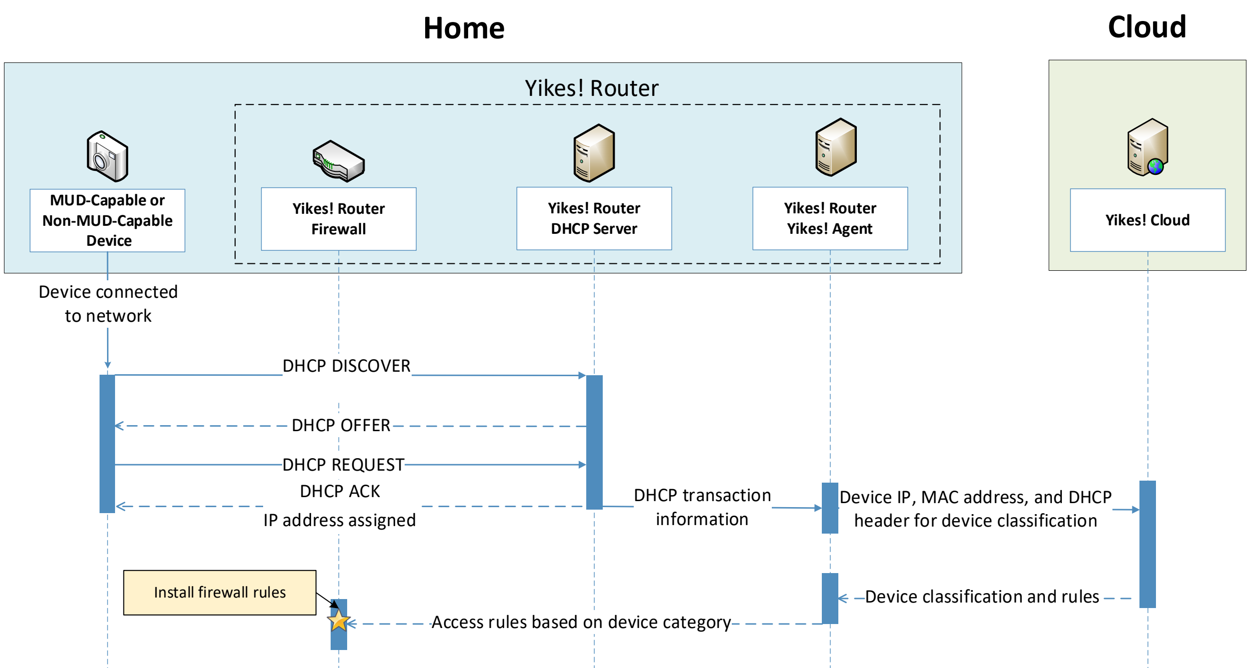
7.3.3.2 *Onboarding All Devices*

Figure 7-5 depicts the message flows involved in the process of onboarding 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 onboarding 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.

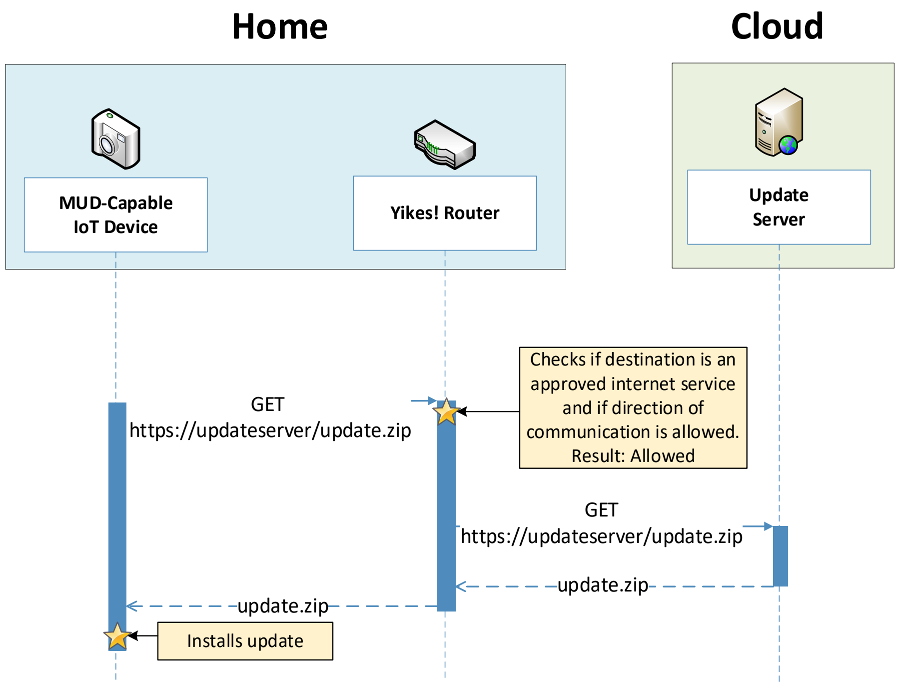
Figure 7-5 Device Onboarding Message Flow—Build 2



7.3.3.3 *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 7-6 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:

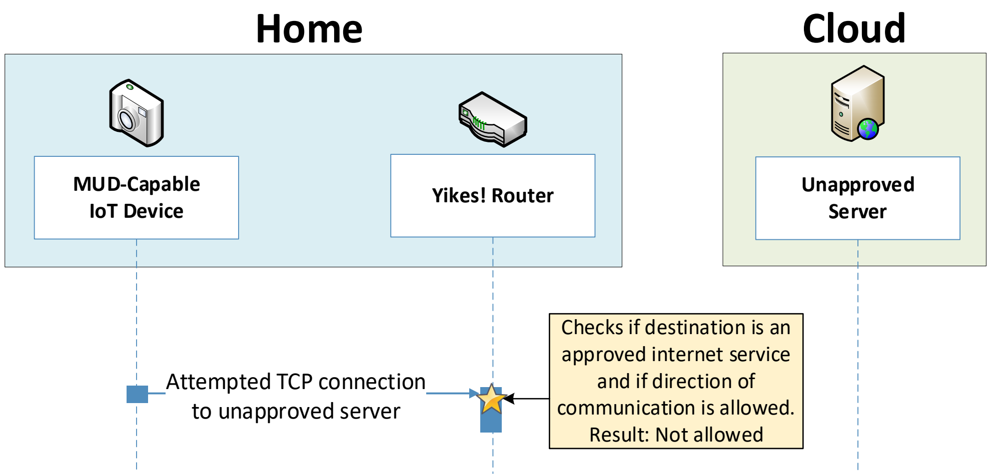
- The device generates an https GET request to its update server.
- The Yikes! router will consult the firewall rules for this device to verify that it is permitted to send traffic to the update server. Assuming there were explicit rules in the device's MUD file enabling it to send messages to this update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

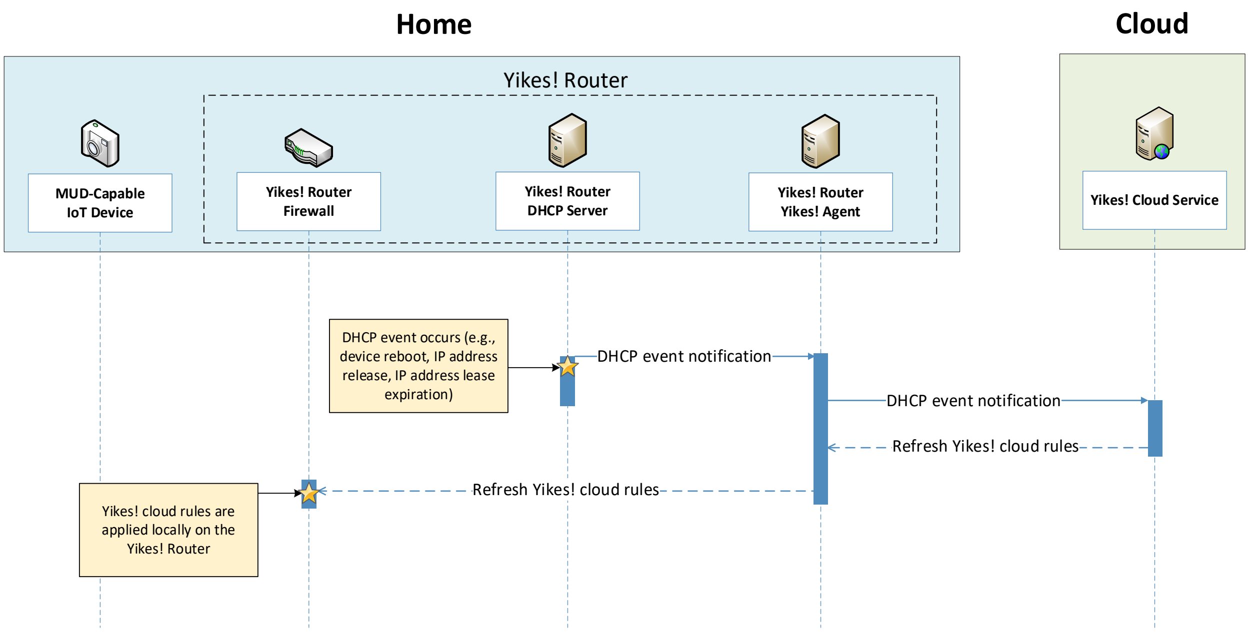


7.3.3.5 *DHCP Events*

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 onboarded 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*

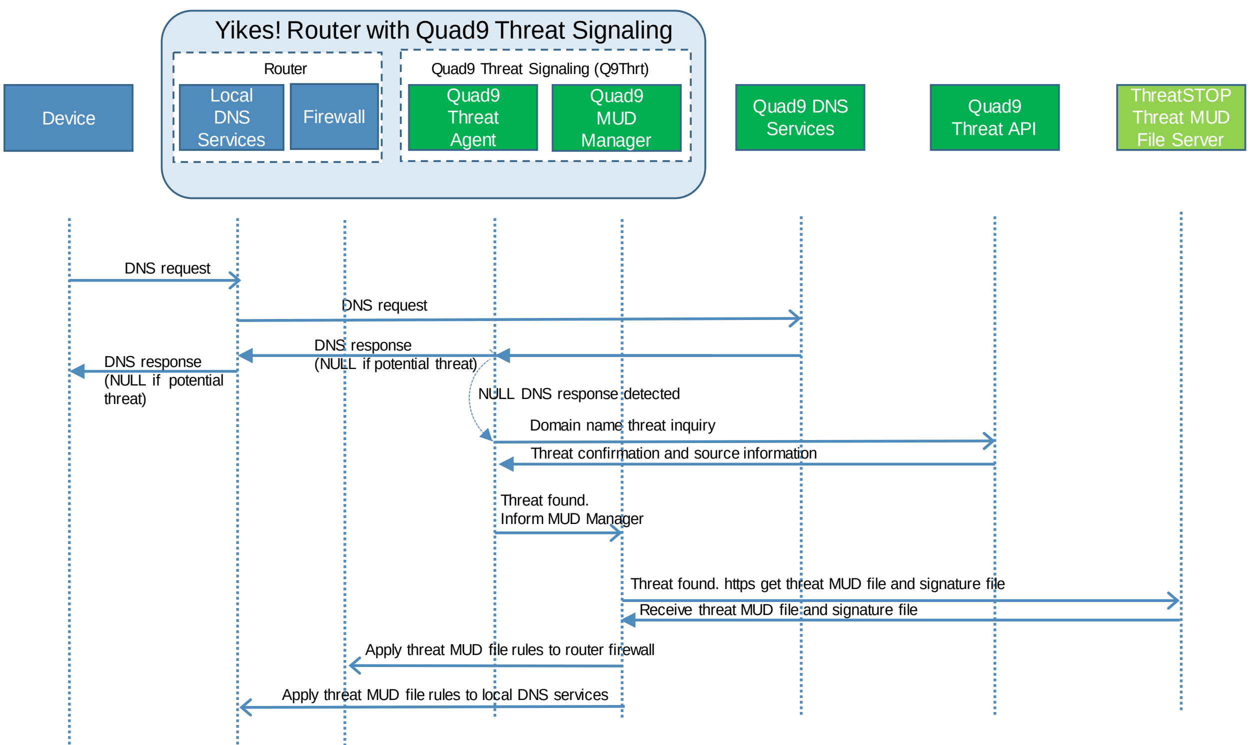
Figure 7-9 shows the message flow required to support threat signaling in Build 2.

- 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	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	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
	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. **NIST SP 800-53 Rev. 4** CM-3, CM-4, SA-10	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.**		

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-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-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 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.**	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 allow 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-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 the time of signing. Local policy has been configured to ensure that if the MUD file for a device has a signature that was signed by a**	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 the time of sign-	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		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.	ing. According to local policy, the **MUD PEP will be configured to either allow or 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-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 by local policy to allow 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 allow 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-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	Pass (for testable procedure, ingress cannot be tested due to Network Address

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.	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 **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.	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

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	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			
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. 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.	Pass
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,**	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

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		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 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.	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
	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 **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	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	Not testable in preproduction implementation

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
	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.		does fetch a new MUD file.	
IoT-11	**ID.AM-1:** Physical devices and systems within the organization are inventoried.	A **MUD-enabled IoT device is capable of emitting a MUD URL.**	Upon initialization, the MUD-enabled IoT device broad-	Pass

Test	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Test Summary	Expected Outcome	Observed Outcome
		The device should leverage one of the specified manners for emitting a MUD URL.	casts a DHCP message on the network, including at most one **MUD URL, in https scheme, within the DHCP transaction.**	

1965

1966 In addition to supporting MUD, Build 2 can identify a device's make (i.e., manufacturer) and model,
1967 categorize devices based on their make and model, and associate device categories with traffic policies
1968 that affect both internal and external traffic transmissions, as shown in Table 7-3.

1969 **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	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

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	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			
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 onboarded devices.**	Onboarded devices have been identified and categorized automatically upon being connected to the network. Using the UI, show that the make and model of a device can be modified, and that the category of the device can be assigned manually.	As expected
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.	**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**	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**	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-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	**devices of a specific make and model.**	**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 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	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)**	**Through the UI,** device category rules can be defined to **permit connectivity between categories of devices.** Set rules for **category x to**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	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 **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	**communications to other categories of devices on the local network. All traffic is enforced according to** rules associated with **the device's category.**	**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 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	The router is capable of querying 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**	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**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	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.	**are prohibited from visiting specific domains and IP addresses.**	**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 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	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**	As expected

Exercise	Applicable Cybersecurity Framework Subcategories and NIST SP 800-53 Controls	Exercise Summary	Expected Outcome	Observed Outcome
	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		**communicate to the IP address is prevented from initiating communication**.	
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 **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, resulting in the router being configured with threat 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.**	Log in to the router and verify that the firewall rules that prohibited communication to malicious domains (and that were verified as present in the previous two tests) are no longer present.	As expected

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 onboarding process. Future versions of the product may collect additional information about a device to improve the specificity of device identification.
- Yikes! mobile application:
  - At the time of demonstration, 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 the time of demonstration, 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 onboarded, 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 the time of demonstration, 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

Build 3, which is still under development, uses equipment supplied by CableLabs to support MUD. It will leverage the Wi-Fi Alliance Easy Connect specification to securely onboard devices to the network. It will also use 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 Micronets available on GitHub. CableLabs is in the process of developing and adding new features and functionality to its open-source reference implementation of Micronets.

Although limited functionality of a preliminary version of Micronets was demonstrated as part of this project, Build 3 is not yet complete and has not yet been subjected to functional evaluation or demonstration. Full documentation of Build 3 is planned for inclusion in the next phase of this project. In the remainder of this section we provide a brief preview of the architecture and functional elements planned for Build 3. A more detailed description of Micronets can be found in CableLabs' Micronets white paper.

8.1 Collaborators

Collaborators currently participating in this build are described briefly in the subsections below. More collaborators may be added once the build is completed.

8.1.1 CableLabs

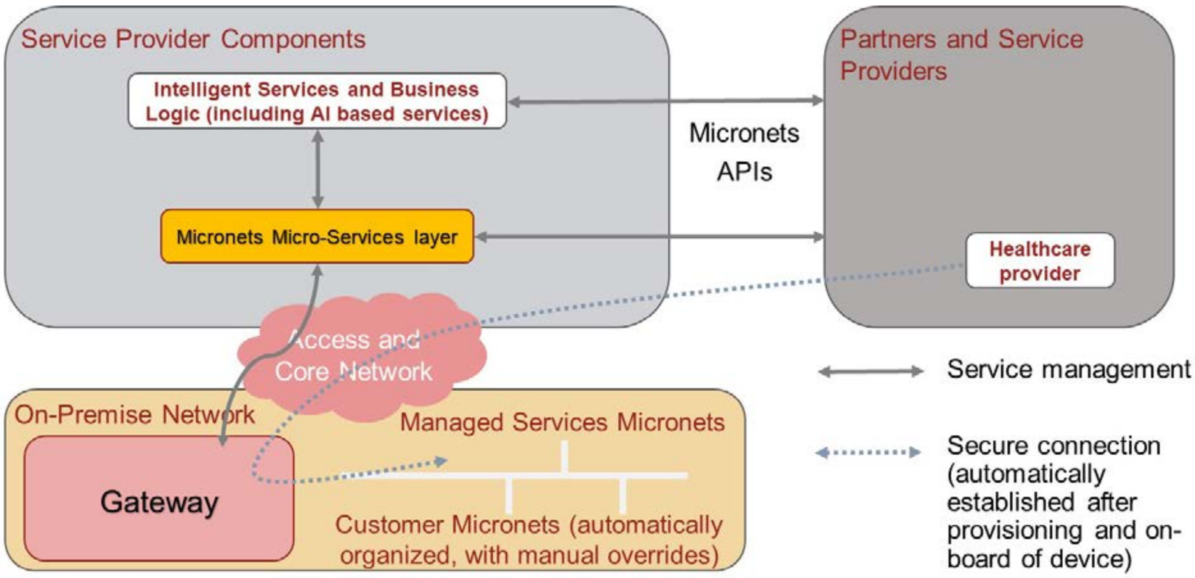
CableLabs is a nonprofit product innovation and research and development enterprise in the cable industry. It includes more than 60 cable-network-operator members around the world, representing approximately 180 million subscribers and roughly 500 million individuals. In November 2018, CableLabs publicly announced Micronets, a next-generation on-premise network platform focused on providing adaptive security for all devices connecting to a residential or small-business network 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 on-premise network platform. 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.

8.2 Micronets Architecture

As illustrated in Figure 8-1 and described in more detail in the subsections below, Micronets' logical architecture currently consists of the following components:

- Intelligent Services and Business Logic layer (e.g., machine-learning-based services), which resides in the cloud and is operated by the service provider
- Micronets Micro-Services layer (e.g., SDN controller, Micronets Manager, MUD manager), which also resides in the cloud and is operated by the service provider. The most important component of this layer is the Micronets Manager, which coordinates the entire state of the Micronets-enabled on-premises network.
- On-premises Micronets, 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 be used to group devices together into trust domains and isolate them from other devices.
- Micronets APIs allow partners and service providers to interface with a customer's micro-networks environment to provision and deliver specific customer-requested services.

Figure 8-1 Logical Architecture—Build 3



8.2.1 Intelligent Services and Business Logic

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 combines them into operational decisions that are handed over to the Micronets micro-services for execution. It may receive information from various Micronets' micro-services (such as the SDN controller) and in turn use that information to dynamically update the access rules for connected IoT devices. For example, to support devices that do not emit a MUD URL, a "synthetic" MUD file generator and MUD server may be provided that can host crowdsourced MUD files that are provided to the Micronets micro-services. Another example is an IoT fingerprinting service that could allow detection of devices in the network or an artificial intelligence/machine-learning-based malware detection service that can provide updated MUD files or access policies based on actively detected threats in the network.

8.2.2 Micronets Micro-Services

The Micronets Micro-Services layer hosts several network management-related micro-services that interact with the on-premises 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. Several micro-services are engaged and managed by the Micronets Manager, including the SDN controller, DHCP/DNS manager, AAA (RADIUS) server, and MUD manager.

8.2.3 On-Premises Micronets

The Micronets Gateway is responsible for creating and enforcing the Micronets on the home/small-business network. Each Micronet represents a distinct trust domain and at the minimum represents a distinct IP subnet. IoT devices that are not permitted to exchange traffic with other IoT devices will be placed in separate Micronets to isolate them from each other. The Micronets Gateway is also an SDN-capable switch that is controlled by the SDN controller that is part of the Micronets Micro-Services layer in the cloud. The Micronets Gateway is integrated with a Wi-Fi access point, but it supports both wired and wireless connectivity.

8.2.3.1 *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. In Build 3, a MUD-based policy will drive the assignment of devices to specific Micronets.

8.2.3.2 *Customer Micronets*

Customers acquire and connect their own devices. They may even integrate entire service-oriented networks, such as a smart 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.

8.2.4 Micronets API Framework

Each 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 provision and deliver specific services that the customer has requested.

8.3 Build 3 Use Case

Build 3 is expected to make use of the following elements:

- a Micronets Gateway and access point to be located on premises at the home/small-business network
- a cloud-based Micronets Manager, SDN controller, identity server, and RADIUS server dedicated to the home/small-business network
- the service provider's cloud-based infrastructure that includes a proxy for the cable service operator, an authentication server, and a MUD manager
- an offsite onboarding clinic that includes a registration server and a MUD file server that holds versions of MUD files that have been customized by the onboarding clinic

Build 3 is expected to use the above components in combination to support MUD. Build 3 is expected to differ from the other builds in this project insofar as it plans to perform device onboarding at an onboarding clinic that is separate from the home/small-business network. Under this paradigm, the MUD file rules will be installed on the home/small-business network's Micronets Gateway during the onboarding process before the device connects to the home/small-business network. Later, when the device connects to the home/small-business network, the MUD rules will already be in place.

The off-premises onboarding clinic is expected to be equipped with a registration server that will associate each device with a version of its MUD file that has been customized by the onboarding clinic. This registration server will invoke the service provider's infrastructure and the home/small-business network's cloud infrastructure to provision a certificate onto the device. This certificate will enable the device to be authenticated and associated with its MUD file traffic profile upon connection to the home/small-business network. The on-premises Micronets Gateway, which is connected to the cloud, will be configured by the MUD manager with the device's MUD file rules during the onboarding process. Later, when the device connects to the home/small-business network, the Micronets Gateway will already be configured to enforce MUD-based traffic constraints for that device based on the certificate that had been provisioned onto the device during its registration process at the offsite onboarding clinic. The Micronets Gateway is also expected to be designed to support dynamic micro-segmentation and incorporate device identity and fingerprinting techniques to enable real-time detection and quarantining of compromised IoT devices.

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/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. 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 to send flow rules to the SDN switch.	Provides ID.AM-3 PR.PT-3
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	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

Component	Product	Function	Cybersecurity Framework Subcategories
		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).	
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
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

Component	Product	Function	Cybersecurity Framework Subcategories
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
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.	ID.AM-1

Component	Product	Function	Cybersecurity Framework Subcategories
		Its traffic is unrestricted.	
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.

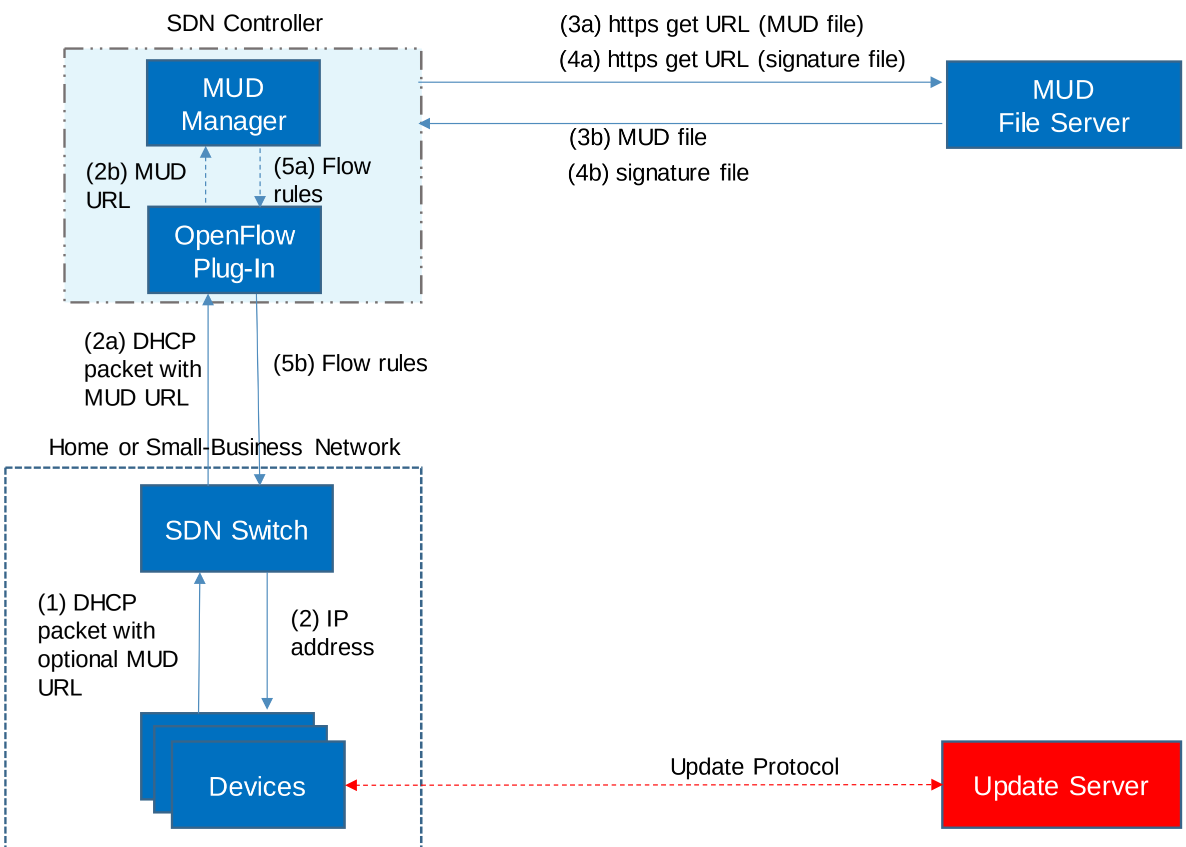
9.3 Build Architecture

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:

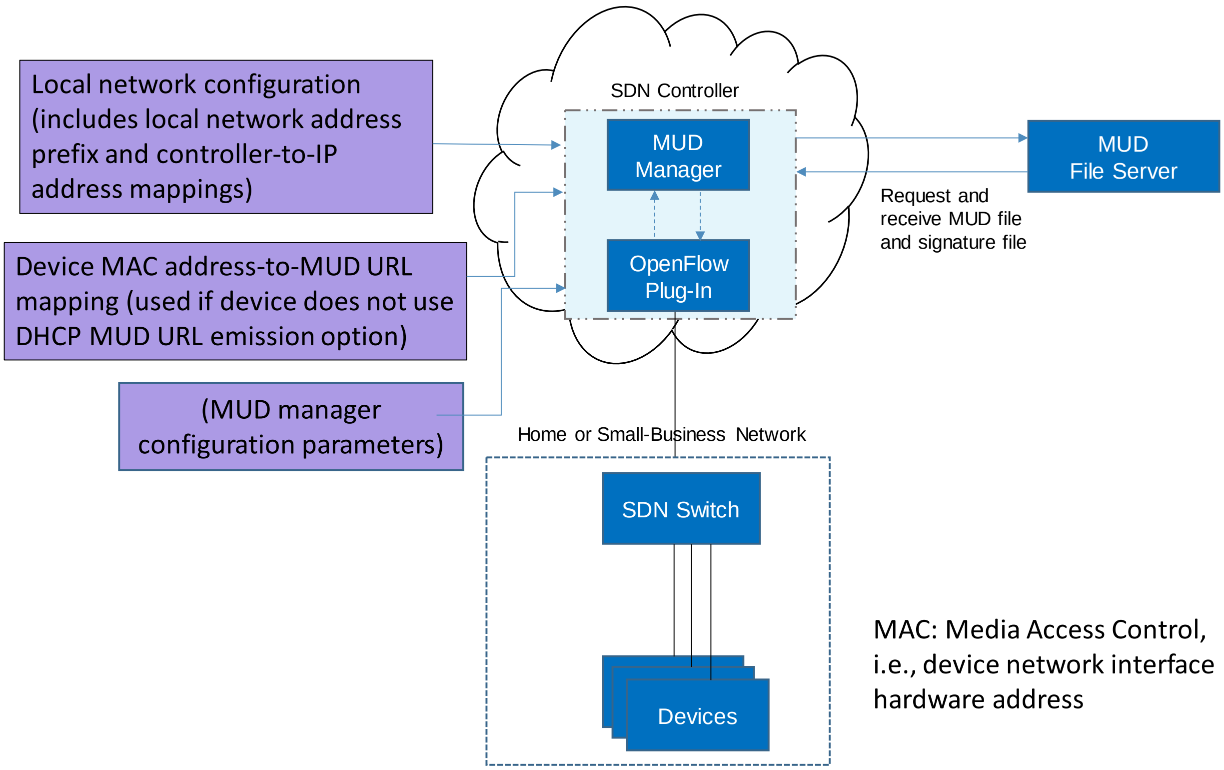
- Upon connecting a MUD-capable device, the MUD URL is emitted via DHCP (step 1).

- 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:

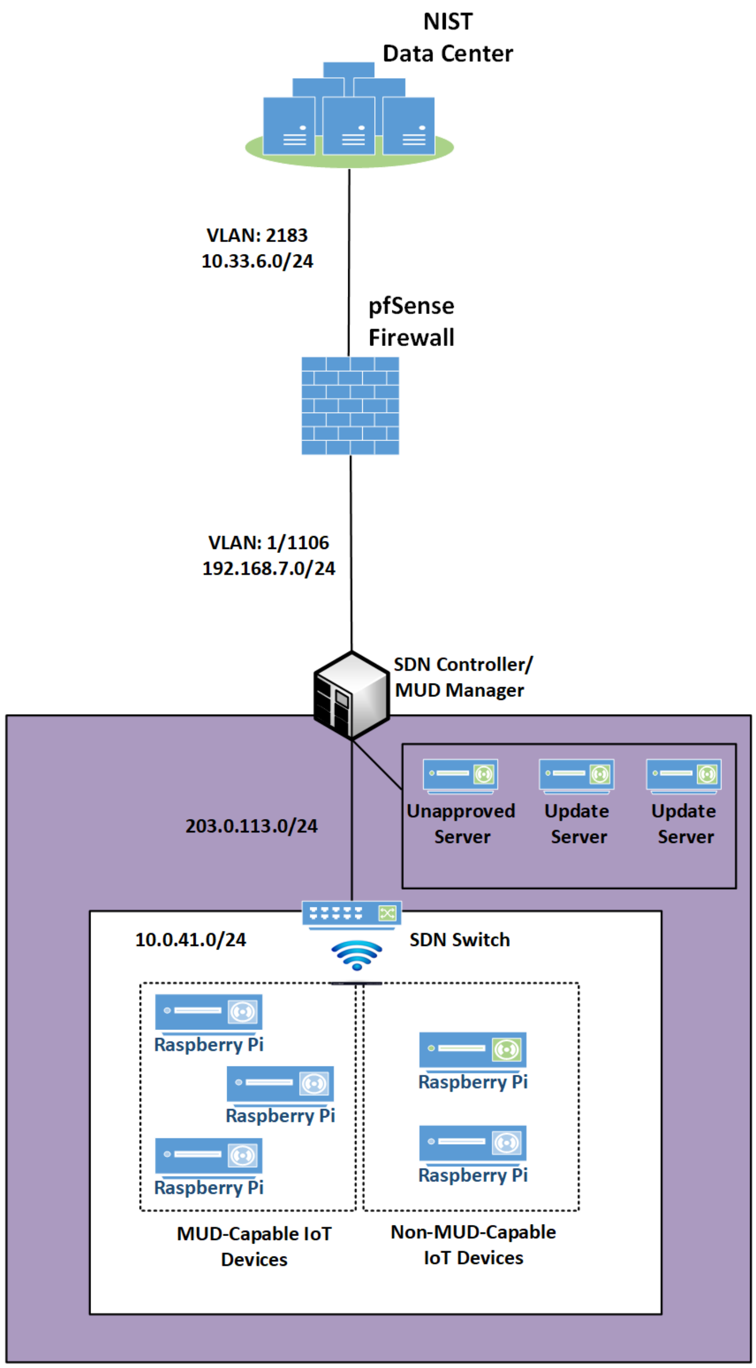
- application configuration—This allows the network administrator to define parameters for the application. The SDN controller/MUD manager must be provided with configuration information for the home and small-business networks that it manages. In addition, configuration parameters for the MUD manager 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.

2338 **Figure 9-3 Physical Architecture—Build 4**



2339

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:

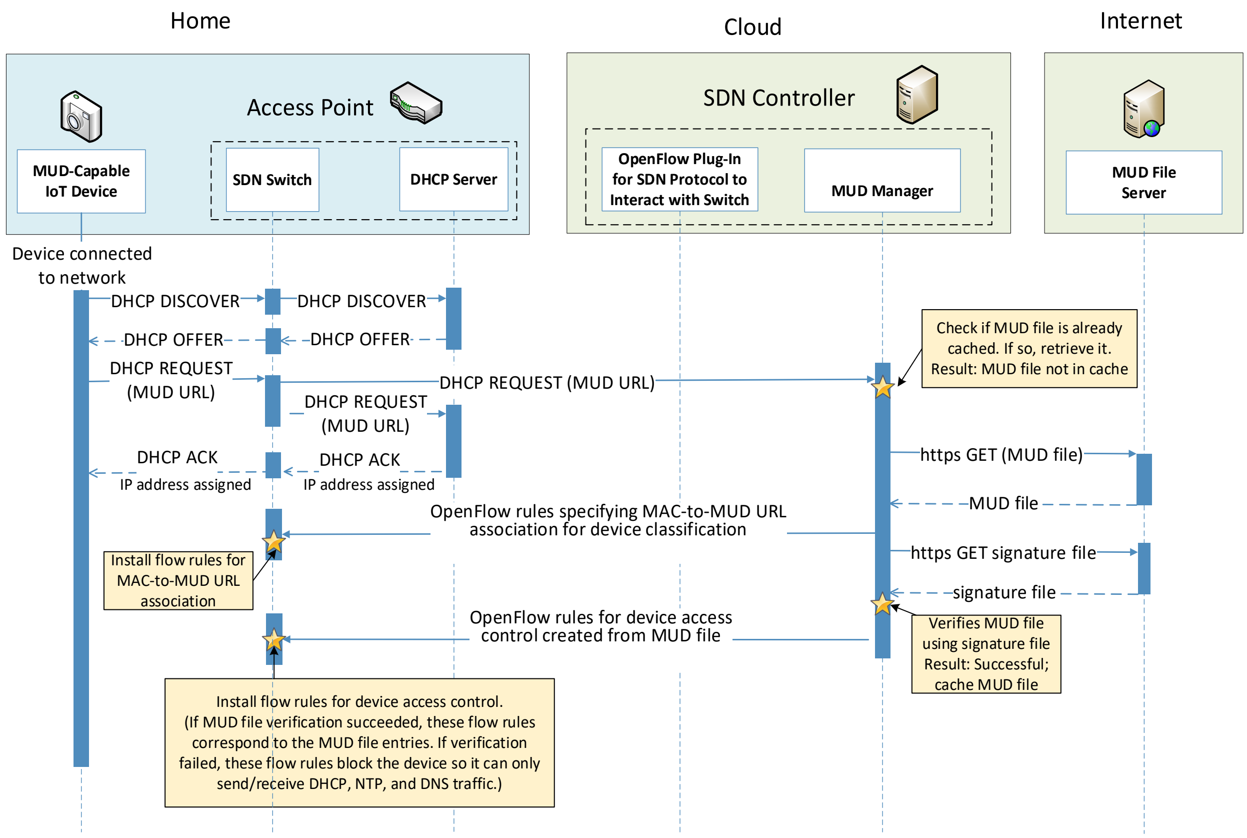
- Table 0, Source MAC address classification table, classifies a packet based on its source IP/MAC address.
- Table 1, Destination MAC address classification table, classifies a packet based on its destination IP/MAC address.
- Table 2, From-Device flow rules table, associates ACEs with the packet based 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 *Onboarding 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.

2375 **Figure 9-4 MUD-Capable IoT Device Onboarding Message Flow—Build 4**



2376

2377 As shown in Figure 9-4, the message flow is as follows:

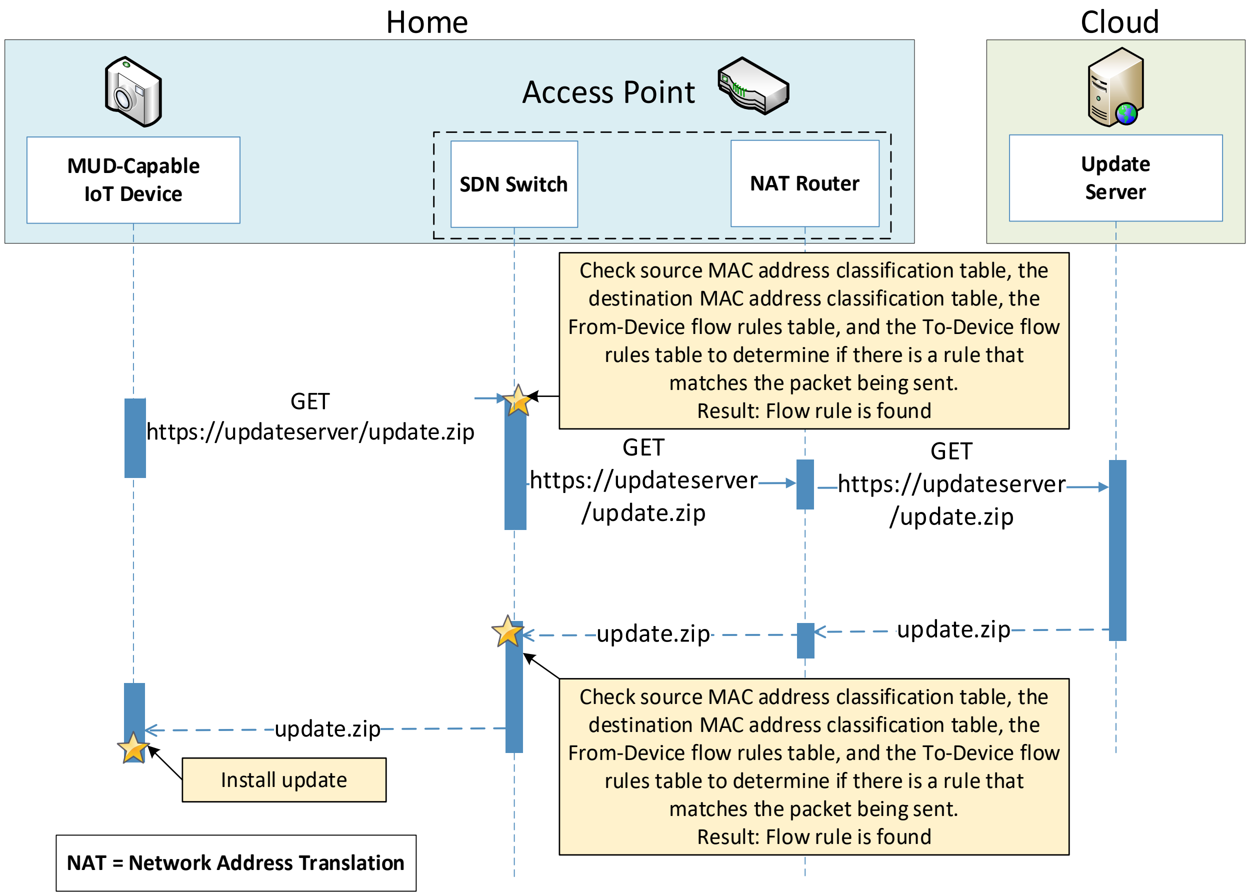
2378 - The IoT device sends out a DHCP DISCOVER message to the SDN switch.
2379 - The AP resident DHCP server sends back a DHCP offer that gets sent back to the device via the
2380 SDN switch.
2381 - The device then sends out a DHCP request containing the MUD URL, which gets sent
2382 simultaneously to the AP resident DHCP server by the SDN switch and to the MUD manager.
2383 - The AP resident DHCP server sends an IP address to the device in a DHCP ACK message via the
2384 switch.
2385 - Based on the MUD URL presented in the DHCP request, the MUD manager checks to see if the
2386 corresponding MUD file is already cached. In the example depicted, the MUD file is not in the
2387 cache.
2388 - 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.

2403 **Figure 9-5 Update Process Message Flow—Build 4**



2404

2405

2406 As shown in Figure 9-5, the message flow is as follows:

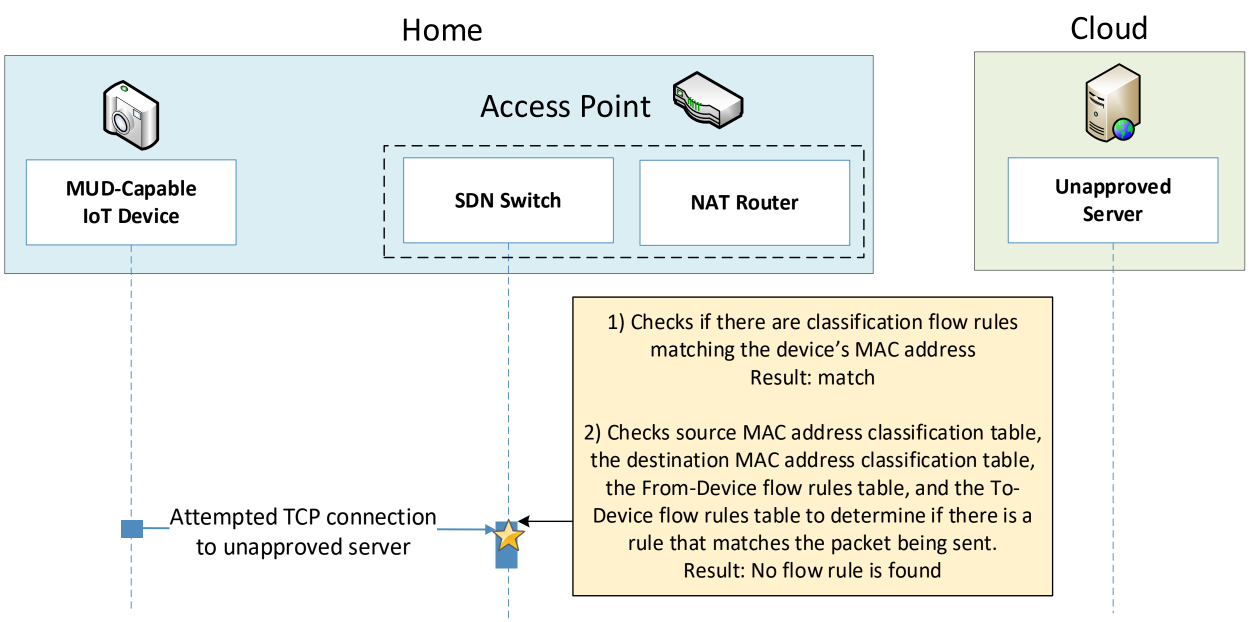
2407 - The device generates an https GET request to its update server.
2408 - The SDN switch will consult its flow rules for this device to verify that it is permitted to send
2409 traffic to the update server. Assuming there were explicit rules in the device's MUD file
2410 enabling it to send messages to this update server, the SDN switch will forward the request to
2411 the NAT router, which will then forward it to the update server.
2412 - The update server will respond with a zip file containing the updates.
2413 - The return traffic will be sent via the NAT router to the switch.
2414 - The destination MAC address of the packet identifies the device, and appropriate metadata is
2415 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:
    - o <<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*

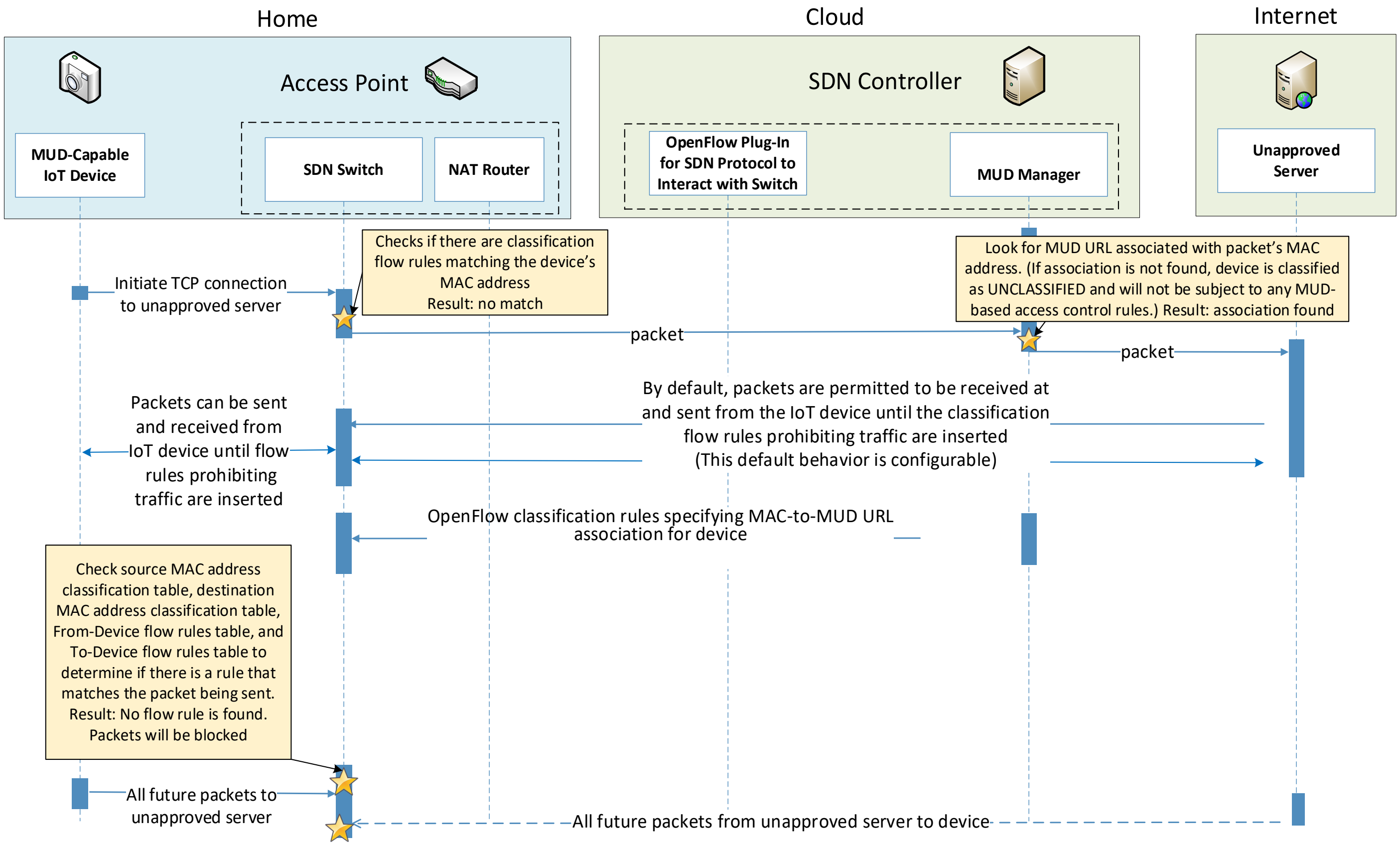
Insertion of flow rules onto the SDN switch on the home/small-business network is dynamic. Rules are computed at the SDN controller/MUD manager and installed on the SDN switch. Flow rules are configured to time out on inactivity to avoid having the SDN switch's flow table fill up. (If an IoT device disconnects 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.

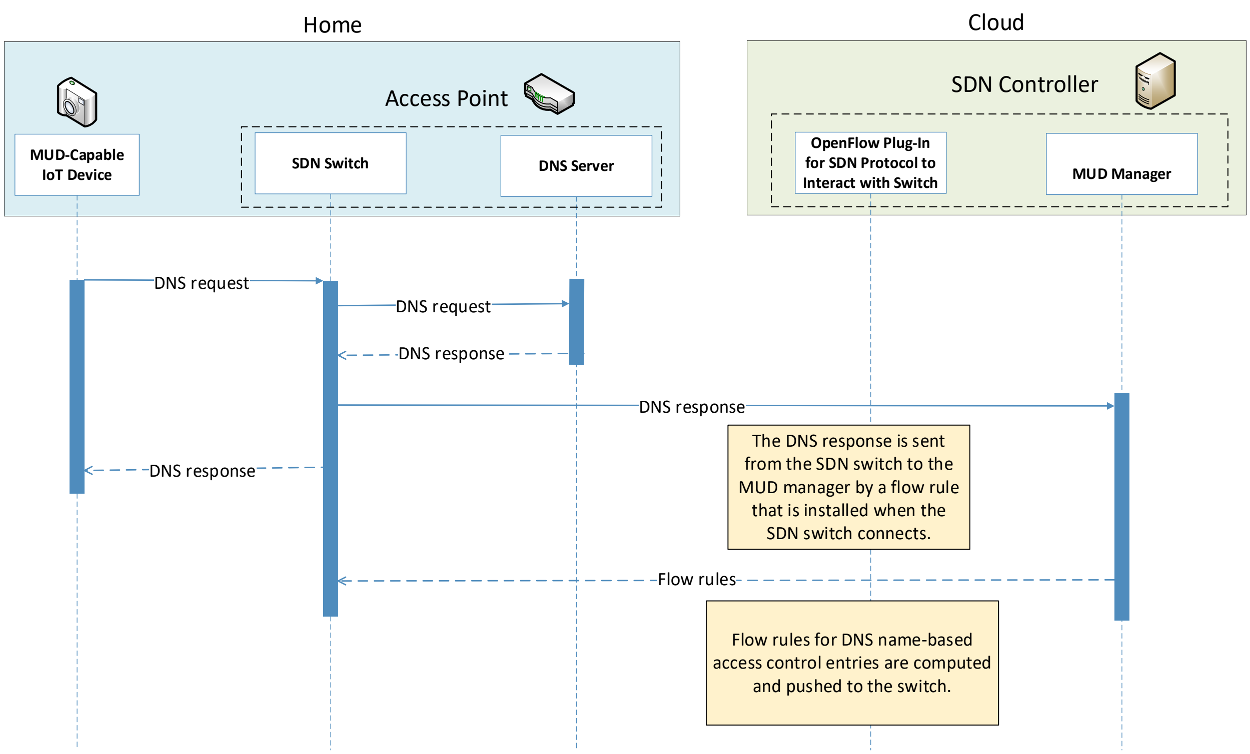
9.3.3.5 *DNS Events*

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. **NIST SP 800-53 Rev. 4** CM-8, PM-5	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 serves the MUD file to the MUD manager. The	Upon connection to the network, the MUD-enabled 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
	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.	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.**		

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 the time of signing. Local policy has been configured to ensure that if the MUD file for a device has a signature**	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
		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.	at the time of 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 is capable of emitting 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 device onboarding. A device may “lie” about its identity by issuing a spurious DHCP request with a MUD URL embedded. There are no certificate-based onboarding checks.

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 devices that have a clear purpose and whose communication needs can be clearly defined. These communication needs are defined in terms of not only what ports and protocols the devices are permitted to use, but also the destinations with which the IoT devices can use those ports and protocols to communicate. If a device is not special-purpose and 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 components on the internal network that are not from authorized manufacturers or authorized device classes
    - preventing the MUD-capable IoT device from being used to access unauthorized external domains
    - preventing the MUD-capable IoT device from being used to access other components on the internal network that are not from authorized manufacturers or that are not authorized device types
- 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 non-MUD-capable 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 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 is a plug-and-play solution that is 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 easily and seamlessly deploy these implementations. 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's certificate. In either of these cases, if the default behavior that the device is configured to perform is not acceptable, the user would need to configure the device to perform the desired behavior. Ideally the device would provide a user-friendly interface through which to do so.
- 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.

- 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 address to which the domain name gets resolved matches 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 would 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 colocated, 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 that is returned to the IoT device when it performs a domain name lookup should be the same as the IP address that was 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

- There are numerous ways in which support for MUD can be provided within a home/small-business network. Build 3 is expected to demonstrate support for MUD in residential gateway equipment and 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 the upstream 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 upstream 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 entity controls the MUD manager will have access to this information. If this entity is a service provider, as in the planned 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 implementing 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, and so by its very nature can be disruptive if used by unauthorized parties." Protecting the MUD infrastructure includes ensuring the security of the IoT device MUD URL emission, the MUD manager, the DHCP server, 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 file for their IoT devices to ensure it specifies communications that are appropriate for the 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 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 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 enable 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 be used to provide strong authentication of 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 does the MUD build 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 nonauthenticated 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), or future authentication types
    - remove state if an unauthenticated method of MUD URL emission is being used and any form of break in that session is detected
    - not include unauthenticated devices into the manufacturer grouping of any specific manufacturer without additional validation
    - 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
    - raise an alert and require administrator approval if the MUD manager detects that the signer of a MUD file has changed, in order to protect against rogue Certificate Authorities
    - raise an alert and require administrator approval if the MUD manager detects that a device's MUD file has changed, in order 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 a MUD URL by default.
- 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, and MUD-capable routers that are 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 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. In small-business and home environments it may not be possible to apply good segmentation policies. But at a minimum, where there are IoT devices that are known to have security risks, e.g., non-MUD-capable devices, keep these on a separate network segment from the everyday computing devices that are afforded with 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 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.
- 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 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.
- 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.

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, additional mechanisms for devices to securely provide their MUD URL, such as use of the Wi-Fi Device Provisioning 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
API	Application Programming Interface
CIS	Center for Internet Security
CMS	Cryptographic Message Syntax
CoAP	Constrained Application Protocol
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
FTP	File Transfer Protocol
GCA	Global Cyber Alliance
GUI	Graphical User Interface
http	Hypertext Transfer Protocol
https	Hypertext Transfer Protocol Secure
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
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
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
RADIUS	Remote Authentication Dial-In User Service
REST	Representational State Transfer
RFC	Request for Comments
RMF	Risk Management Framework
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
SP	Special Publication
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
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
WAN	Wide Area Network
WPA3	Wi-Fi Protected Access 3 Security Certificate protocol
YANG	Yet Another Next Generation

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 is capable of emitting 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 new device gains access to the wired or wireless network for the first time
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. 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 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/.

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

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

November 2019

PRELIMINARY 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

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 1800-15C, Natl. Inst. Stand. Technol. Spec. Publ. 1800-15C, 113 pages, (November 2019), 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: November 21, 2019 through January 21, 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

botnets; Internet of Things; IoT; Manufacturer Usage Description; MUD; router; server; software update server; threat signaling.

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
Ashwini Kadam	CableLabs
Craig Pratt	CableLabs
Darshak Thakore	CableLabs
Mark Walker	CableLabs
Tao Wan	CableLabs

Name	Organization
Russ Gyurek	Cisco
Peter Romness	Cisco
Brian Weis	Cisco
Rob Cantu	CTIA
Dean Coclin	DigiCert
Clint Wilson	DigiCert
Katherine Gronberg	Forescout
Tim Jones	Forescout
Rae'-Mar Horne	MasterPeace Solutions
Nate Lesser	MasterPeace Solutions
Tom Martz	MasterPeace Solutions
Daniel Weller	MasterPeace Solutions
Mo Alhroub	Molex
Jaideep Singh	Molex
Bill Haag	NIST
Bryan Dubois	Patton Electronics

Name	Organization
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
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
John Bambenek	ThreatSTOP

Name	Organization
Paul Watrobski	University of Maryland
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 Service provider server Partner and service provider server Prototype medical devices–Raspberry Pi
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 three 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:

- NIST SP 1800-15A: *Executive Summary*
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics*–what we built and why
- NIST SP 1800-15C: *How-To Guides*–instructions for building the example solutions **(you are here)**

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 three 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 commercial 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 three 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 patches are installed promptly

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

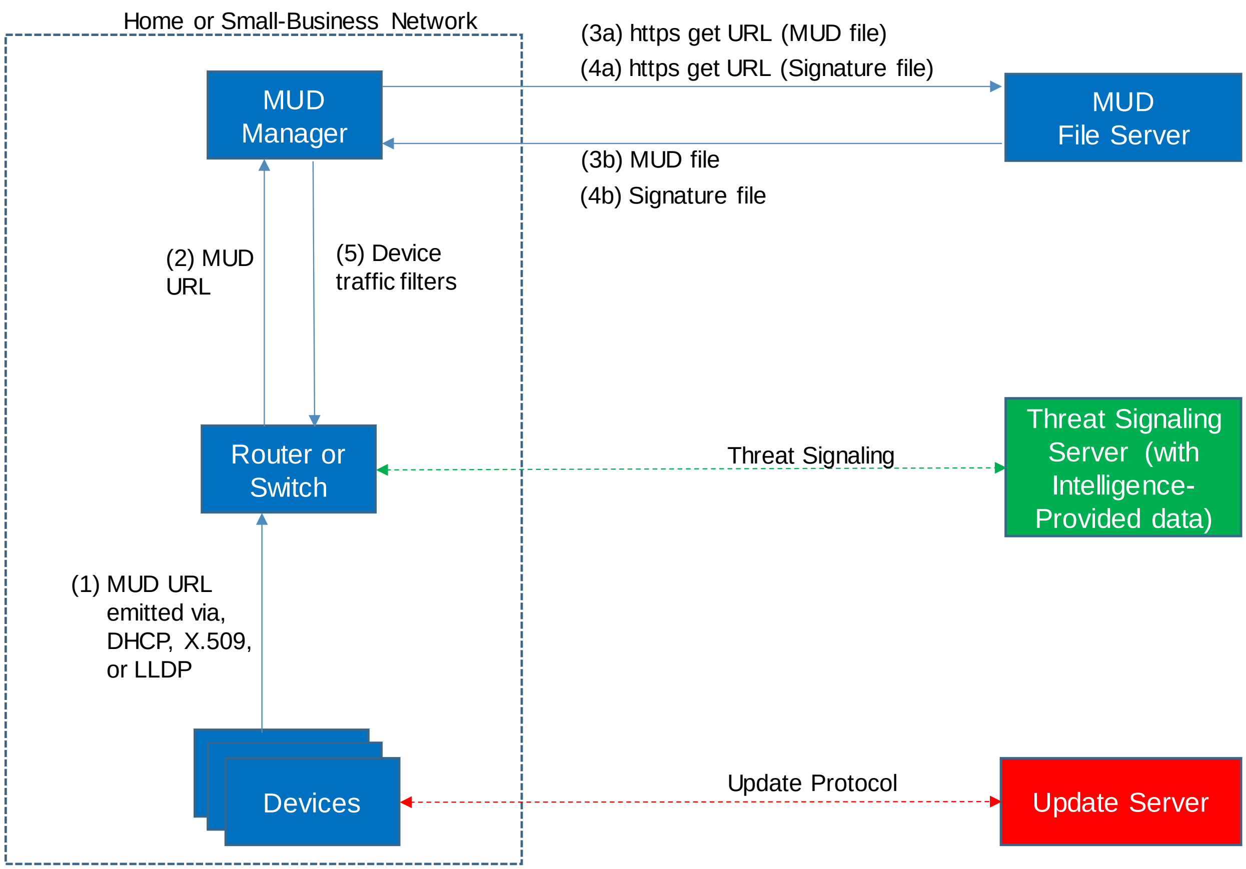
1.2.1 Usage Scenarios

Each of the three 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 three 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 smart 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 smartphones, two smart lighting devices, a smart assistant, a smart printer, a baby monitor with remote control and video and audio capabilities, a smart wireless access point, and a smart digital video recorder. The cameras, smart lighting devices, baby monitor, and digital video recorder are all controlled and managed by a smartphone. 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 three 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 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.

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 three builds support MUD and the ability to receive faux updates from a notional update server, only Build 2 currently supports threat signaling. In addition, 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.

Briefly, the four builds of the reference architecture that have been undertaken, three of which are complete and have been 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 to onboard devices and support MUD. Although limited functionality of a preliminary version of this build was demonstrated as part of this project, Build 3 is still under development. Therefore, it is not documented in this practice guide.
- 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 RFC to demonstrate feasibility and scalability. Certificates from DigiCert are also used.

The logical architectures and detailed descriptions of Builds 1, 2, 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.

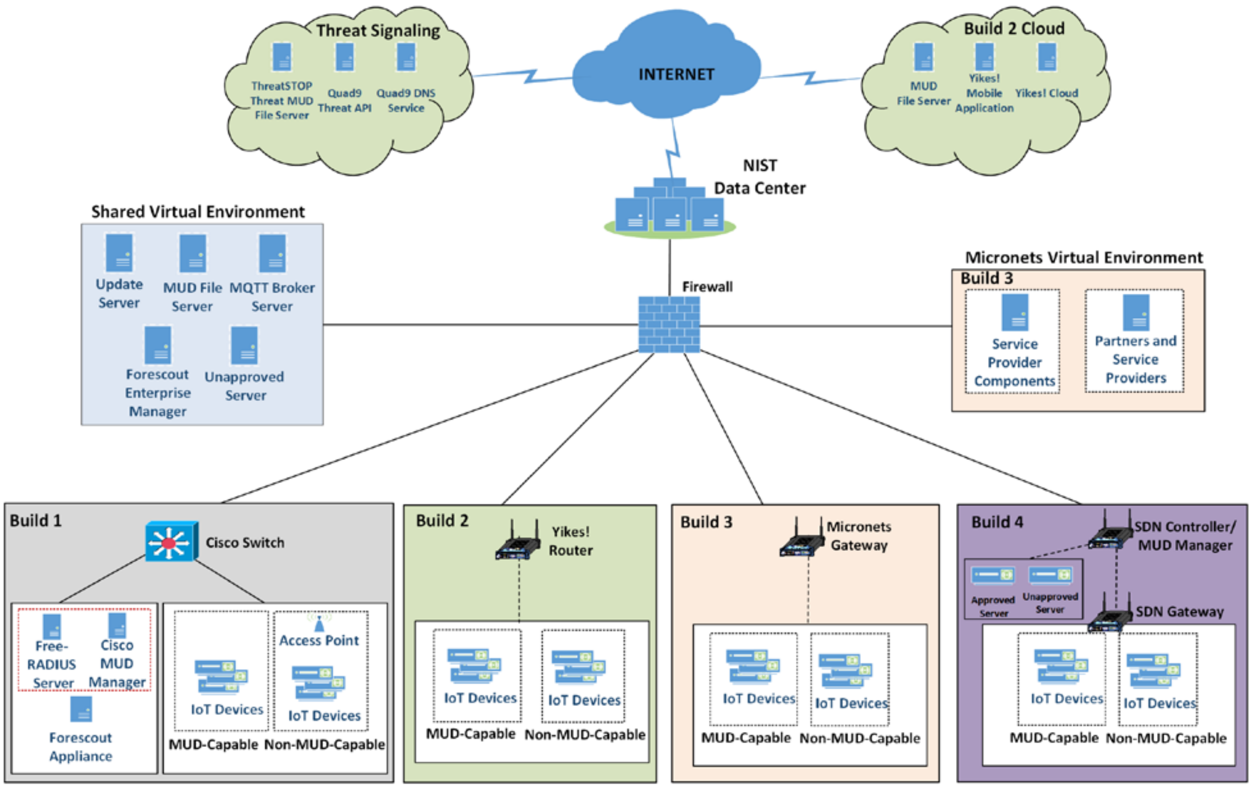
The following four builds, as depicted in the diagram, are supported within the physical architecture:

- 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 is still under development and is expected to be completed by the next phase of this project. As of this writing, Build 3's 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.
- 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, smartphones, lighting devices, a smart 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, and 4.

409 **Figure 1-2 NCCoE Physical Architecture**

410 

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 of 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 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
   ```

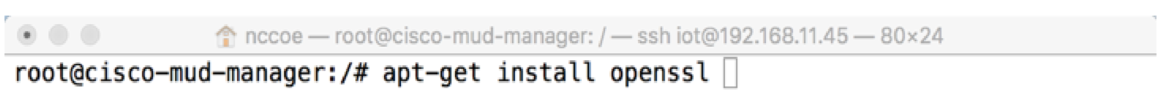
2. Change to the root directory:

   ```
   cd /
   ```

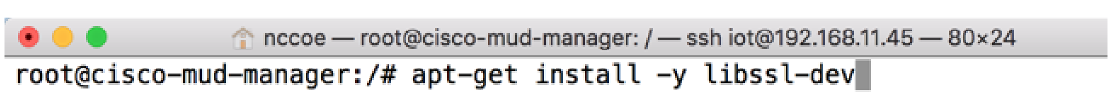
3. To install OpenSSL from the terminal, enter the following command:

   ```
   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:

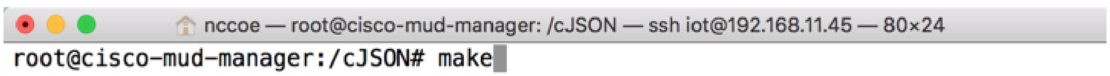
```
git clone https://github.com/DaveGamble/cJSON
```

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

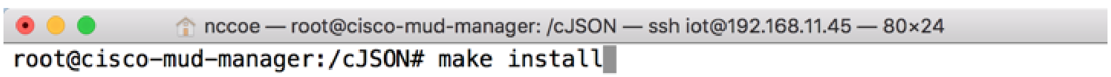
```
cd cJSON
```

   b. Build cJSON by entering the following commands:

```
make
```

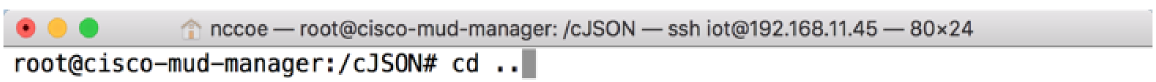



```
make install
```



5. Change directories back a folder by entering the following command:

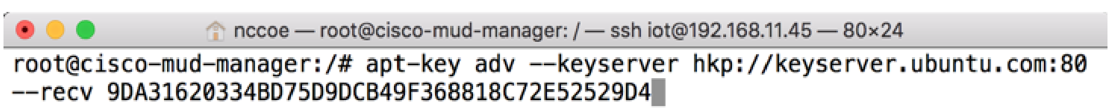
```
cd ..
```



6. To install MongoDB, enter the following commands:

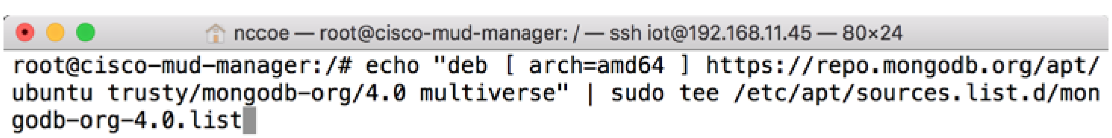
   a. Import the public key:

```
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
```

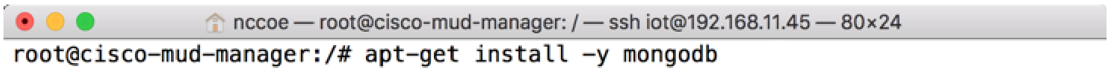


c. Reload the local package database:

```
apt-get update
```

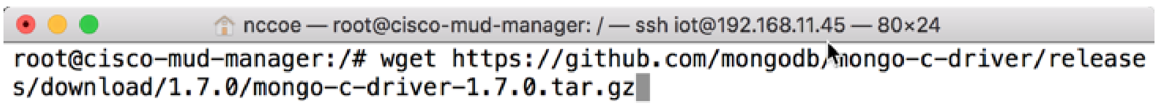
d. Install the MongoDB packages:

```
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
```

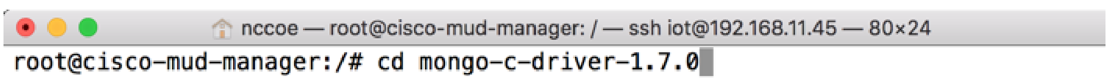


a. Untar the file by entering the following command:

```
tar -xzf mongo-c-driver-1.7.0.tar.gz
```

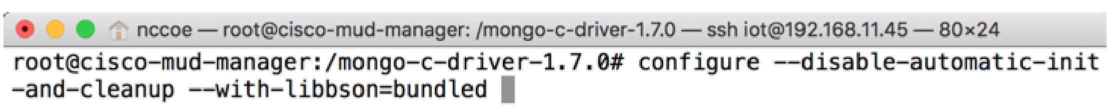
b. Change into the mongo-c-driver-1.7.0 directory by entering the following command:

```
cd mongo-c-driver-1.7.0/
```

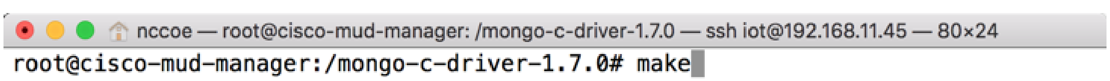


c. Build the Mongo C driver by entering the following commands:

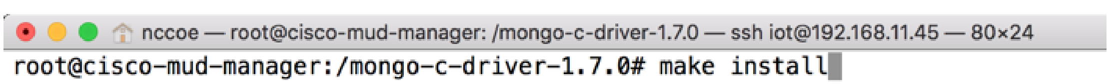
```
./configure --disable-automatic-init-and-cleanup --with-libbson=bundled
```




```
make
```

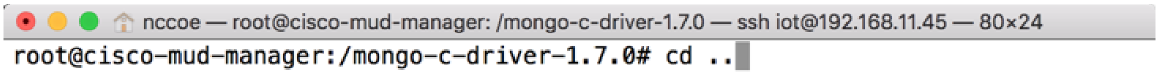



```
make install
```



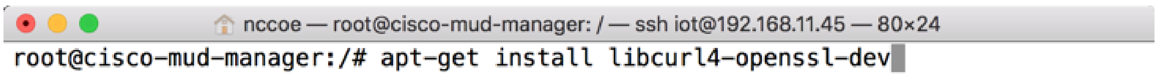
8. Change directories back a folder by entering the following command:

```
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:

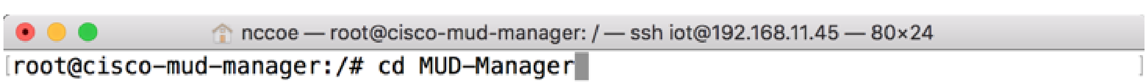
```
cd /
```

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
```

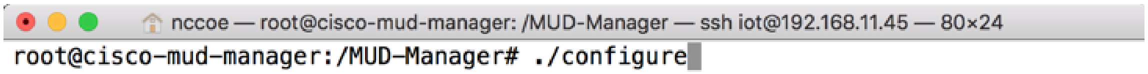
4. Change into the MUD manager directory:

```
cd MUD-Manager
```



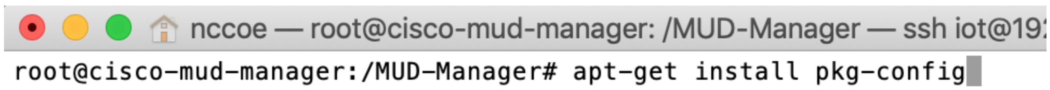
5. Build the MUD manager by entering the following commands:

```
./configure
```

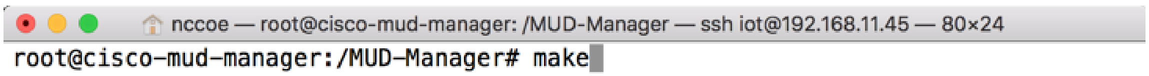


Note: If a "pkg-config error" is thrown, run the command below to install the missing package:

```
apt-get install pkg-config
```

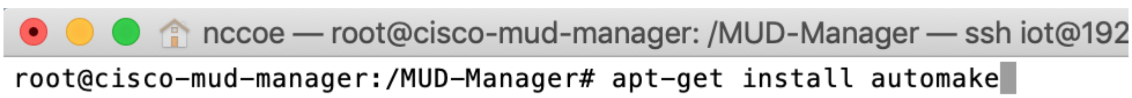



```
make
```

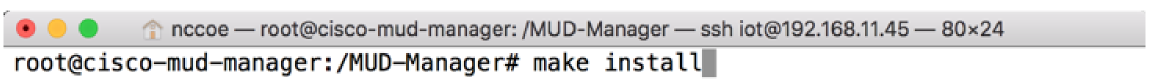


Note: If an "ac.local error" is thrown, run the command below to install the missing package:

```
apt-get install automake
```




```
make install
```



2.1.3.3 MUD Manager Configuration

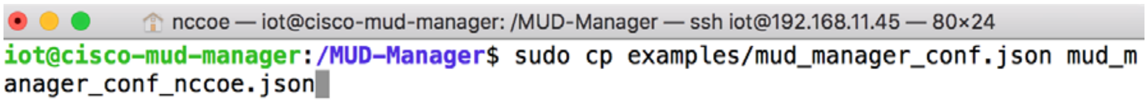
This section describes configuring the MUD manager to communicate with the NCCoE MUD file server and defining the attributes used for translating the fetched MUD files. Details about the configuration file and additional fields that can be set within this file can be accessed here: https://github.com/CiscoDevNet/MUD-Manager#editing-the-configuration-file.

1. In the terminal, change to the MUD manager directory:

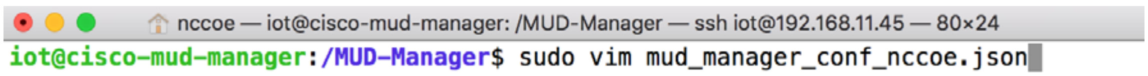
   ```
   cd /MUD-Manager
   ```

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",
```

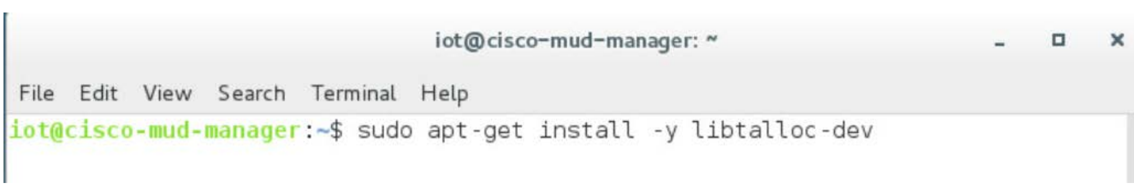
```
                "www.trytechy.com" : "99.84.104.21"
        },

        "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.

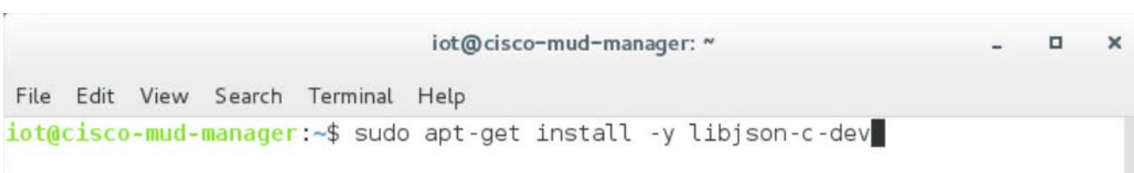
2.1.3.4 FreeRADIUS Installation

1. Install the dependencies for FreeRADIUS:

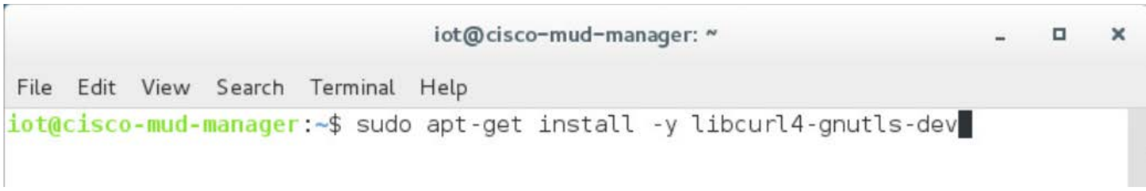
   a. `sudo apt-get install -y libtalloc-dev`



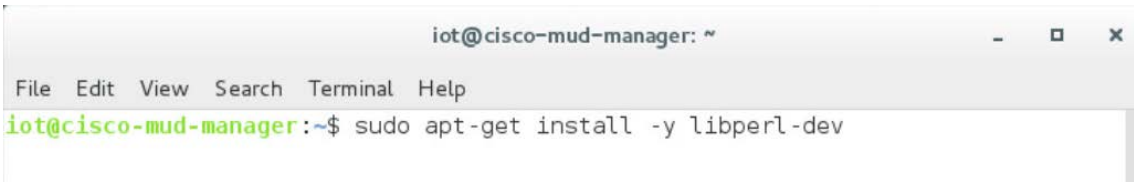
   b. `sudo apt-get install -y libjson-c-dev`



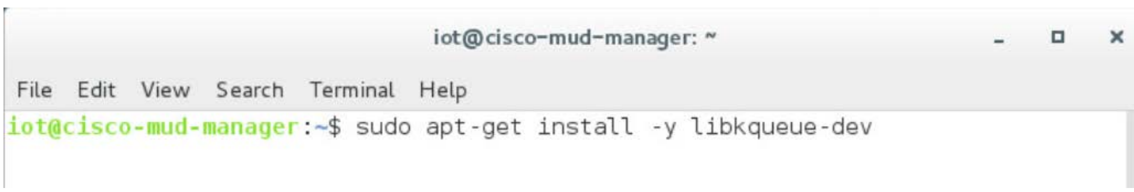
   c. `sudo apt-get install -y libcurl4-gnutls-dev`



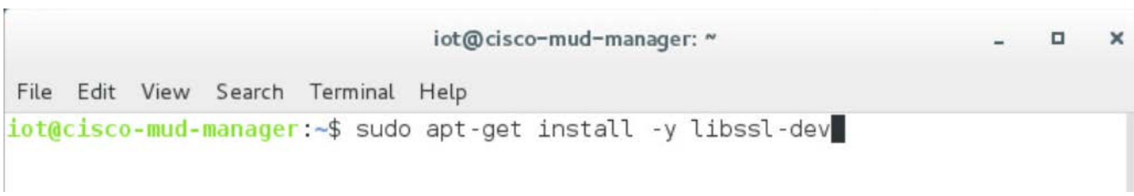
d. `sudo apt-get install -y libperl-dev`



e. `sudo apt-get install -y libkqueue-dev`

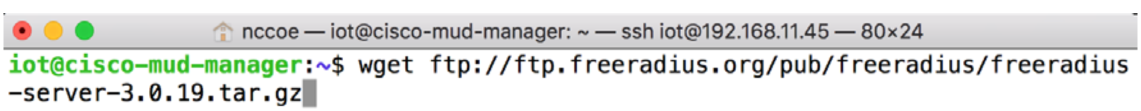


f. `sudo apt-get install -y libssl-dev`



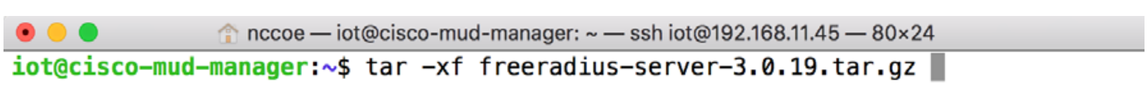
2. Download the source by entering the following command (Note: Version 3.0.19 and later are recommended):

   `wget` ftp://ftp.freeradius.org/pub/freeradius/freeradius-server-3.0.19.tar.gz



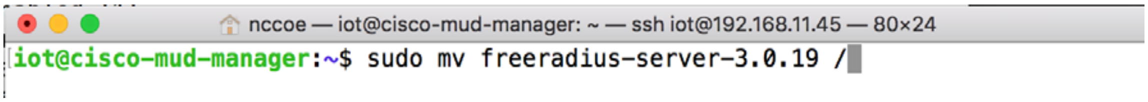
3. Untar the downloaded file by entering the following command:

   `tar -xf freeradius-server-3.0.19.tar.gz`



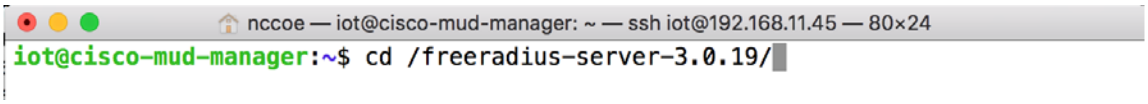
4. Move the FreeRADIUS directory to the root directory:

   `sudo mv freeradius-server-3.0.19/ /`



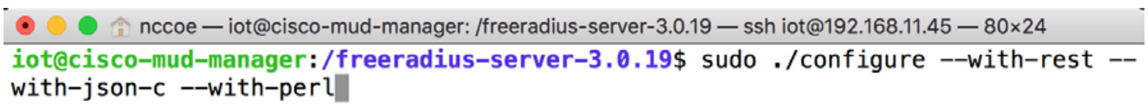
5. Change to the FreeRADIUS directory:

   `cd /freeradius-server-3.0.19/`

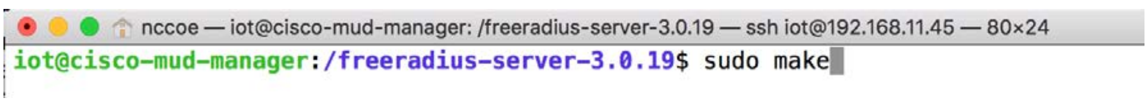


6. Make and install the source by entering the following:

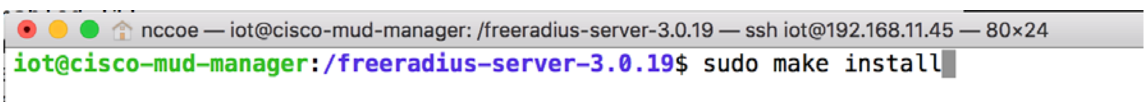
   a. `sudo ./configure --with-rest --with-json-c --with-perl`



   b. `sudo make`



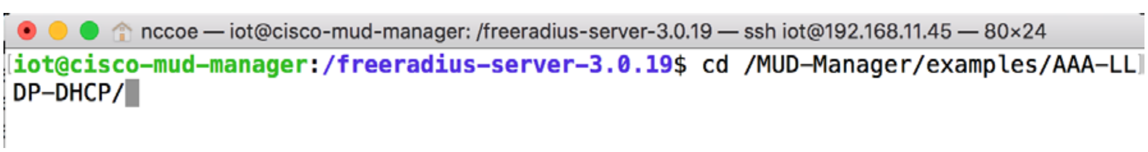
   c. `sudo make install`



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:

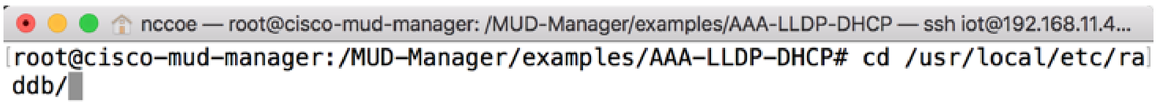
   `sudo ./FR-setup.sh`

3. Enter the following command to log in as root:

```
sudo su
```

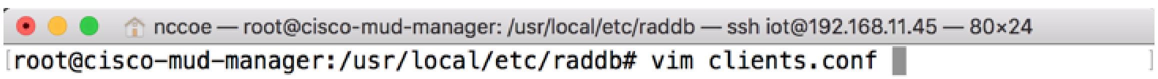
4. Change to the radius directory:

```
cd /usr/local/etc/raddb/
```



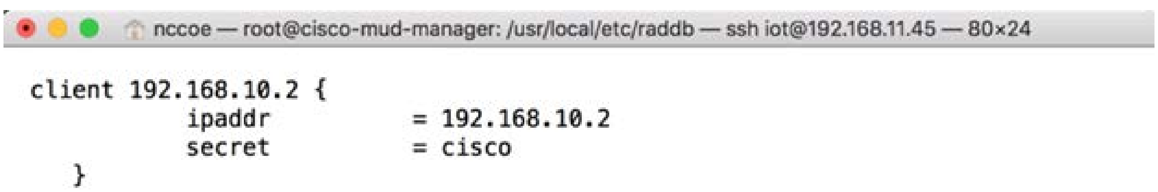
5. Open the *clients.conf* file:

```
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
   }
```

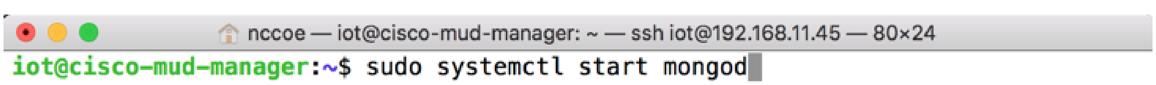


7. Save and close the file.

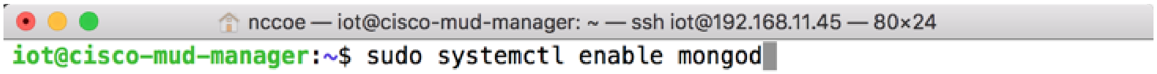
2.1.3.6 Start MUD Manager and FreeRADIUS Server

1. Start and enable the database by executing the following commands:

```
sudo systemctl start mongod
```

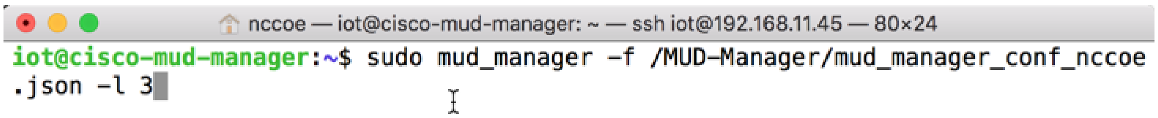



```
sudo systemctl enable mongod
```

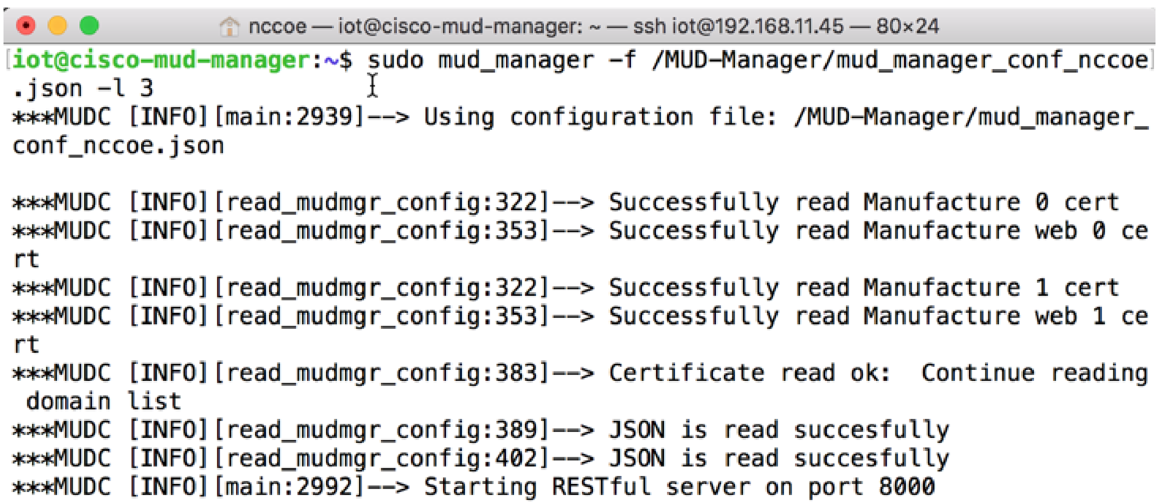


2. Start the MUD manager in the foreground with logging enabled by entering the following command:

```
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

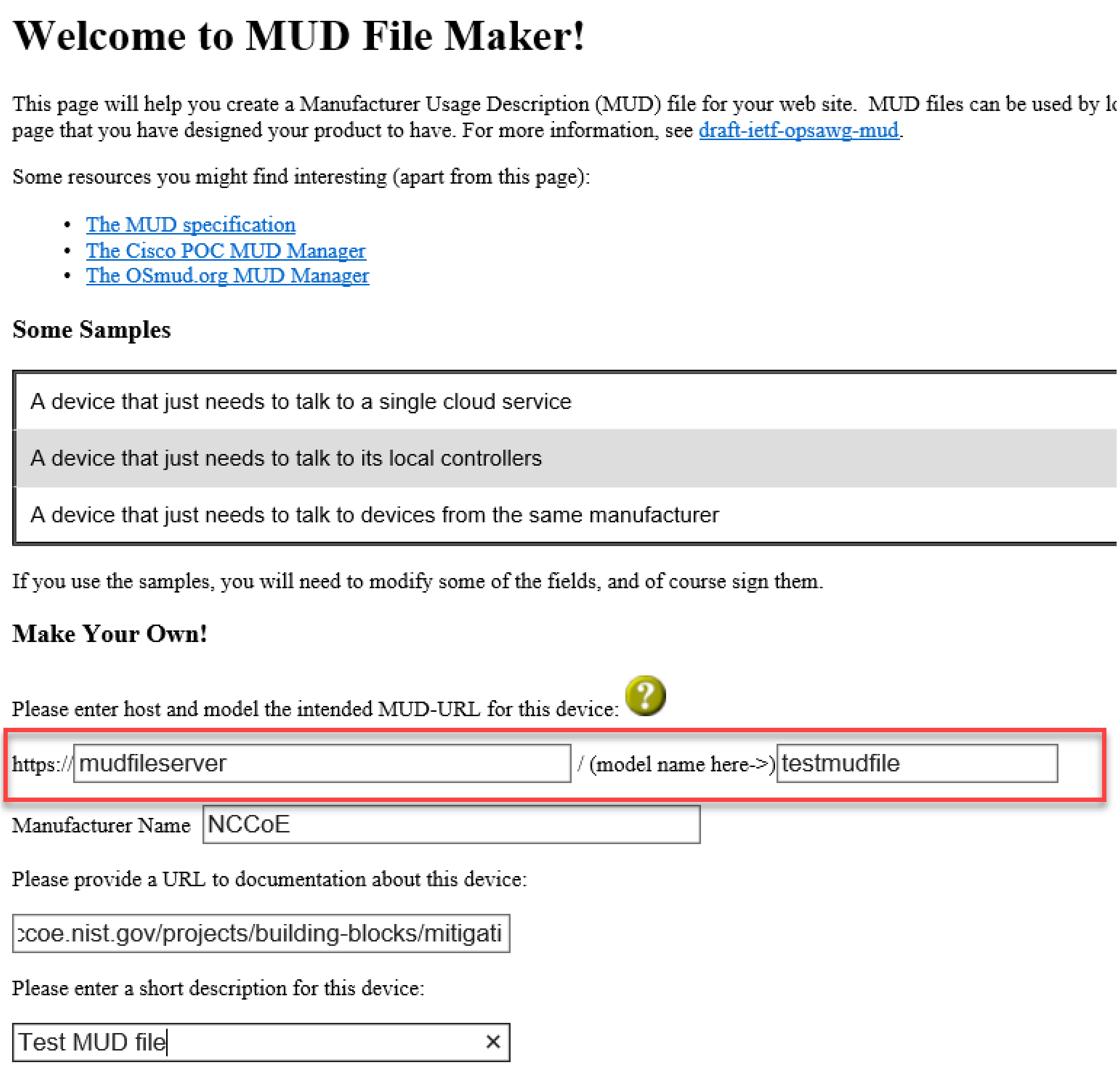
This section details creating and signing a MUD file on the MUD file server. The MUD specification does not mandate that this signing process be performed on the MUD file server itself.

2.2.3.2.1 MUD File Creation

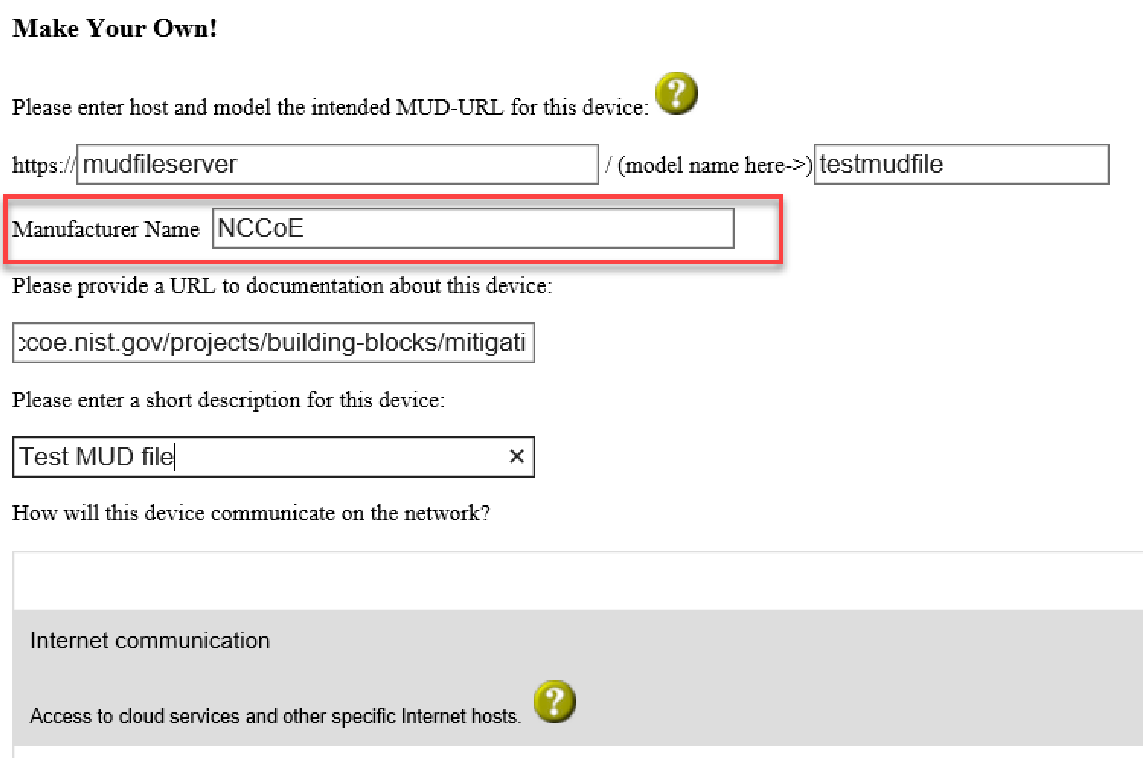
An online tool called MUD Maker was used to build MUD files. Once the permitted communications have been defined for the IoT device, proceed to www.mudmaker.org to leverage the online tool. There is also a list of sample MUD files on the site, which can be used as a reference. Upon navigating to www.mudmaker.org, 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):

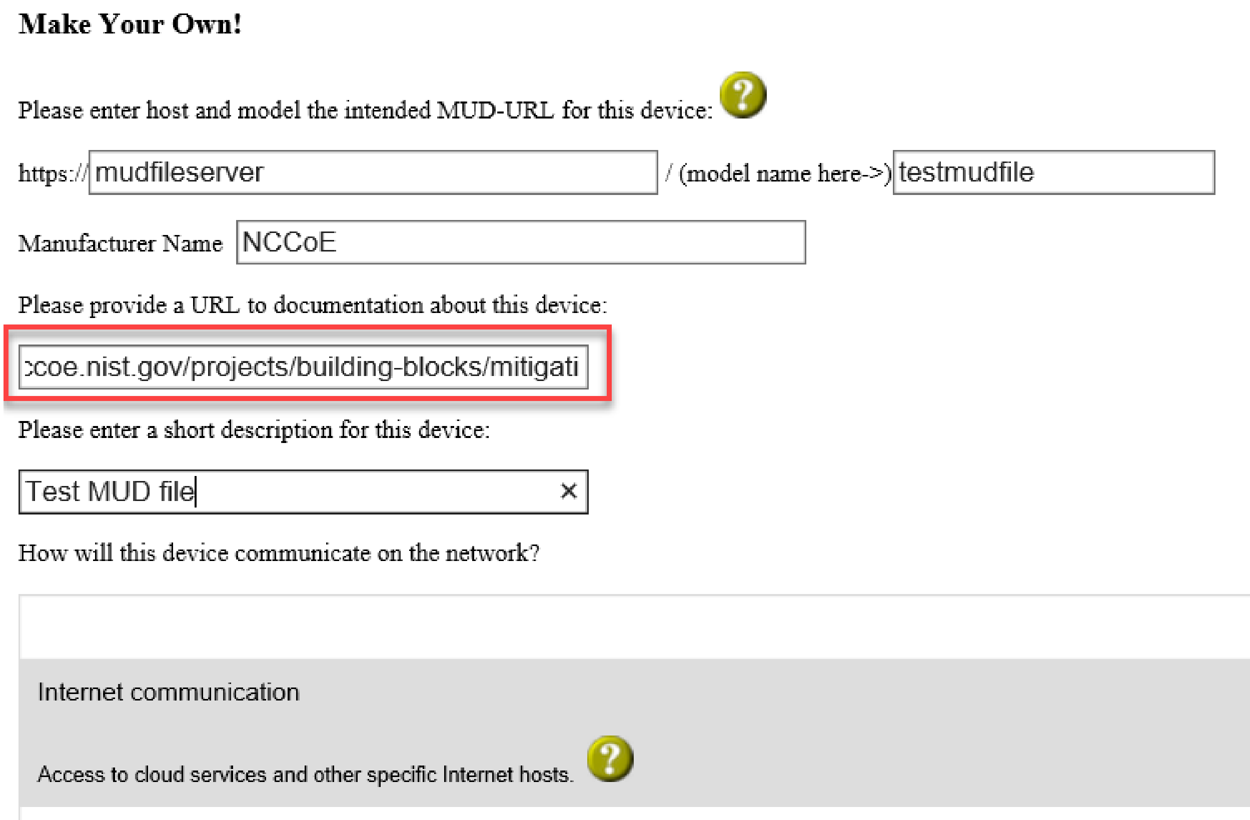
   Sample input: mudfileserver, testmudfile



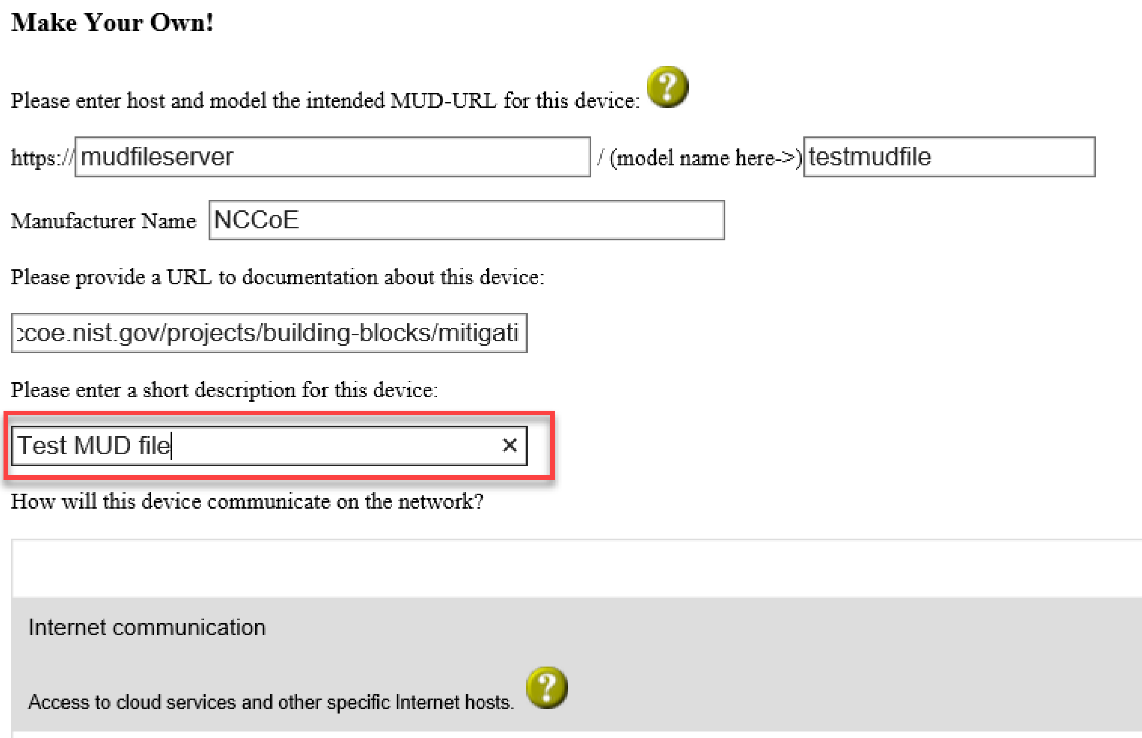
2. Specify the Manufacturer Name of the device in the appropriate input field, which is outlined in red in the screenshot below:



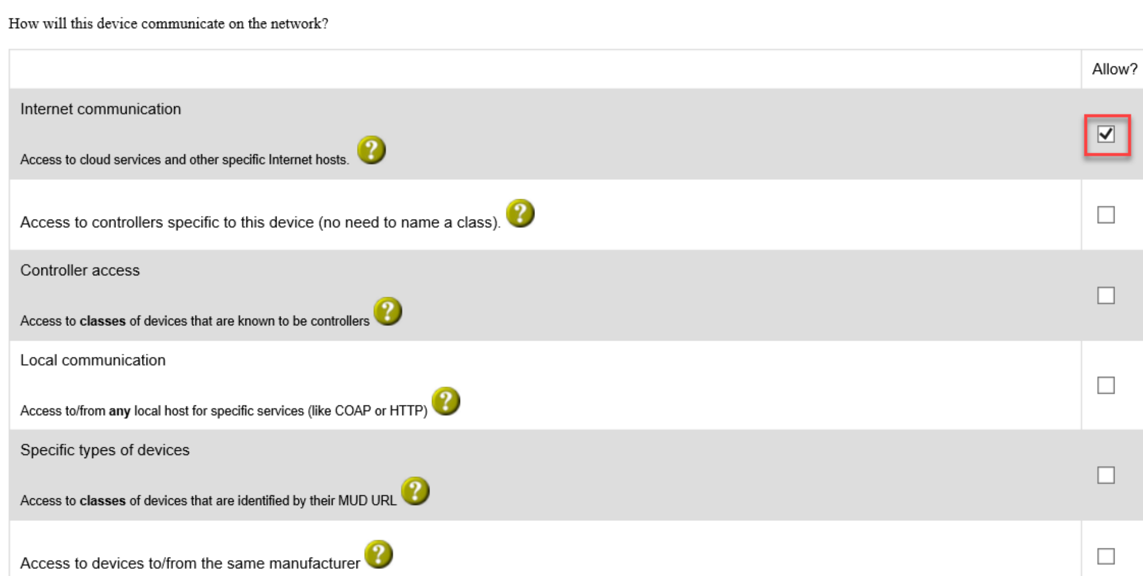
3. Include a URL to provide documentation about this device in the appropriate input field, which is outlined in red in the screenshot below:



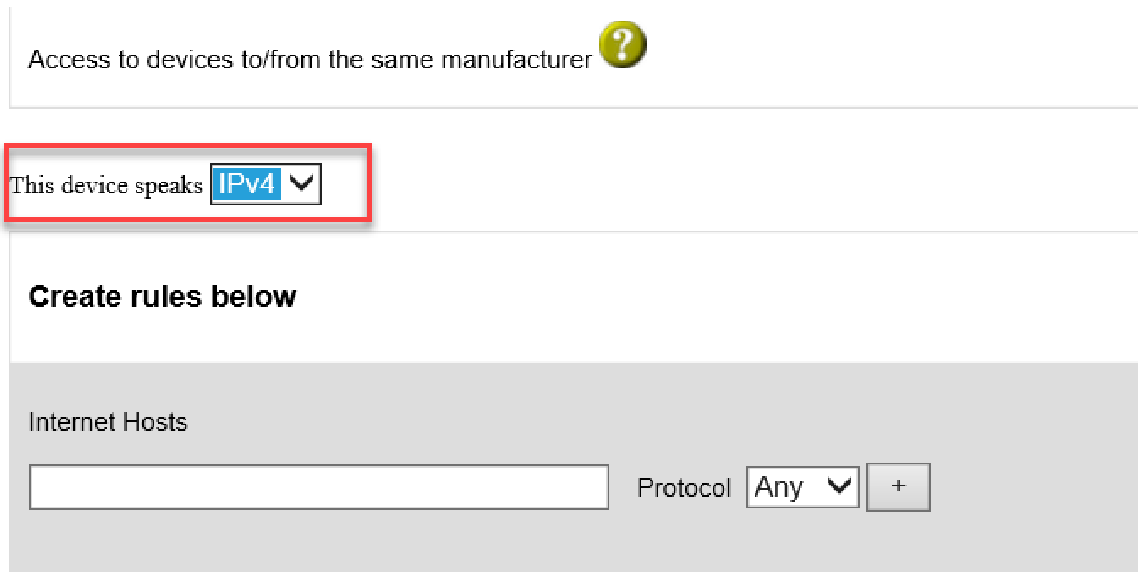
4. Include a short description of the device in the appropriate input field, which is outlined in red in the screenshot below:



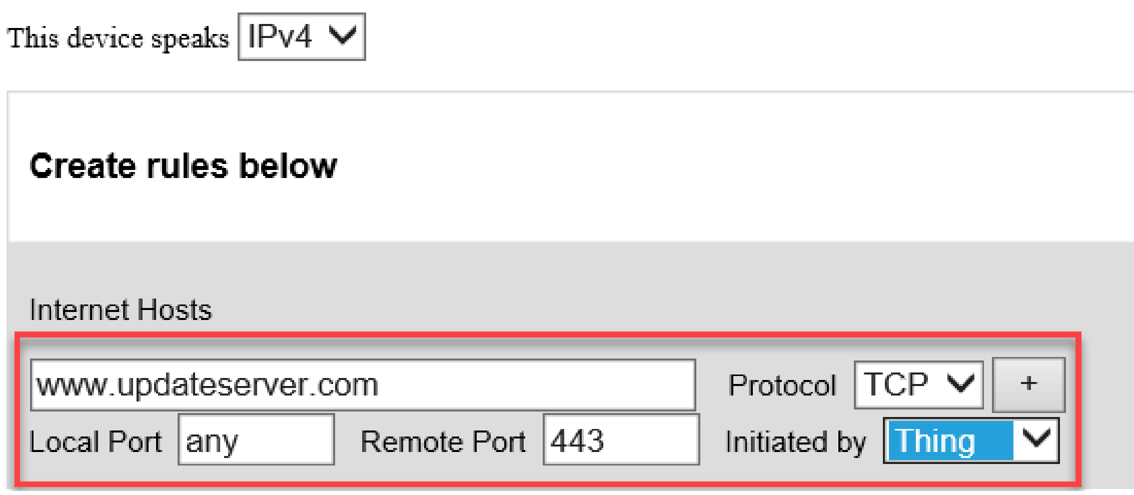
5. Check the boxes for the types of network communication that are allowed for the device:



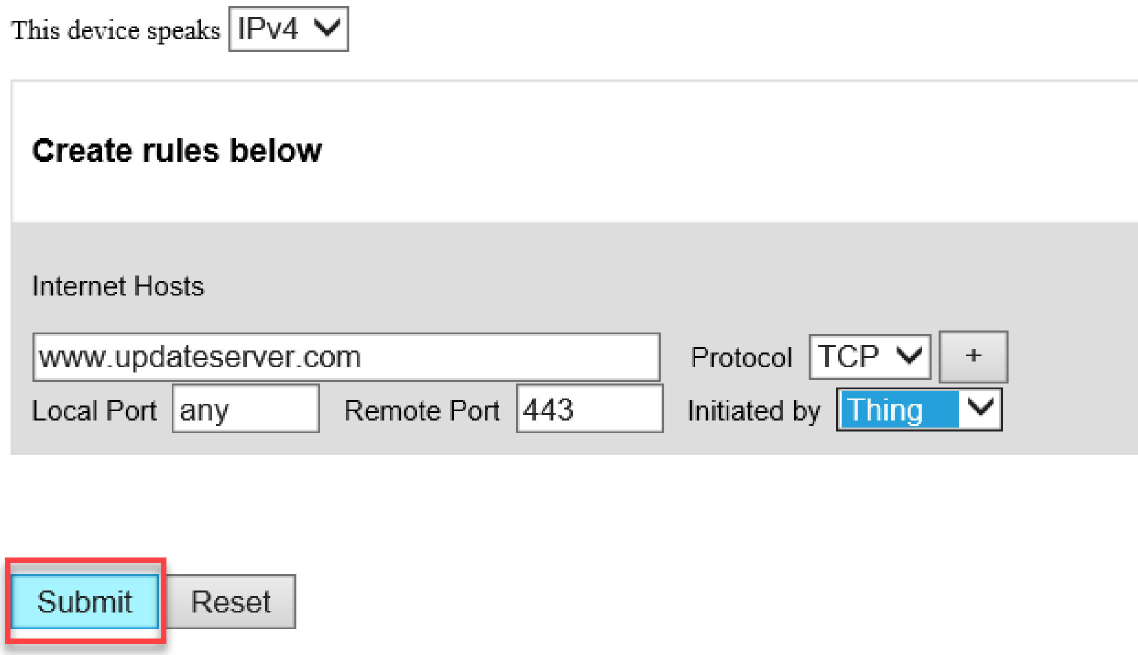
6. Specify the internet protocol version that the device leverages:



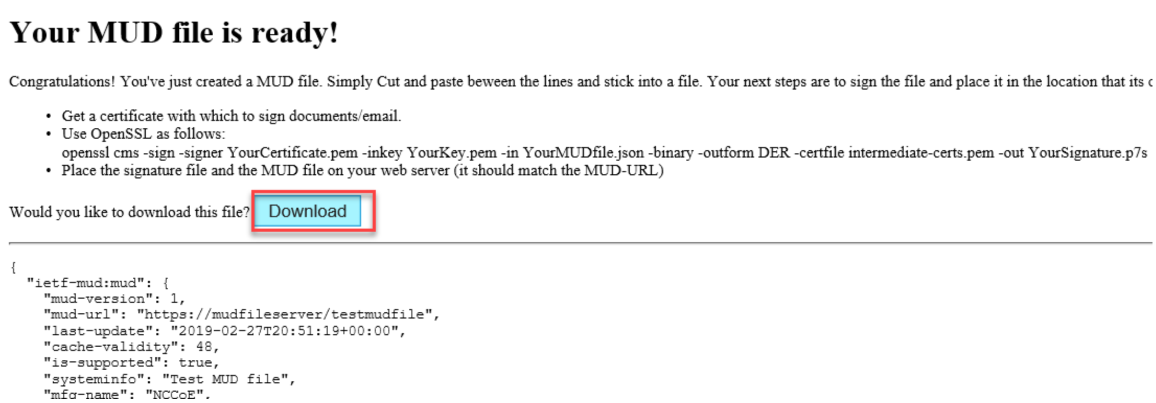
7. Specify values for the fields (Internet Hosts, Protocol, Local Port, Remote Port, and Initiated by) that describe the communications that will be permitted for the device:



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:



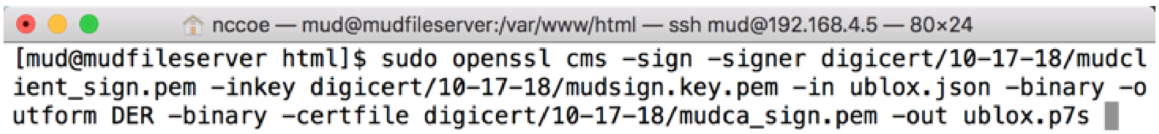
10. Click **Save** to store a copy of the MUD file:

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 for the Signing Certificate.

1. Sign the MUD file by using the following command:

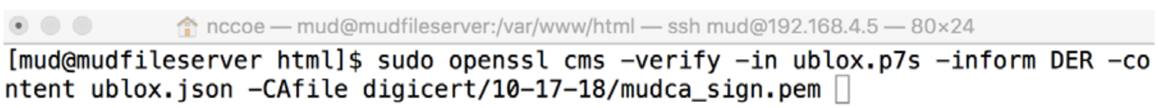
```
sudo openssl cms -sign -signer <Signing Certificate> -inkey <Private Key for
Signing Certificate> -in <Name of MUD File> -binary -outform DER -binary -
certfile <Intermediate Certificate for Signing Certificate> -out <Name of MUD
File without the .json file extension>.p7s
```



   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 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.

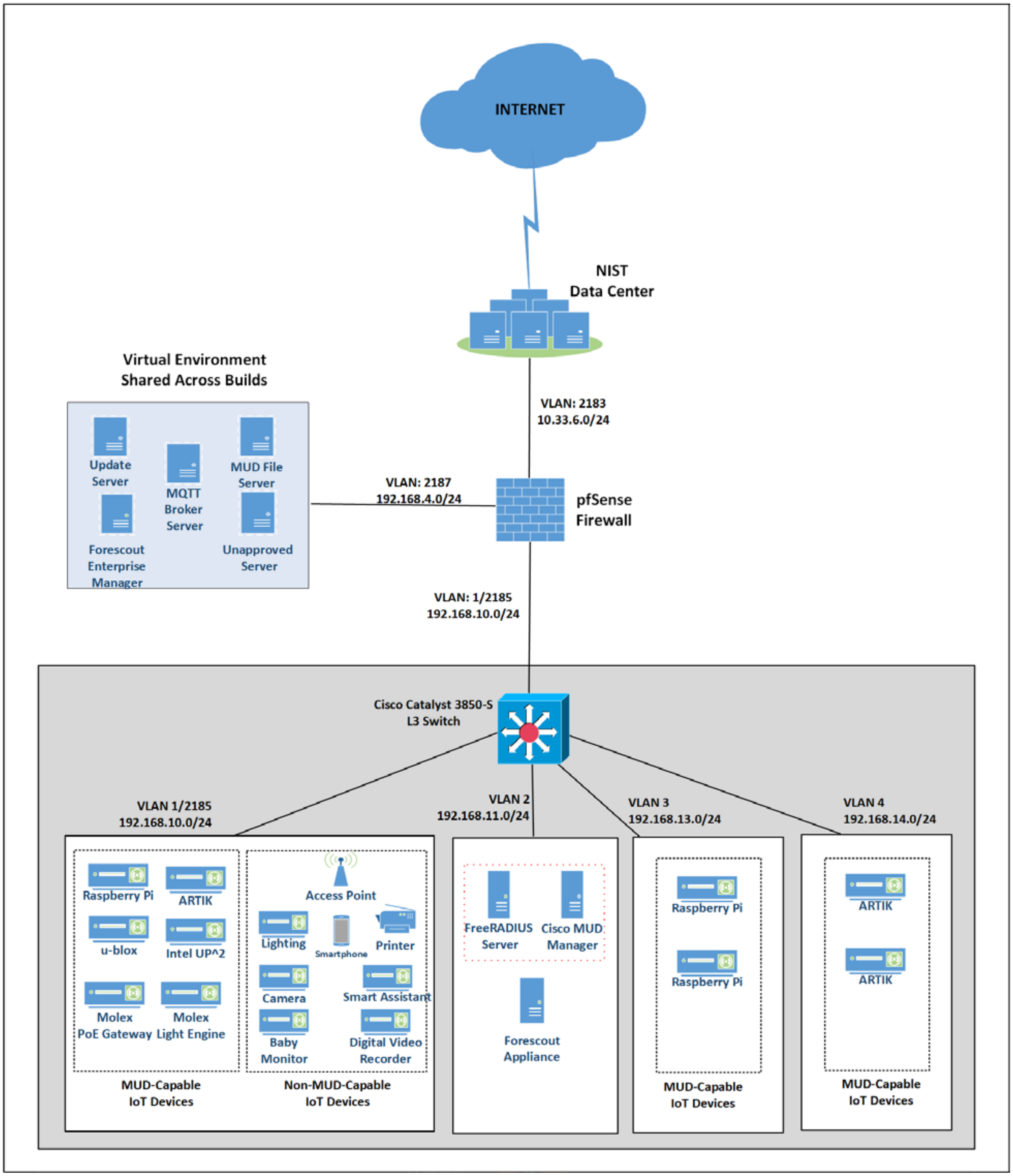
2.3.2 Configuration Overview

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.

784 **Figure 2-1 Physical Architecture–Build 1**



785

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 operating system 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://www.digicert.com/certcentral-support/digicert-getting-started-guide.pdf

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://www.digicert.com/certcentral-support/client-certificate-guide.pdf.

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 smart device by leveraging an MQTT broker server (we describe setting up the MQTT broker server in Section 2.8).

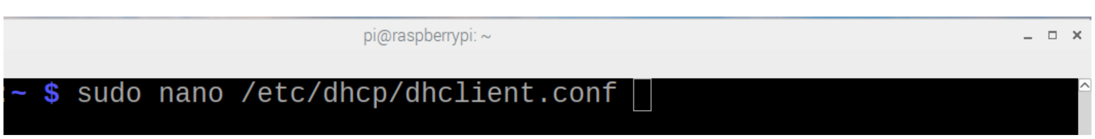
2.5.2.2.1 DHCP Client Configuration

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:

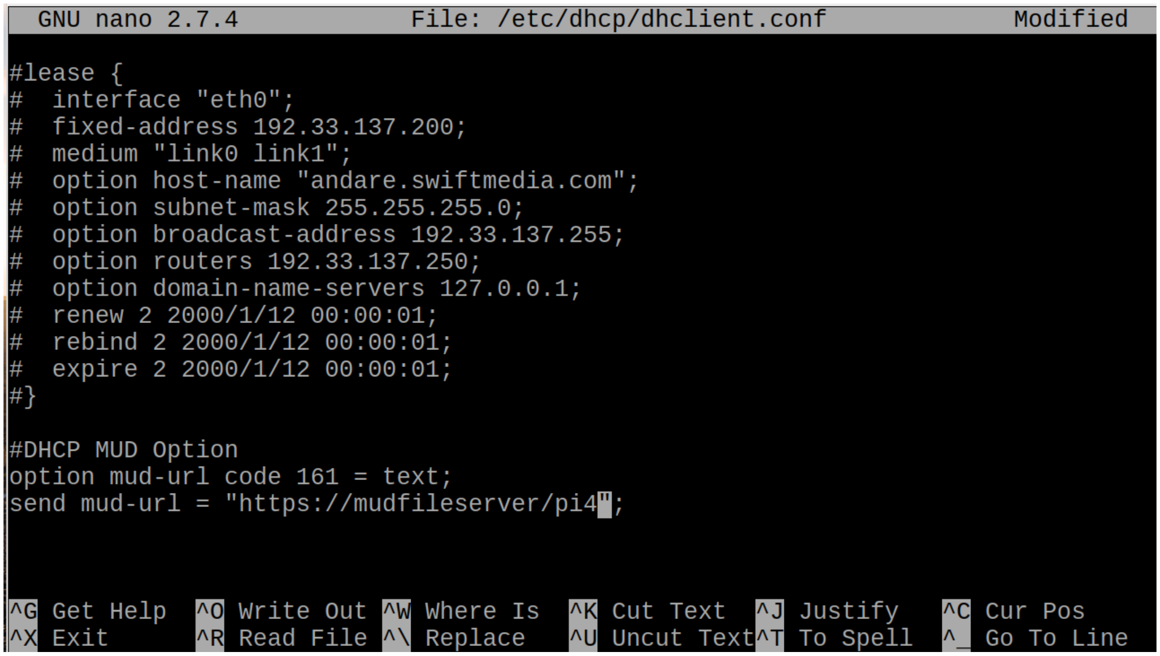
1. Open a terminal on the device.
2. Ensure that any other conflicting DHCP clients are disabled or removed.
3. Install the dhclient package (if needed).
4. Edit the *dhclient.conf* file by entering the following command:

```
sudo nano /etc/dhcp/dhclient.conf
```



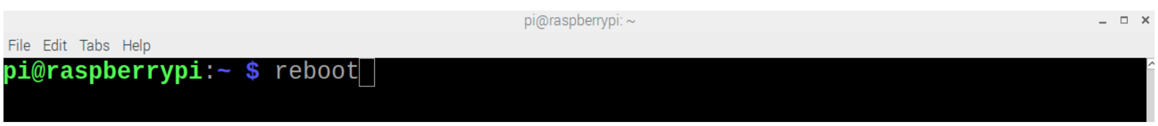
5. Add the following lines:

```
option mud-url code 161 = text;
send mud-url = "<insert URL for MUD File here>";
```



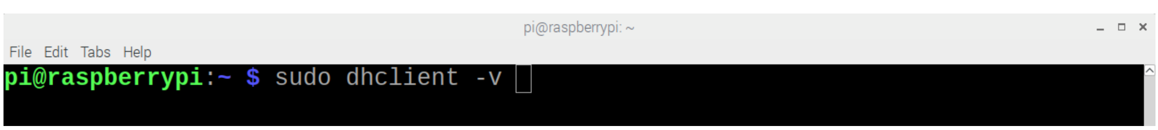
6. Save and close the file.
7. Reboot the device:

```
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 operating system 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 smart 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, smartphones, lighting, a smart 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 this, 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 per 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 operating system 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.

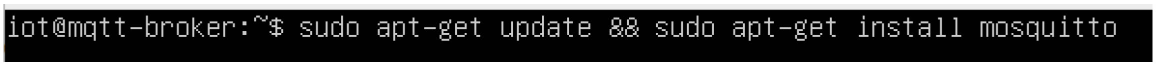
2.8.3 Setup

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 operating system 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 of the software necessary to enable a MUD infrastructure on premises. 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, the 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.

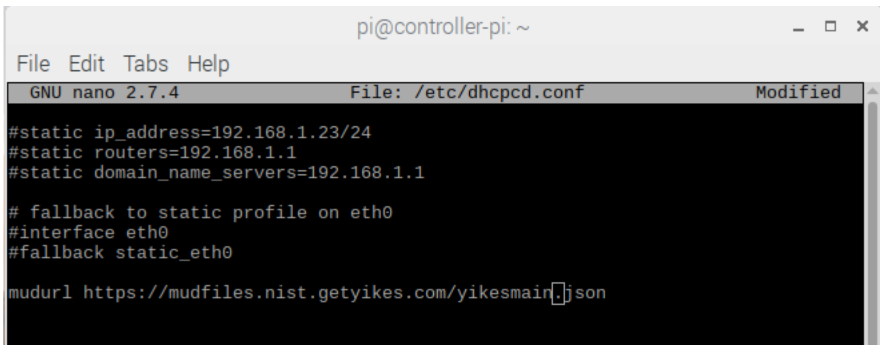
3.6.1.2 Setup

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."

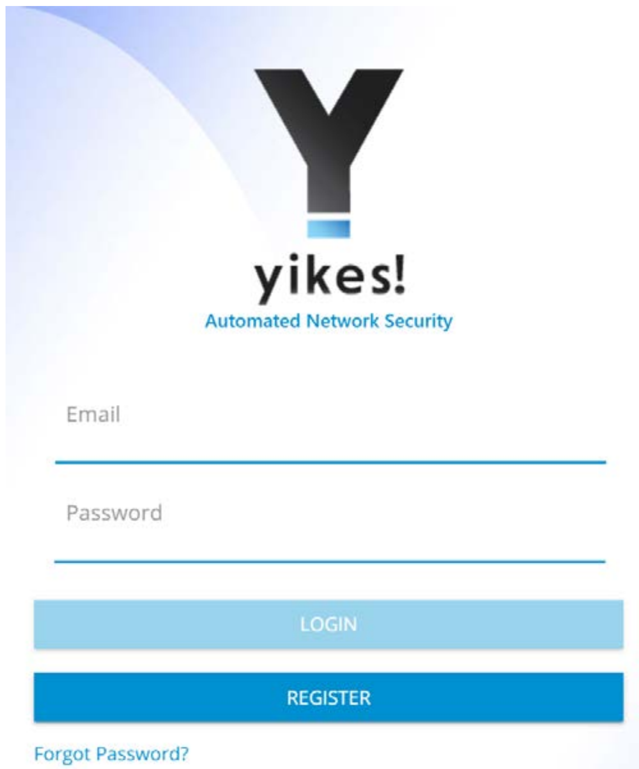
3.9.3 Setup

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.

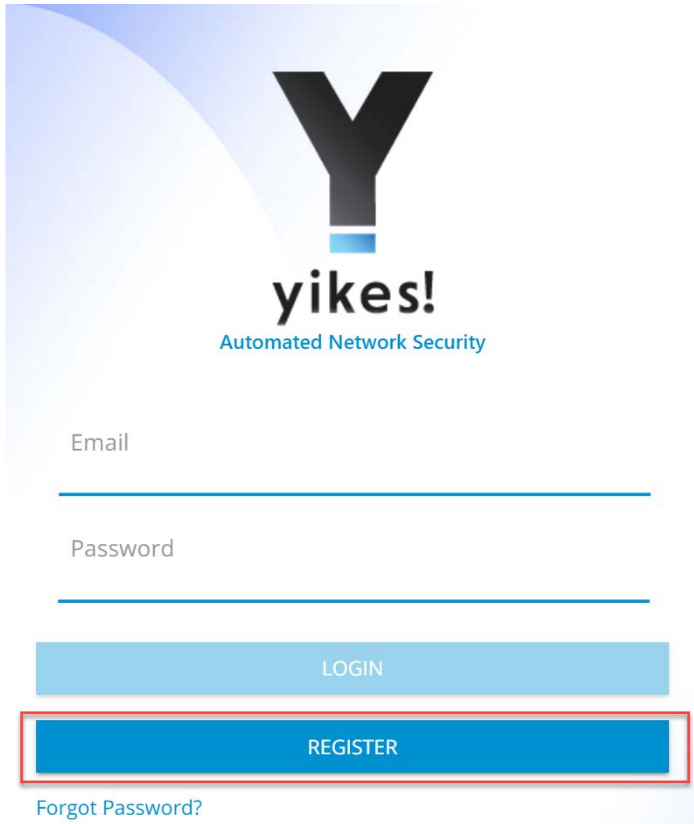
3.9.3.1 Yikes! Router and Account Cloud Registration

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.

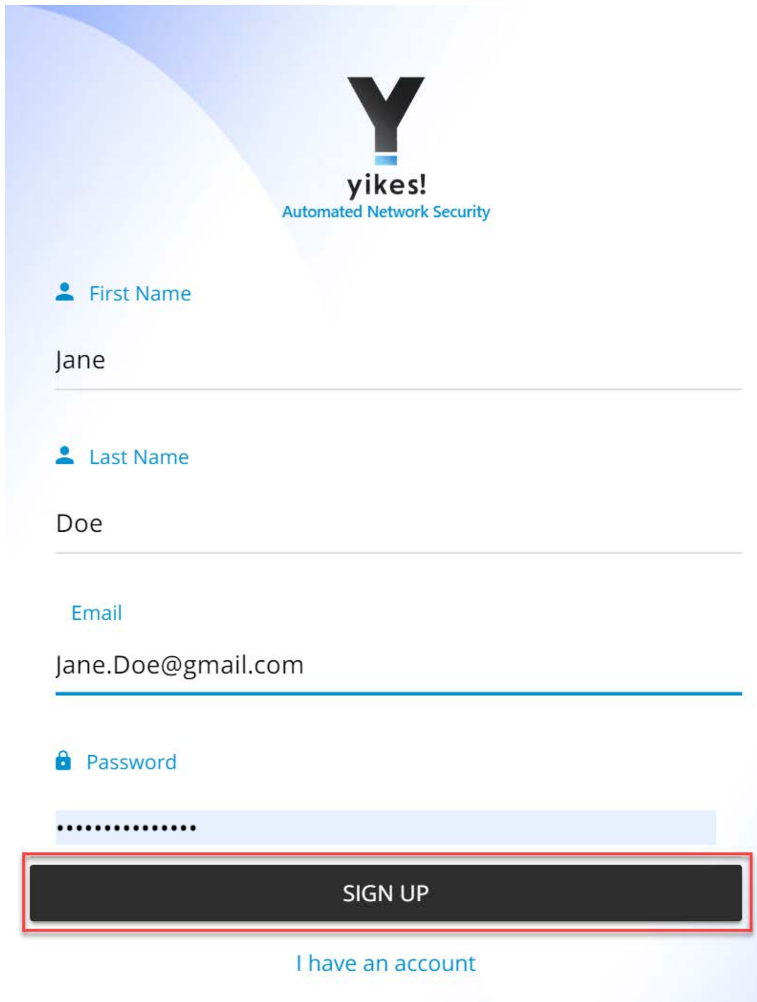
1. Open a browser and access the Yikes! UI. (In the preproduction version of the router, accessing the UI required inputting a URL provided by MasterPeace.):



2. Click on the **Register** button to sign up for an account:

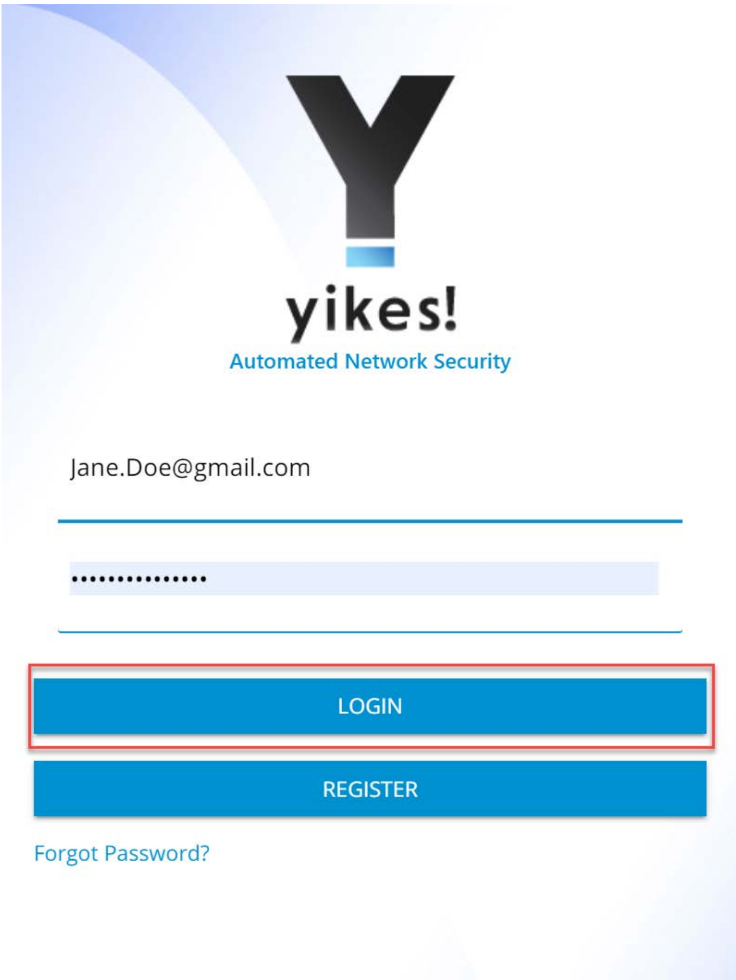


3. Populate the requested information for the account: First Name, Last Name, Email, and Password. Click **Sign Up:**



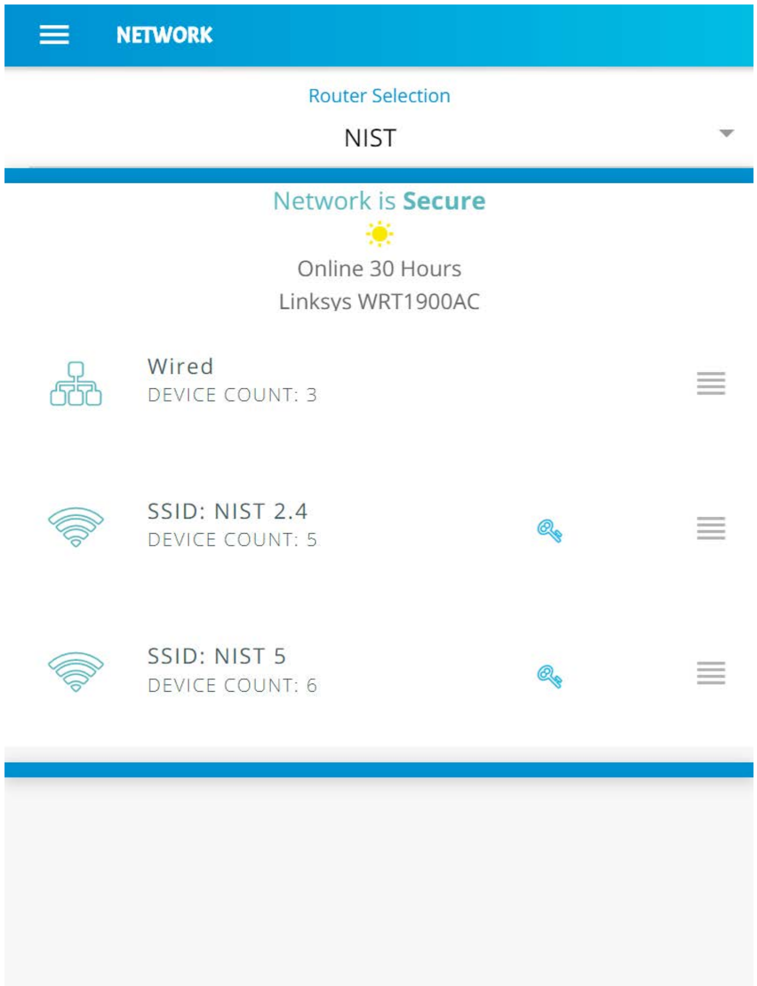
Note: There will be additional steps related to associating the Yikes! router with the Yikes! account being created. However, at this implementation, this process was still under development.

1474 4. Once the account is approved and linked to the Yikes! router, **Log in** with credentials created in
1475 step 3:



1476

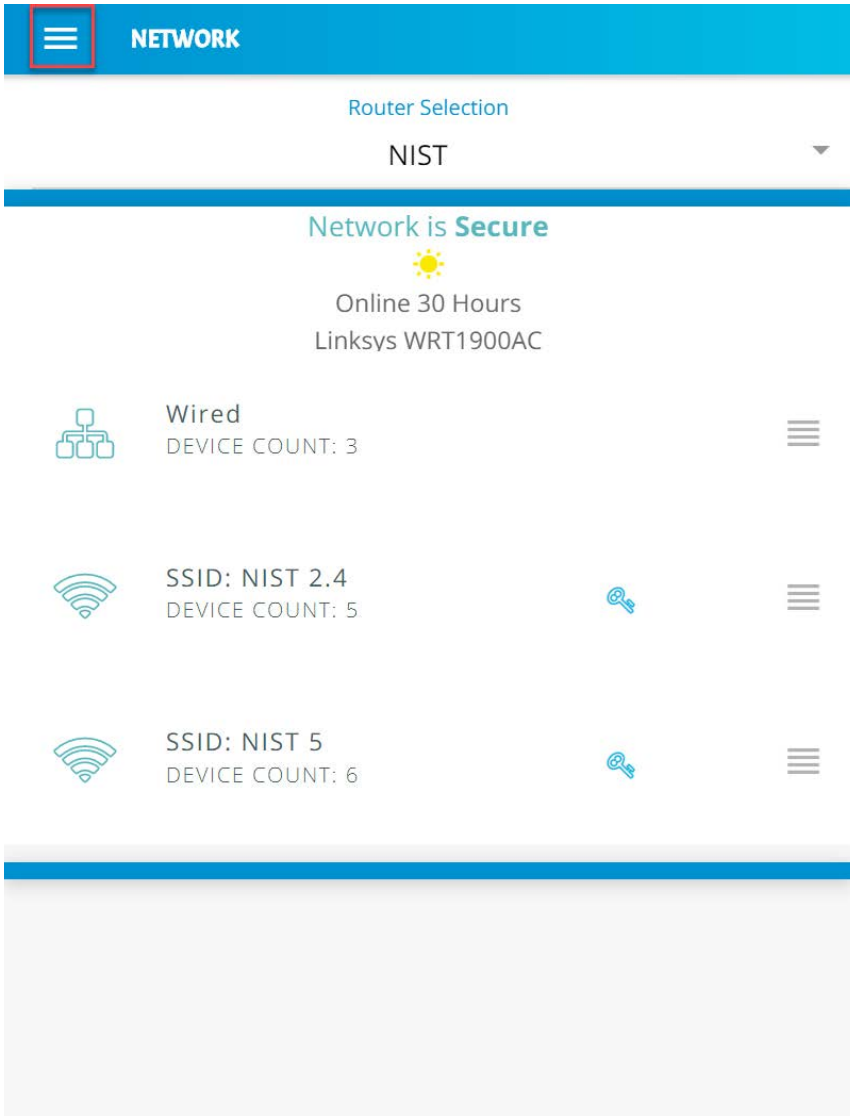
5. The home screen will show the network overview:



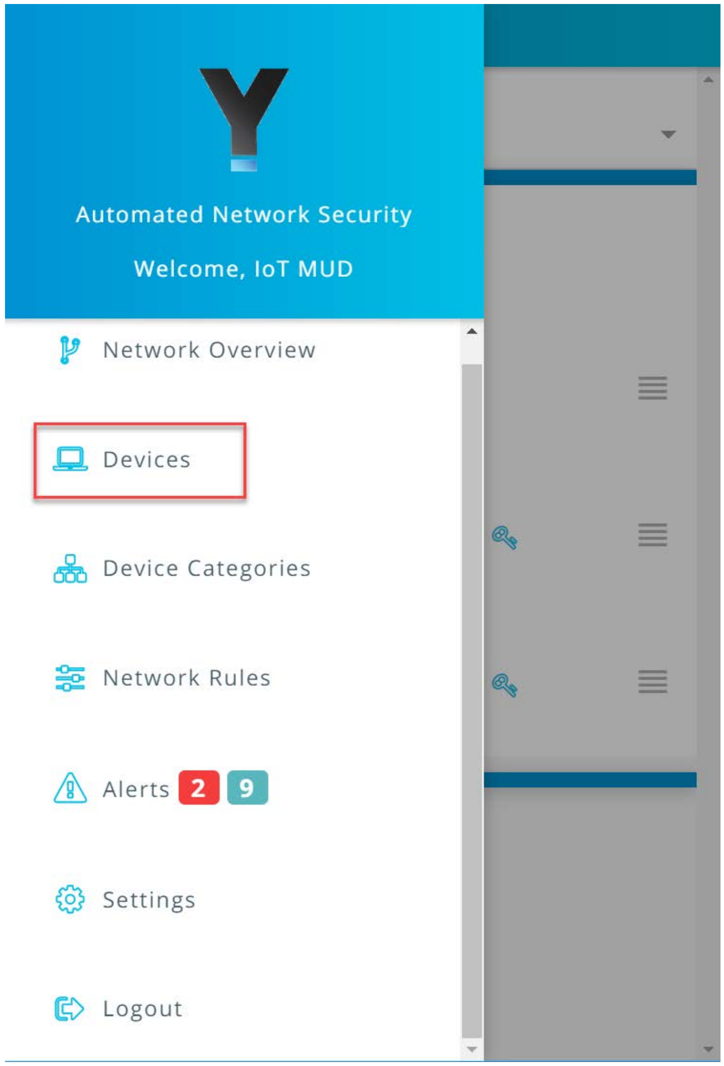
3.9.3.2 Yikes! MUD-Capable IoT Device Discovery

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.

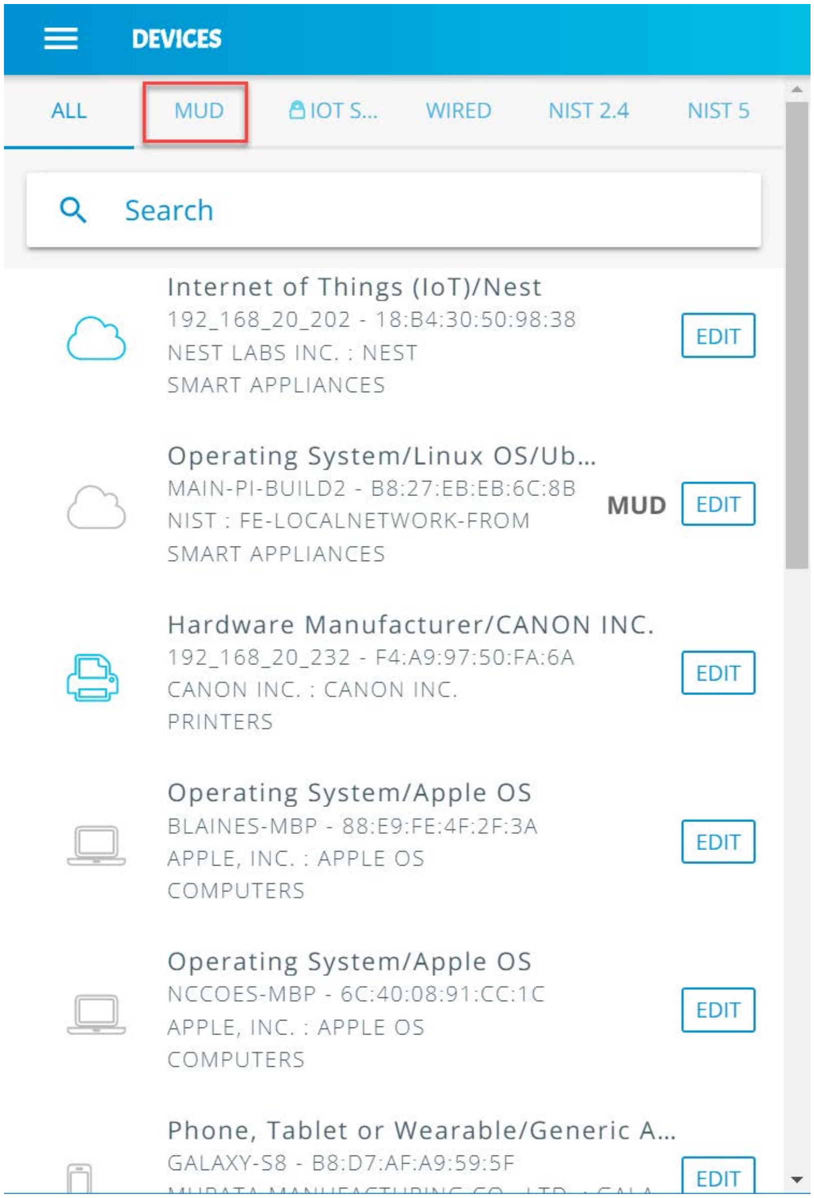
1483 1. Open the menu pane in the UI:

1484 

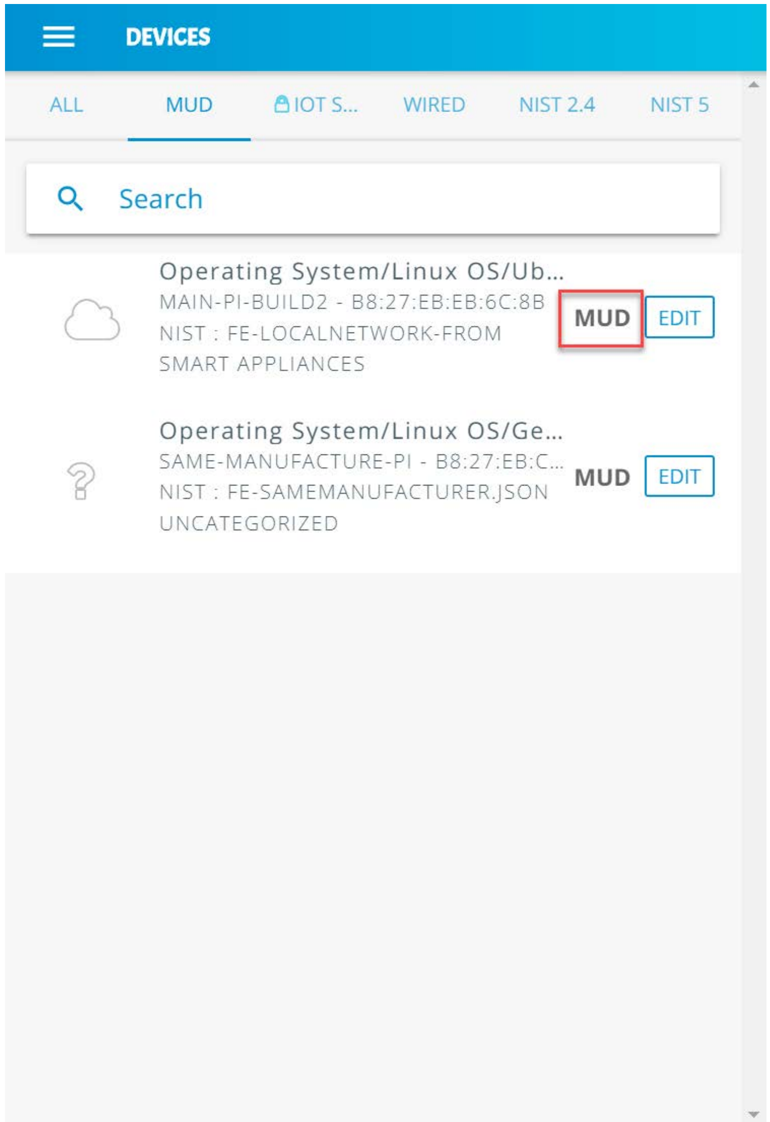
2. Click the **Devices** button to open the devices menu:



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:

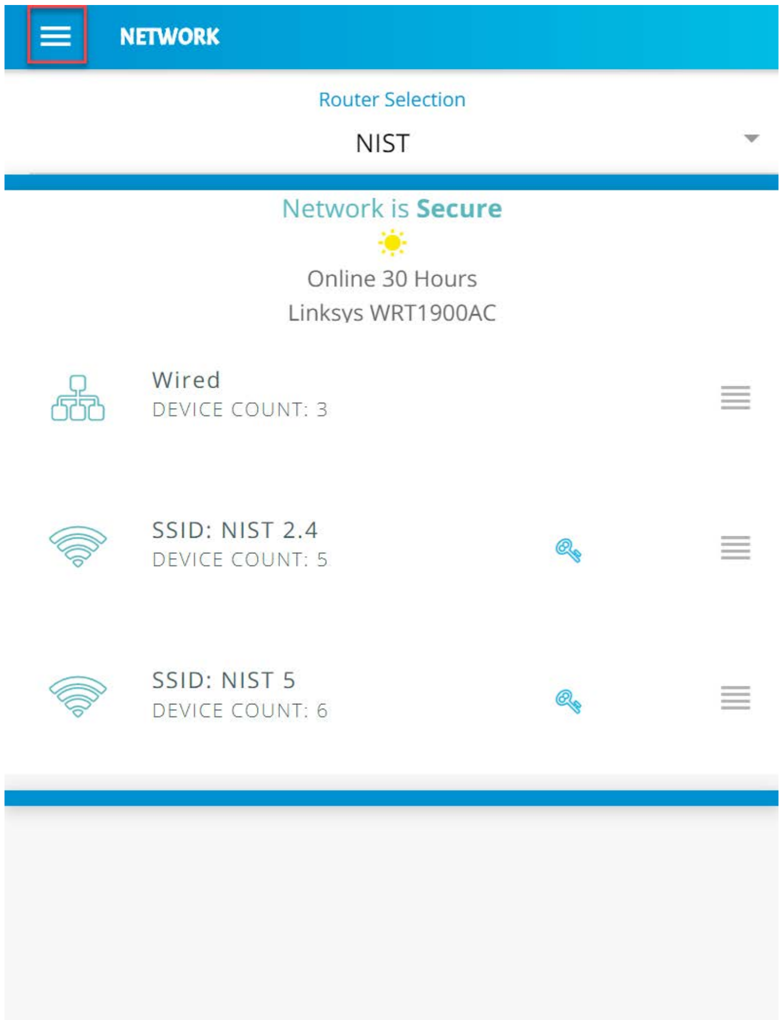


3.9.3.3 Yikes! Alerts

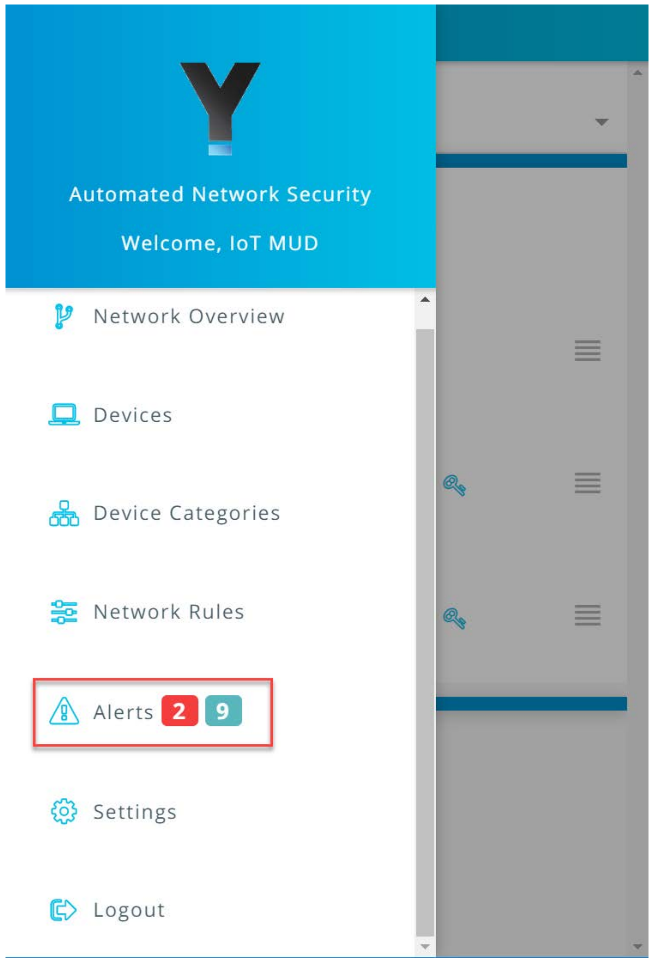
This section details the Yikes! alerting capability. This feature is accessible through the Yikes! mobile application and notifies users when new devices have been connected to the network. Additionally, this feature alerts the user when new devices are not recognized as known devices and are placed 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:

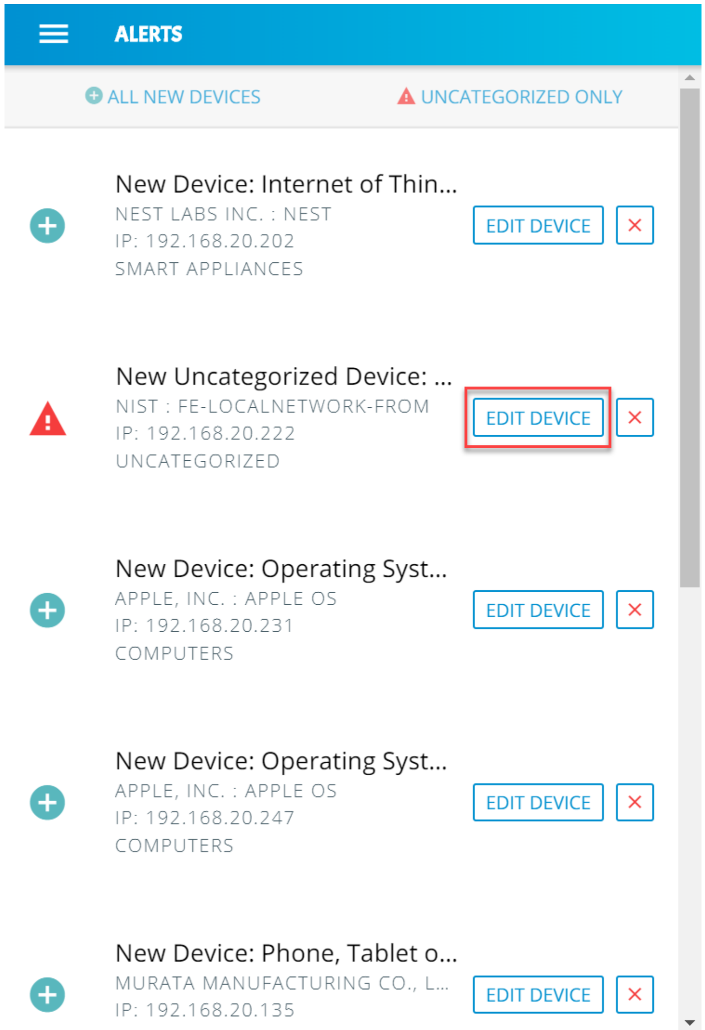


1502 2. Click the **Alerts** to open the Alerts menu:

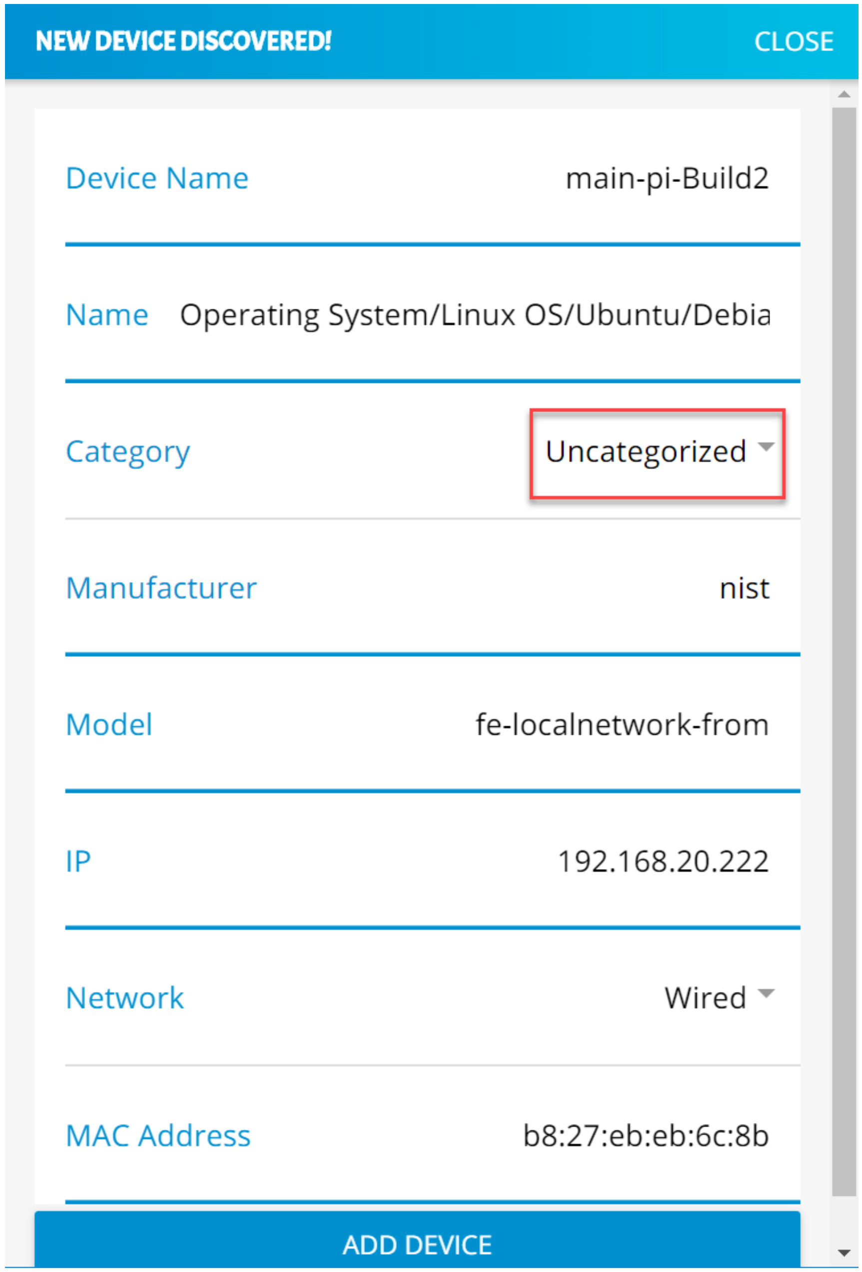


1503

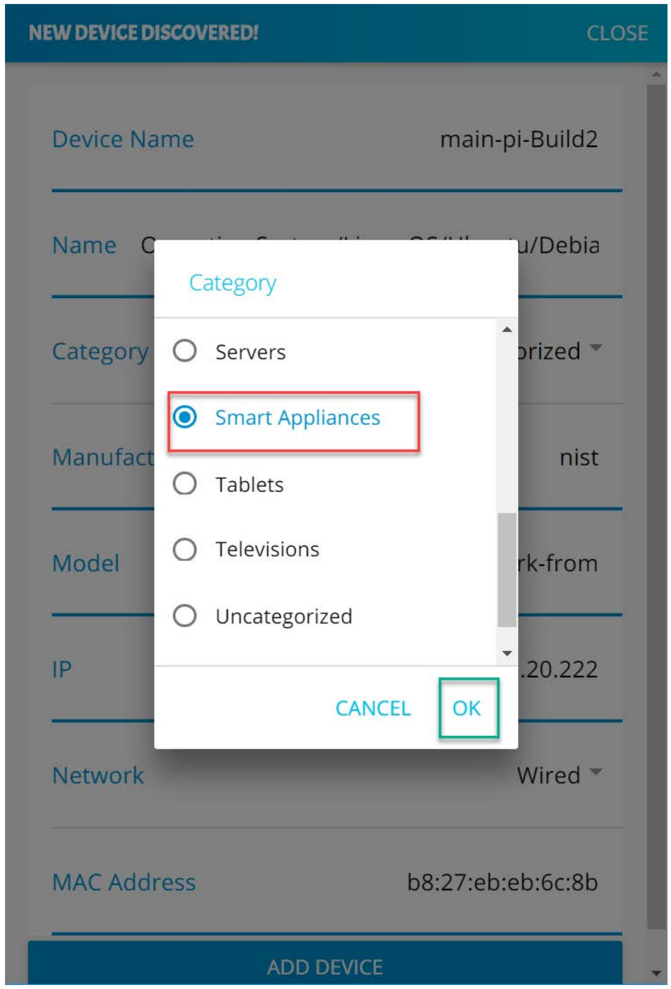
1504 3. Select a device to edit the device information and category by clicking **Edit Device:**

1505 

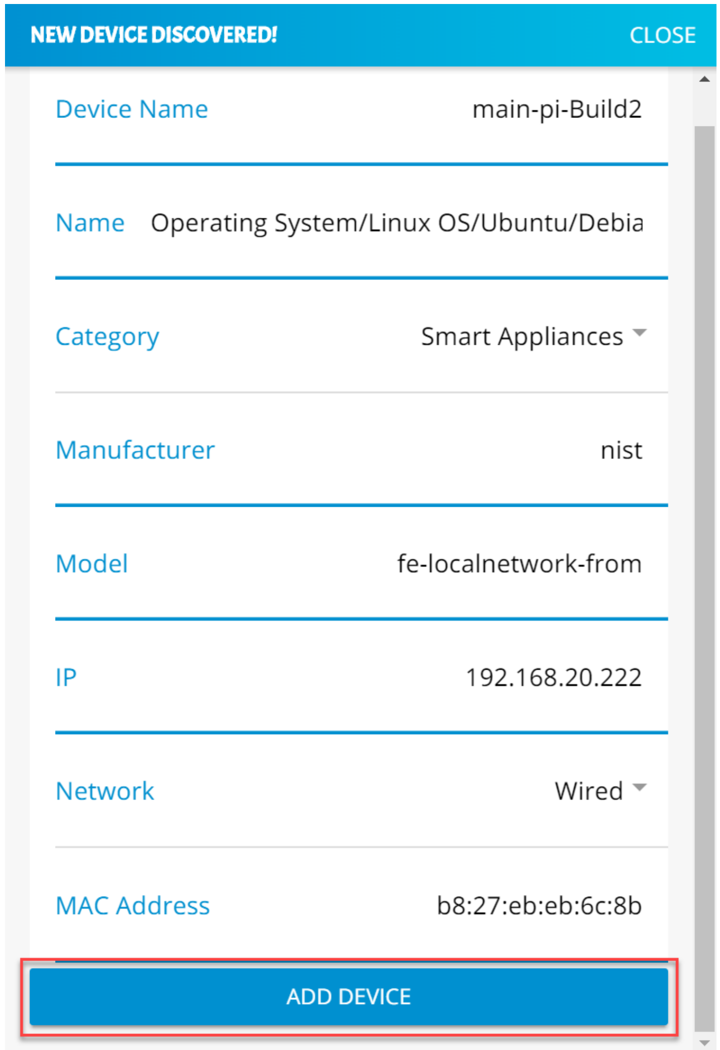
1506 4. Modify the **Category** of the device by clicking the device's current category:

1507 

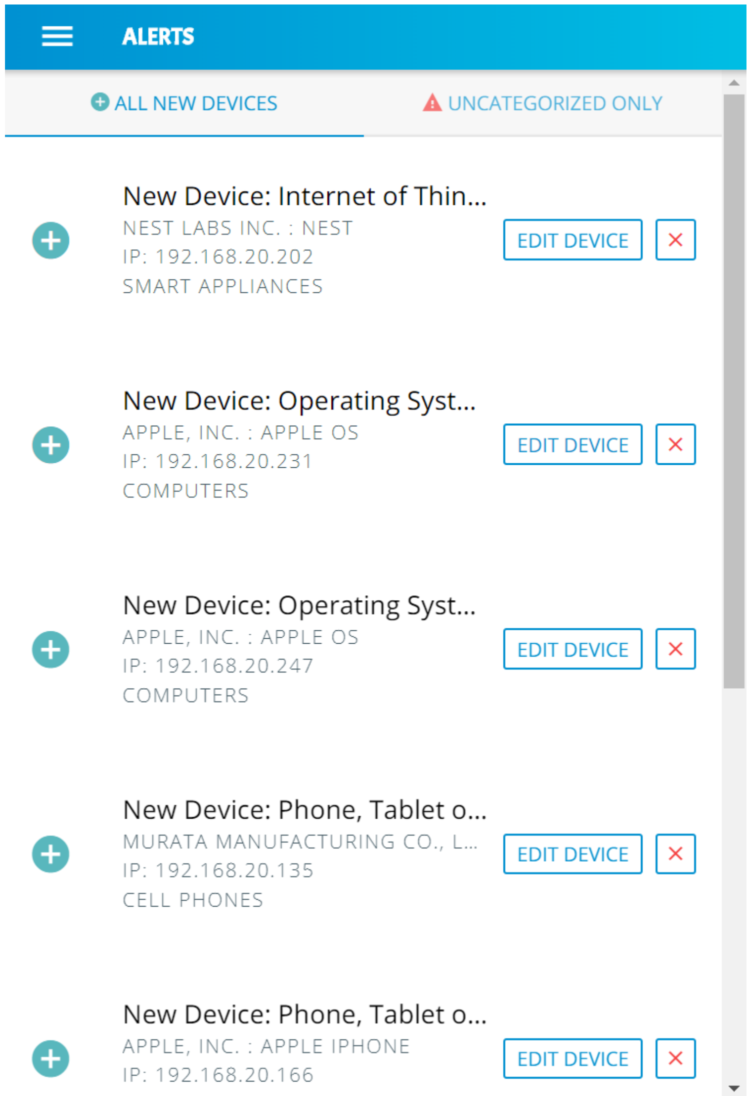
1508 5. Select the desired category, in this case **Smart Appliances,** and click **OK:**

1509 

6. The device **Category** will update to reflect the new selection. Click **Add Device** to complete the process:



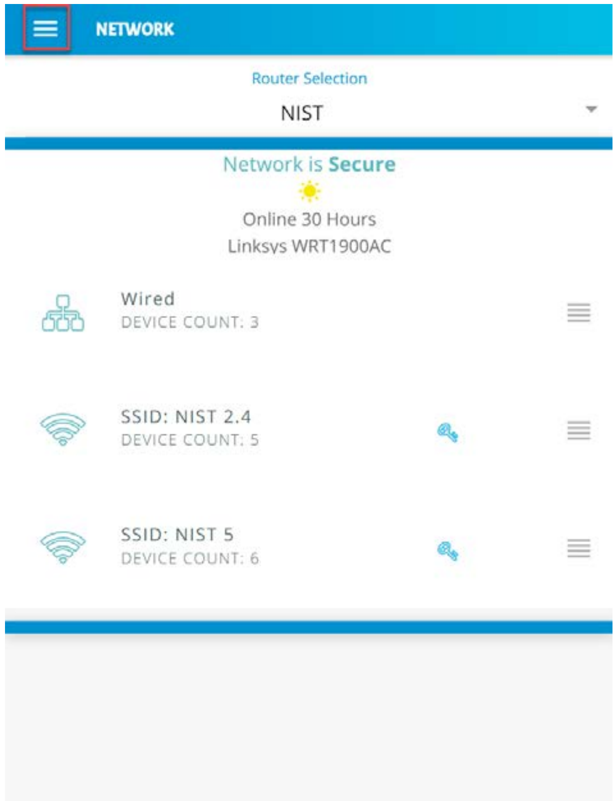
7. The alerts menu will update and no longer include the device that was just modified and added:



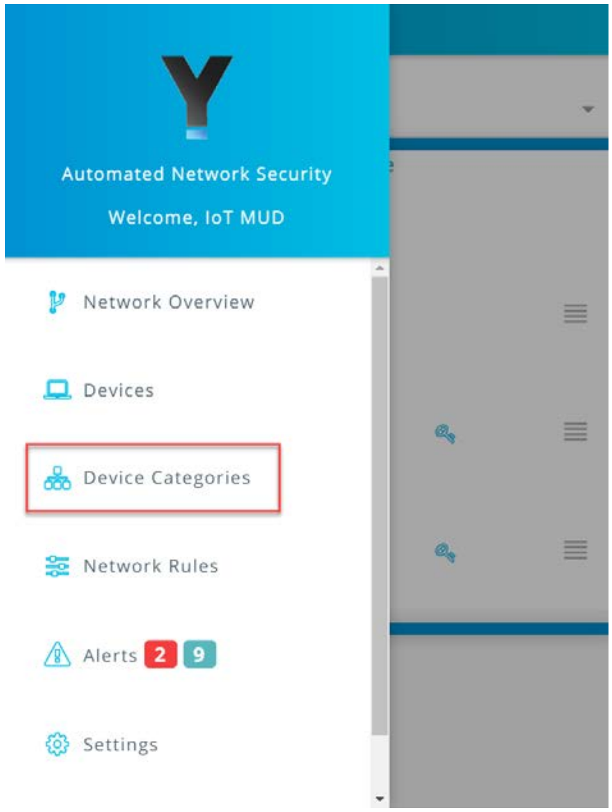
3.9.3.4 Yikes! Device Categories and Setting Rules

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.

1. Click the menu bar to open the menu pane:



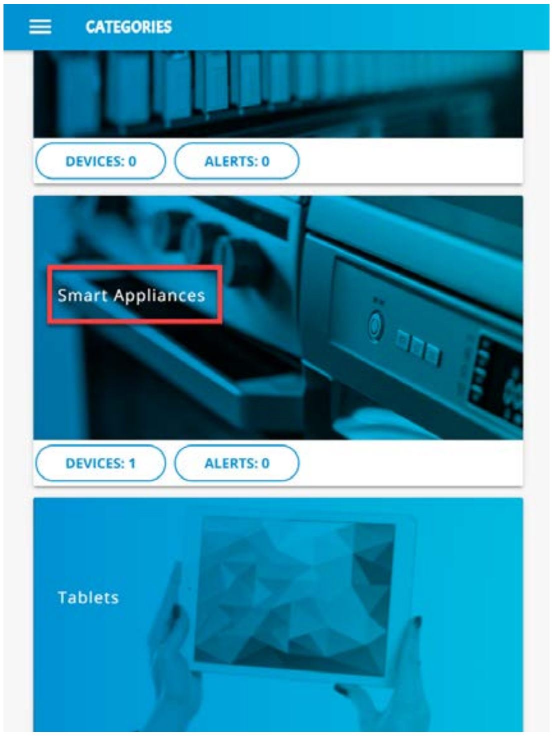
1521 2. Click the **Device Categories** option to view all device categories:



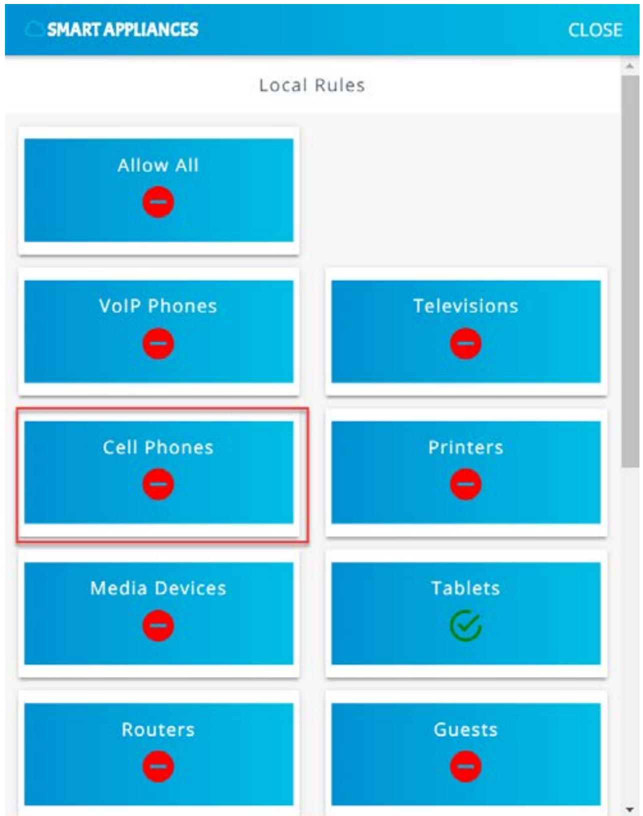
1522

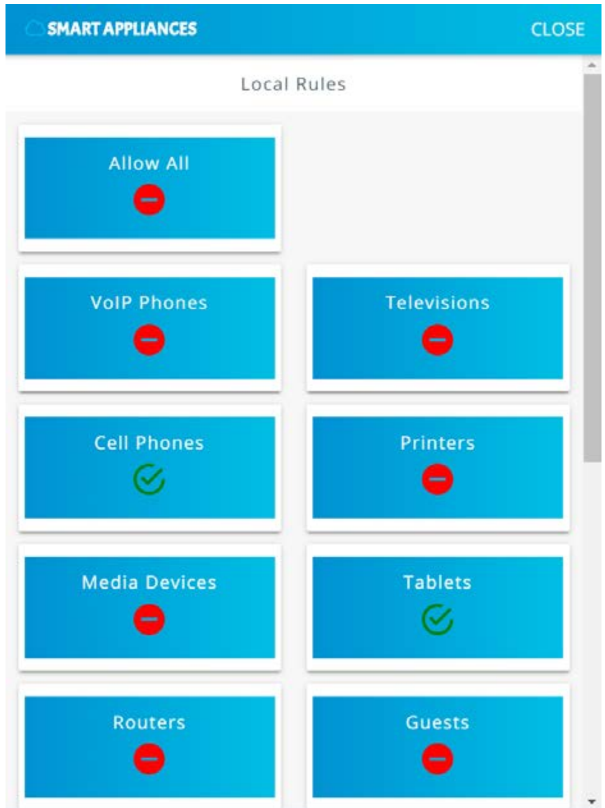
1523 3. Select the category of device to view and configure rules:

1524

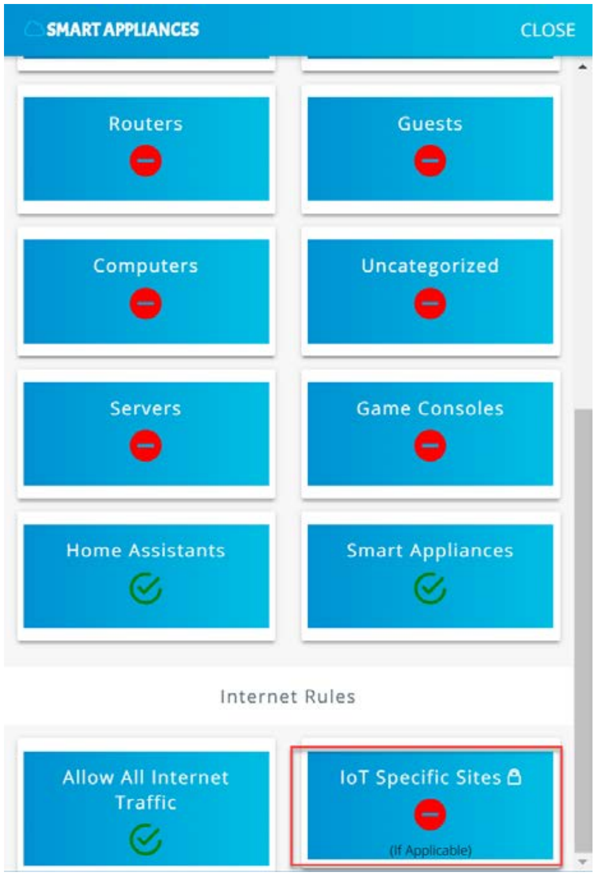


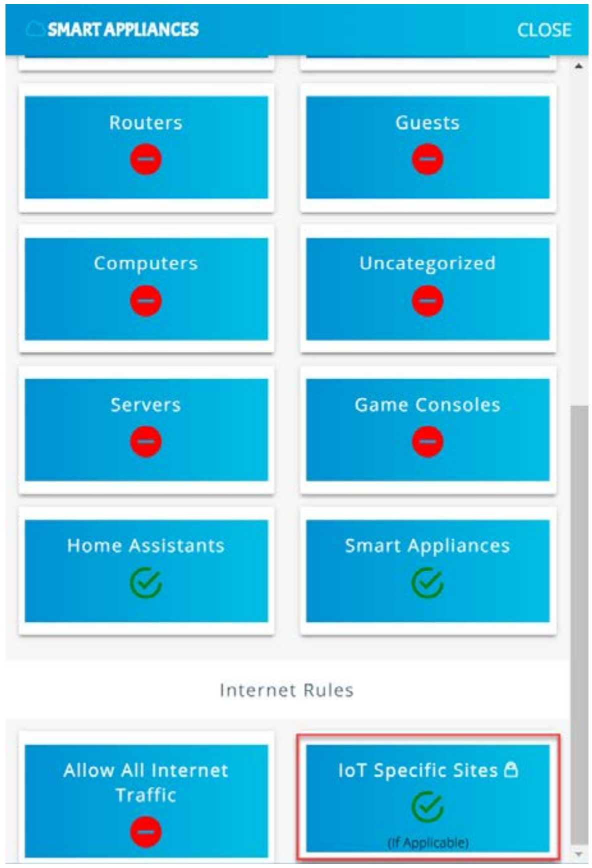
1525 4. Modify local rules by clicking on the category of devices with which the selected category is
1526 permitted to communicate:

1527 

1528 
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:**

1531 
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)

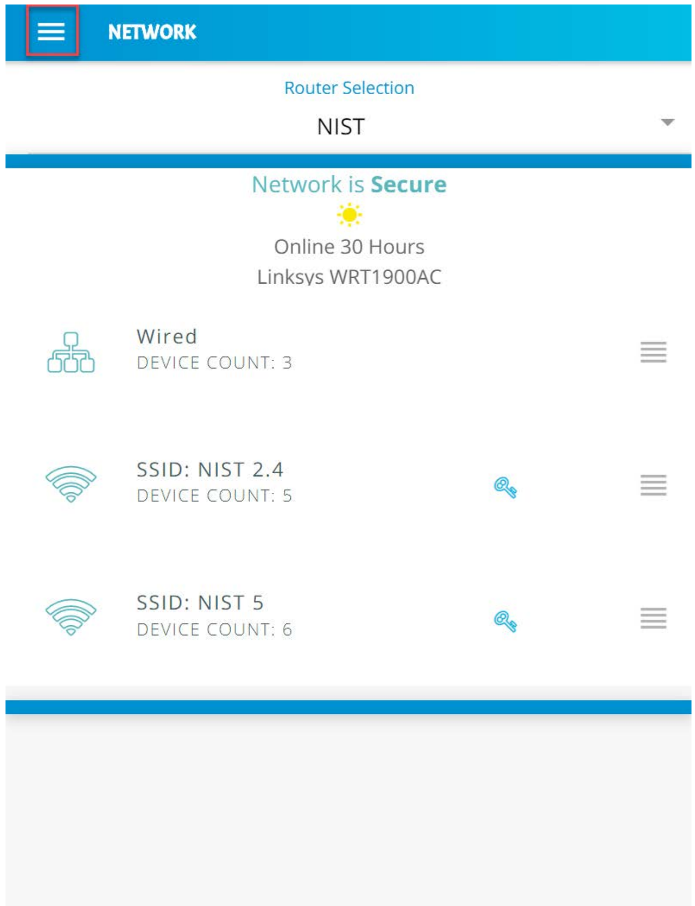


Smart appliances should now be permitted to communicate locally to Smart Appliances, Home Assistants, Tablets, Cell Phones, and, externally, to IoT Specific Sites.

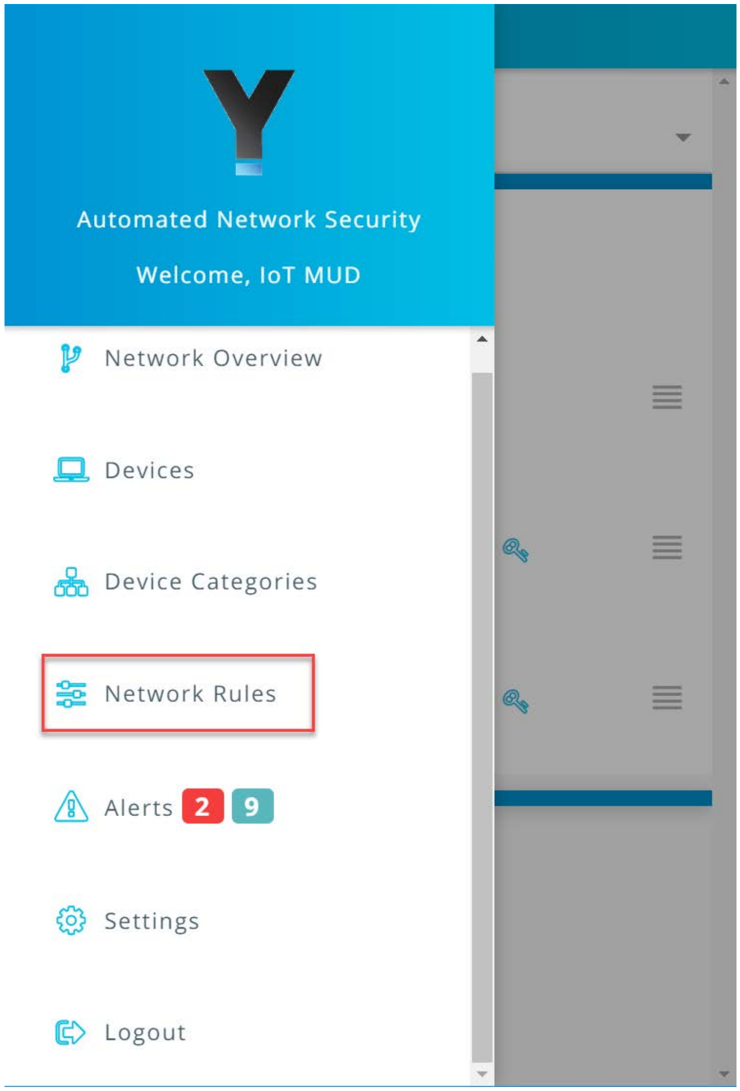
3.9.3.5 Yikes! Network Rules

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:

1541

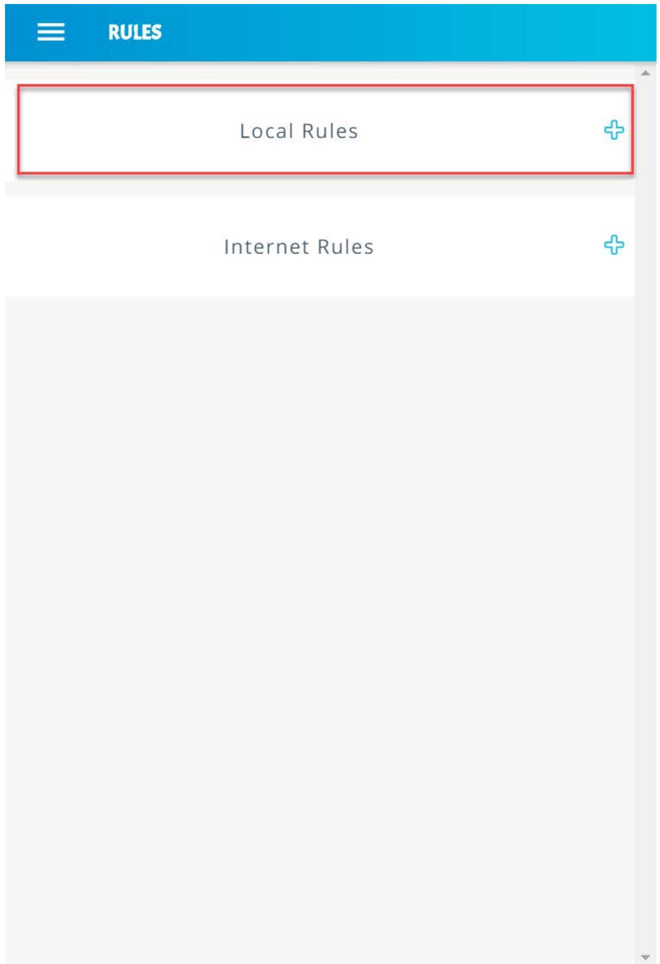

NETWORK
Router Selection
NIST
Network is Secure
Online 30 Hours
Linksys WRT1900AC
Wired
DEVICE COUNT: 3
SSID: NIST 2.4
DEVICE COUNT: 5
SSID: NIST 5
DEVICE COUNT: 6

1542 2. Click **Network Rules** to navigate to the rules menu:

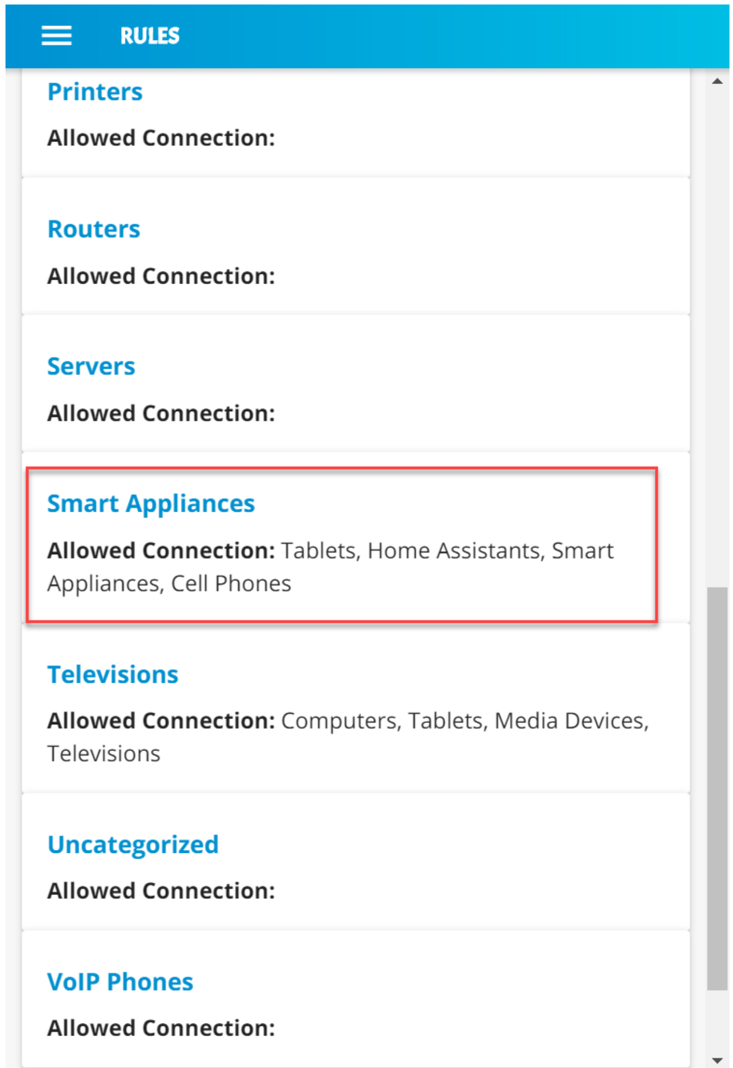


1543

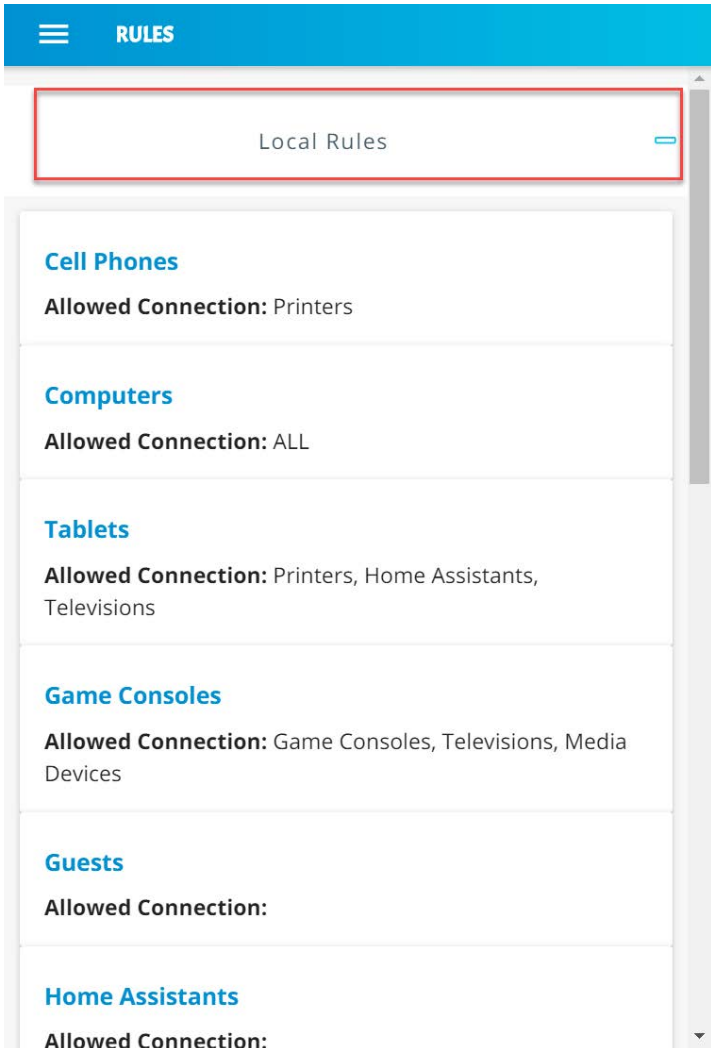
3. Click **Local Rules** to view the permitted local communications for each device category:



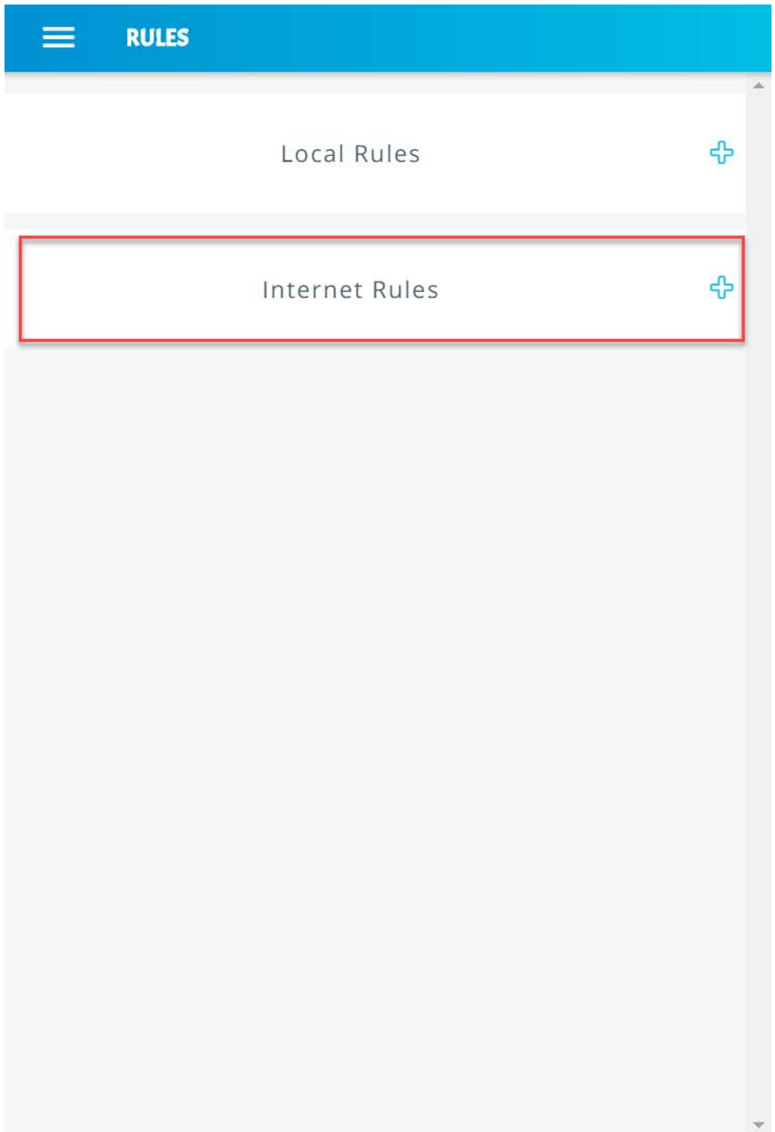
1546 4. Scroll down to view the local rules for the **Smart Appliances** category:

1547 

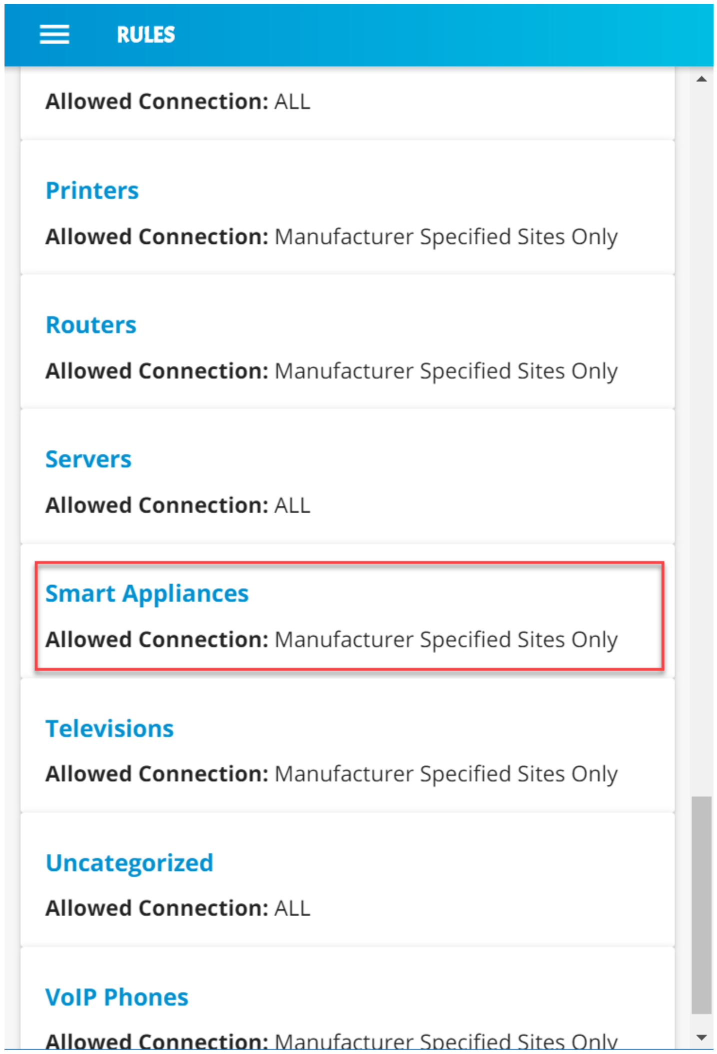
5. Minimize the rules by clicking on the **Local Rules** button:



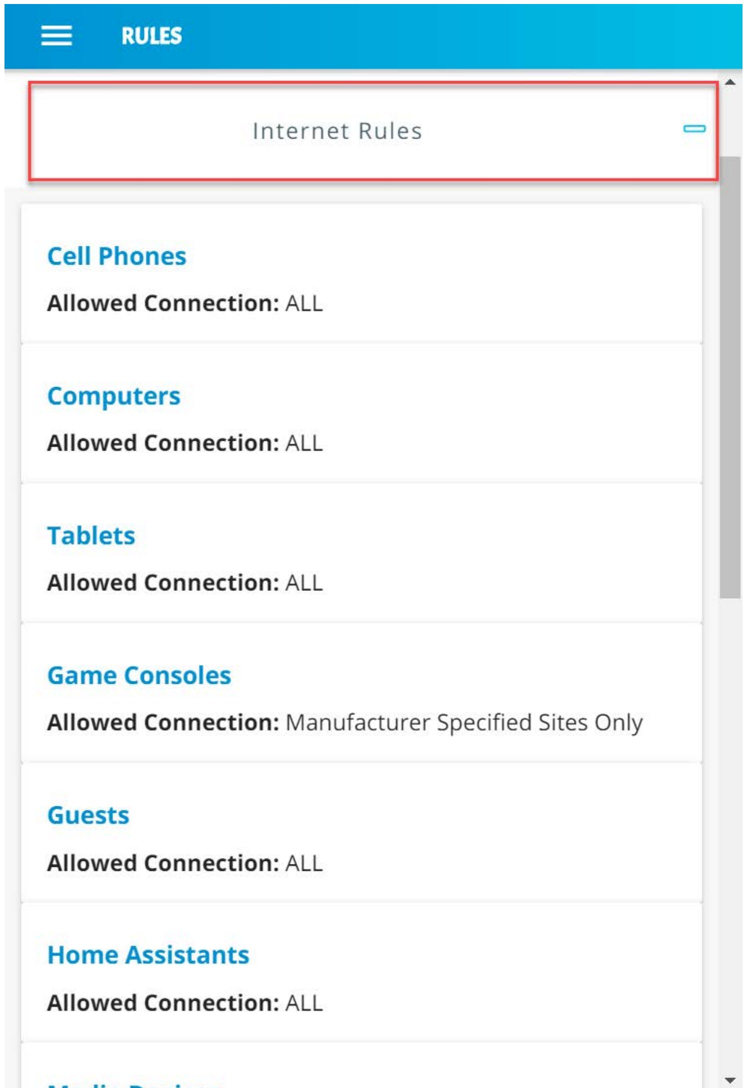
1550 6. Expand the rules that show internet rules for device categories by clicking **Internet Rules:**

1551 

7. Scroll down to view the internet rules for the **Smart Appliances** category:



8. Minimize the rules by clicking on 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 upon connection of the Yikes! router 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

Because Build 3 is still under development, instructions for installing and configuring its components are not yet provided. Those instructions are planned for inclusion in the guide that will be published for the next phase of this project. For a brief description of the planned architecture of Build 3, please refer to NIST SP 1800-15B.

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

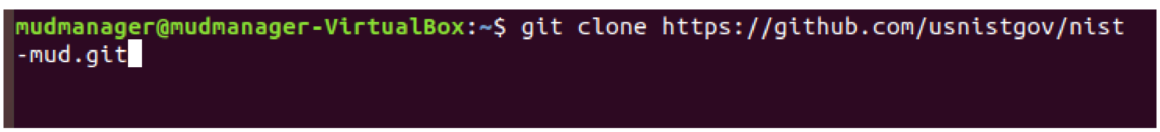
Build 4's GitHub page provides documentation that was followed to complete this section: https://github.com/usnistgov/nist-mud.

- 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

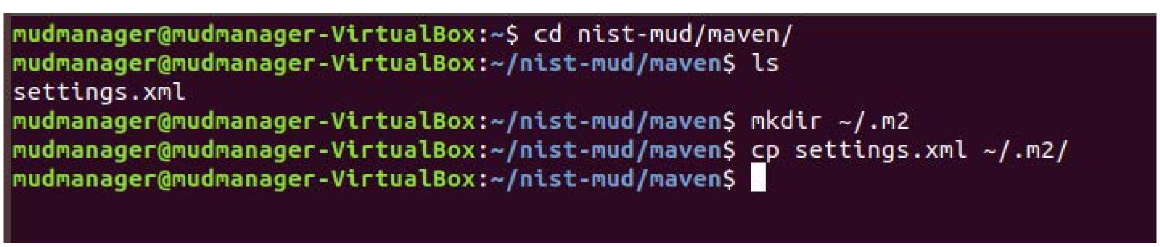
1. Execute the following command to clone the Git project:

```
git clone https://github.com/usnistgov/nist-mud.git
```



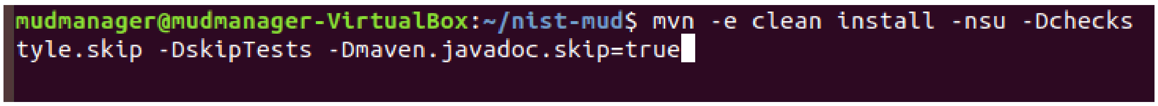
2. Copy the contents of `nist-mud/maven/settings.xml` to `~/.m2` by executing the commands below:

```
cd nist-mud/maven/
mkdir ~/.m2
cp settings.xml ~/.m2
```



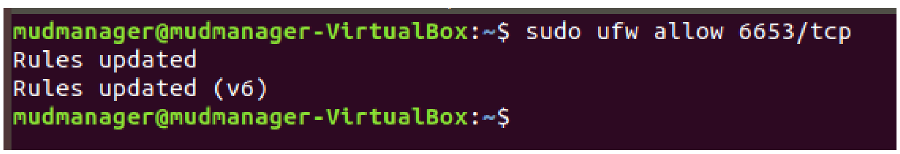
3. In the nist-mud directory, run the commands below:

```
cd
cd nist-mud/
mvn -e clean install -nsu -Dcheckstyle.skip -DskipTests -
Dmaven.javadoc.skip=true
```



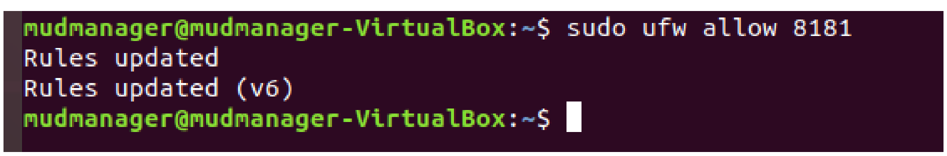
4. Open port 6653 on the controller stack for TCP access so the switches can connect by executing the command below:

```
sudo ufw allow 6653/tcp
```



5. OpenDaylight uses port 8181 for the Representational State Transfer (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
```

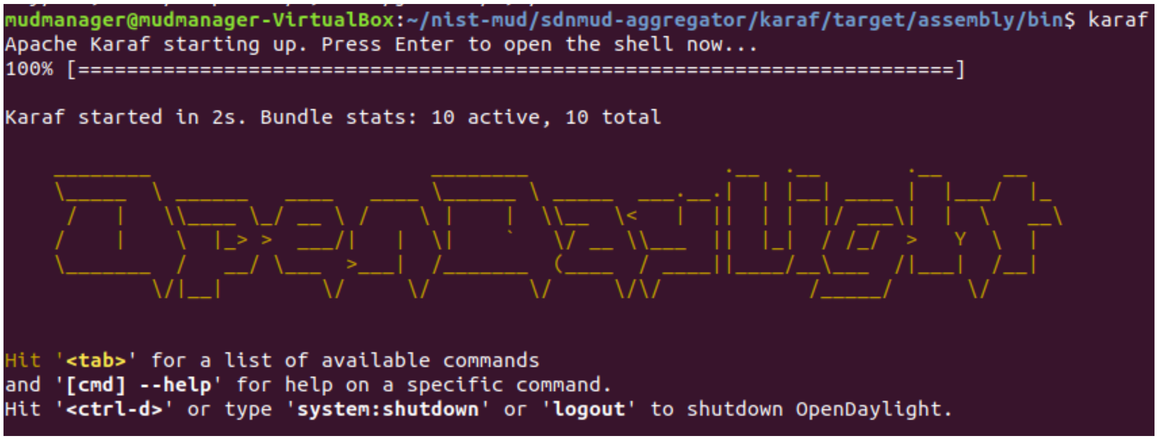


6. Change to the bin directory by executing the command below:

```
~/nist-mud/sdnmud-aggregator/karaf/target/assembly/bin
```

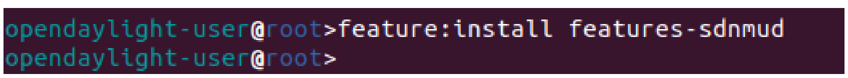
7. Run the command below:

```
./karaf clean
```

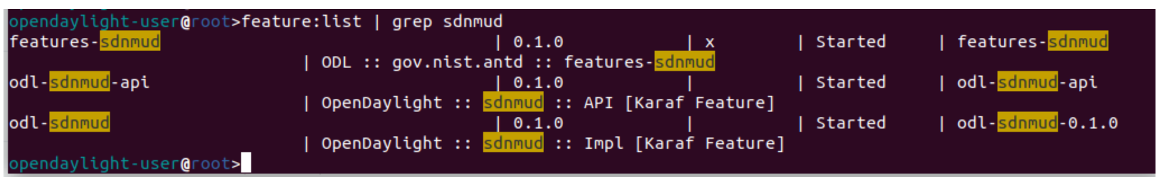


8. At the Karaf prompt, install MUD capabilities using:

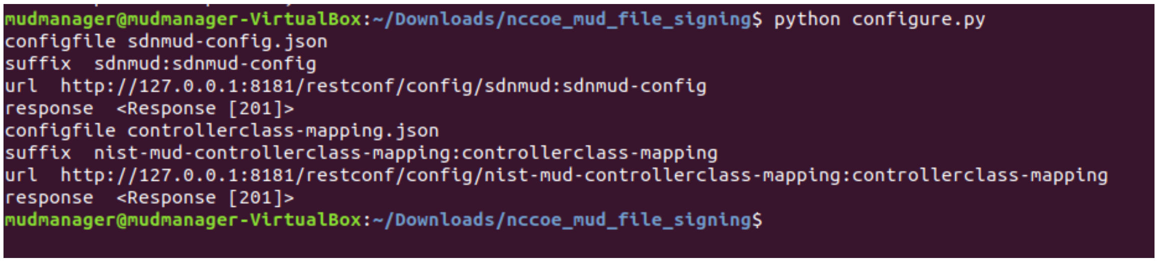
   `feature:install features-sdnmud`



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.

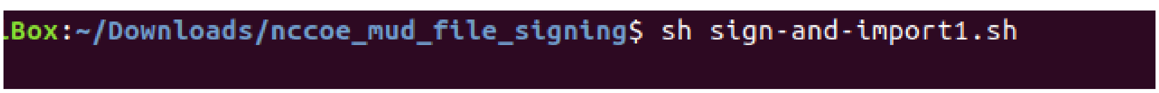
5.2.3 Setup

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
   ```

   ```
   rtualBox:~/Downloads/nccoe_mud_file_signing$ 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:

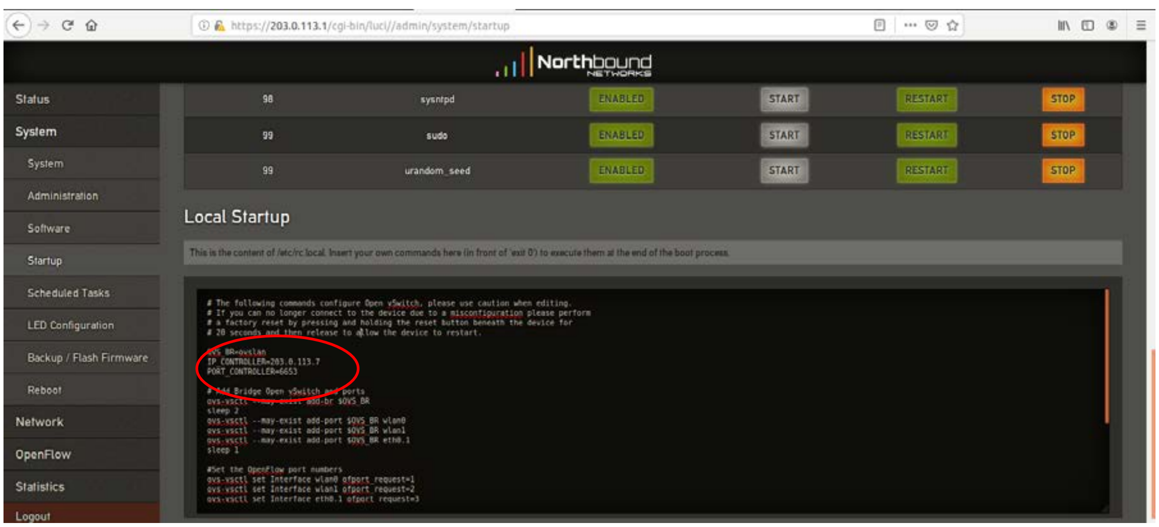
   ```
   addn-hosts=/etc/hosts.nist.local
   ```

   ```
   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
ACE	Access Control Entry
ACK	Acknowledgment
ACL	Access Control List
API	Application Programming Interface
CMS	Cryptographic Message Syntax
COA	Change of Authorization
CoAP	Constrained Application Protocol
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
FIPS	Federal Information Processing Standard
GCA	Global Cyber Alliance
GUI	Graphical User Interface
http	Hypertext Transfer Protocol
https	Hypertext Transfer Protocol Secure
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
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
NTP	Network Time Protocol
OS	Operating System
PC	Personal Computer
PoE	Power over Ethernet
RADIUS	Remote Authentication Dial-In User Service
REST	Representational State Transfer
RFC	Request for Comments
RMF	Risk Management Framework
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SP	Special Publication
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TEAP	Tunnel Extensible Authentication Protocol
TFTP	Trivial File Transfer 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
WPA2	Wi-Fi Protected Access 2 Security Certificate Protocol (IEEE 802.11i-2004 standard)
WPA3	Wi-Fi Protected Access 3 Security Certificate protocol
YANG	Yet Another Next Generation

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 operating system 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.

PRELIMINARY 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

Eliot Lear
Cisco

Adnan Baykal
Global Cyber Alliance

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

November 2019

PRELIMINARY 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:

- NIST Special Publication (SP) 1800-15A: *Executive Summary*
- NIST SP 1800-15B: *Approach, Architecture, and Security Characteristics*—what we built and why
- NIST SP 1800-15C: *How-To Guides*—instructions for building the example solutions

This document, *Functional Demonstration Results,* is a supplement to NIST SP 1800-15B, *Approach, Architecture, and Security Characteristics*. This proof-of-concept document describes the functional demonstration results for three 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.* Four builds are implemented, one of which is still under development. The functional demonstration results of three of these builds are reported in this document:

- 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 to onboard devices and support MUD. Although limited functionality of a preliminary version of this build has been demonstrated as part of this project, elements of Build 3 are still under development. Therefore, it has not yet been subjected to functional evaluation or demonstration of the full range of its capabilities.
- 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, 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 three builds are documented below.

1.2 Functional Demonstration Activities

Builds 1, 2, and 4 were tested to determine the extent to which they correctly implement basic functionality defined within the MUD RFC. Builds 1 and 2 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 include device discovery, identification and classification, and support for threat signaling.

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
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.

Test Case Field	Description
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, 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 pathnames; references to documents that are not hyperlinks; new terms; and placeholders	For detailed definitions of terms, see the *NCCoE Glossary.*
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, 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

205 ## 2.1.2 Test Cases

206 This section contains the test cases that were used to verify that Build 1 met the requirements listed in
207 Table 2-1.

208 ### *2.1.2.1 Test Case IoT-1-v4*

209 **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.

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>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.</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 details:
<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 1–7 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.</td></tr>
<tr><td>Actual Results</td><td><u>Dynamic access-session on switch</u>:</td></tr>
</table>

Test Case Field	Description
	`` 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 ``` **<u>access-list on switch</u>:** ``` 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 ```
Overall Results	Pass

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.

217 ### 2.1.2.3 Test Case IoT-3-v4

218 **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 Subcate-gory(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.

<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 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:
<pre>
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
</pre></td></tr>
</table>

<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

223 As explained above, test IoT-4-v6 is identical to test IoT-4-v4 except that it uses IPv6, DHCPv6, and IANA
224 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

Test Case Field	Description
	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)
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

Test Case Field	Description
	i) the IoT device to initiate communication with *https://yes-permit-from.com* 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 strikethrough are 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,

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>but it is not forwarded by the MUD PEP, nor is it received at the IoT device. (ingress)</td></tr>
<tr><td>Expected Results</td><td>Each of the results that is listed as needing to be verified in procedure steps above occurs as expected.</td></tr>
<tr><td>Actual Results</td><td>Procedure 2:
Connection to update server successfully initiated by IoT device:
<pre>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]</pre><hr>Procedure 3:
Update server failed to connect to IoT device:
<pre>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.</pre><hr>Procedure 6:
IoT device failed to connect to unapproved server:
<pre>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.</pre></td></tr>
</table>

Test Case Field	Description
	Procedure 7: Unapproved server attempts to connect to IoT device: `[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)

227 As explained above, test IoT-5-v6 is identical to test IoT-5-v4 except that it uses IPv6, DHCPv6, and IANA
228 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

229 *2.1.2.6 Test Case IoT-6-v4*

230 **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.

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 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
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 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	Note: Procedure steps with strikethrough are 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*.

Test Case Field	Description
	~~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 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	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]`

Test Case Field	Description
	`Saving to: 'index.html.4'` `index.html.4      100%[===================>]  10.45K  --.-KB/s    in 0s` `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.`

Test Case Field	Description
	`HTTP request sent, awaiting response... 200 OK` `Length: 10701 (10K) [text/html]` `Saving to: 'index.html.12'` `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.`

Test Case Field	Description
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.

2.1.2.7 Test Case IoT-7-v4

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

Test Case Field	Description
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.
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`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
Server Policies:
            ACS ACL: mud-81726-v4fr.in
         Vlan Group: Vlan: 3

Method status list:
      Method   State
        mab   Authc Success
</pre>
<hr>
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)

Test Case Field	Description
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). 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

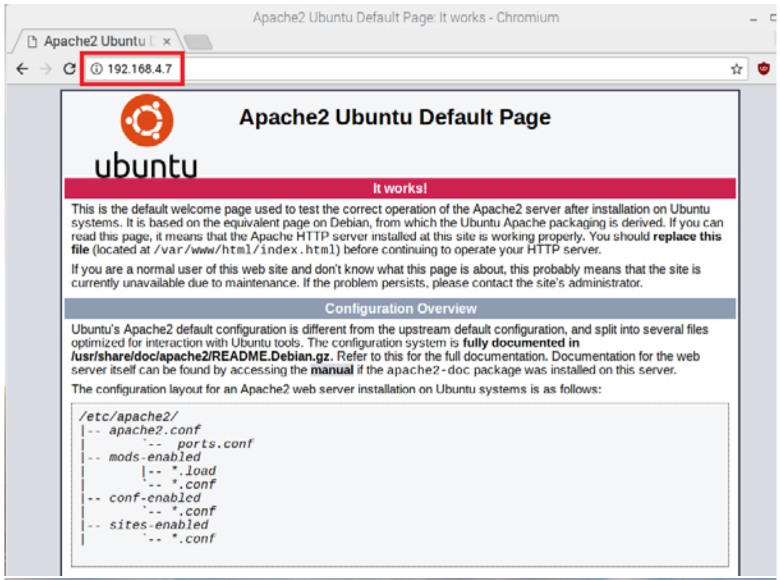
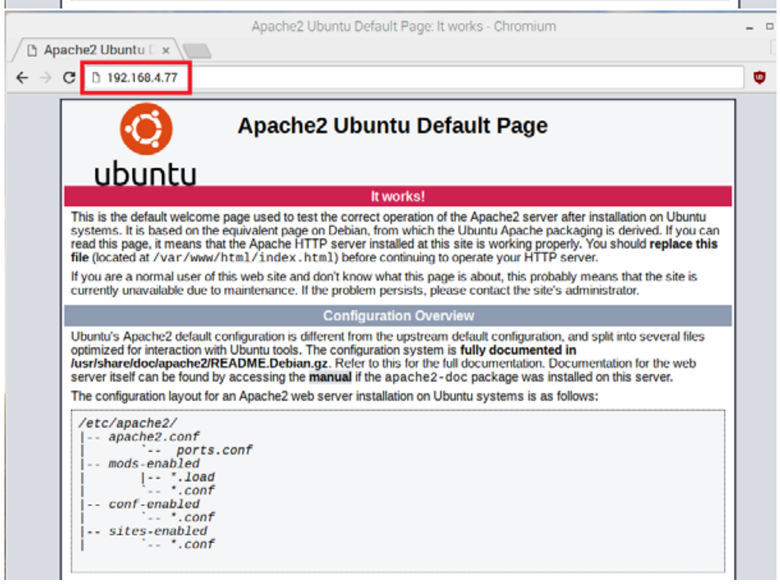
Test Case Field	Description
Parent Requirements	(CR-13) The IoT DDoS example implementation shall ensure that for each rule in a MUD file that pertains to an external domain, the MUD PEP router/switch will get configured with all possible instantiations of that rule, insofar as each instantiation contains one of the IP addresses

Test Case Field	Description
	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 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*.)

Test Case Field	Description
	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. 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`

Test Case Field	Description
	`***MUDC [INFO][validate_muduri:3009]--> uri:` *https://mudfileserver/dnstest.jsonhttps://mudfileserver/dnstest.json* `***MUDC [INFO][validate_muduri:3035]--> ip: mudfileserver, filename: dnstest.json` `***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:**

Test Case Field	Description
	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 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:**

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.

245 ### 2.1.2.10 Test Case IoT-10-v4

246 **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
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	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. 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

Test Case Field	Description
	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)**:** `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` **Cache is valid** (the MUD manager does NOT retrieve the MUD file from the MUD file server)**:** `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`

Test Case Field	Description
	`83 deny ip any any` **Cache is not valid** (the MUD manager does retrieve the MUD file from the MUD file server)**:** `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 1–9 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	**MUD manager logs for valid 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 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` `***MUDC [INFO][mudc_print_request_info:2192]--> remote ip addr: 0xe7719c38` `***MUDC [INFO][mudc_print_request_info:2193]--> remote port: 49344`

Test Case Field	Description
	***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" : { "$oid" : "5c182c7edb40218cde918776" }, "URI" : "*https://mudfileserver/ciscopi2*" }> ***MUDC [INFO][fetch_uri_from_macaddr:1711]--> ============= Returning URI:*https://mudfileserver/ciscopi2* ***MUDC [INFO][handle_get_aclname:2513]--> Found URI *https://mudfileserver/ciscopi2* for MAC address b827ebeb6c8b ***MUDC [INFO][validate_muduri:2373]--> uri: *https://mud-fileserver/ciscopi2* ***MUDC [INFO][validate_muduri:2399]--> ip: mudfileserver, filename: ciscopi2 ***MUDC [INFO][handle_get_aclname:2558]--> Got URL from mes-sage <*https://mudfileserver/ciscopi2*> ***MUDC [INFO][query_policies_by_uri:1419]--> found the rec-ord <{ "_id" : { "$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:in-acl#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 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

Test Case Field	Description
	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**> ***MUDC [INFO][fetch_uri_from_macaddr:1702]--> found the fields <{ "_id" : { "$oid" : "5c182c7edb40218cde918776" }, "URI" : "*https://mudfileserver/ciscopi2*" }>

Test Case Field	Description
	`***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 mes-`** **`sage <`*`https://mudfileserver/ciscopi2`*`>`** `***MUDC [INFO][query_policies_by_uri:1399]-->` **`Cache has ex-`** **`pired`** **[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-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 HTTP/1.1`** `Host: mudfileserver` `Accept: */*`

Test Case Field	Description
	[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

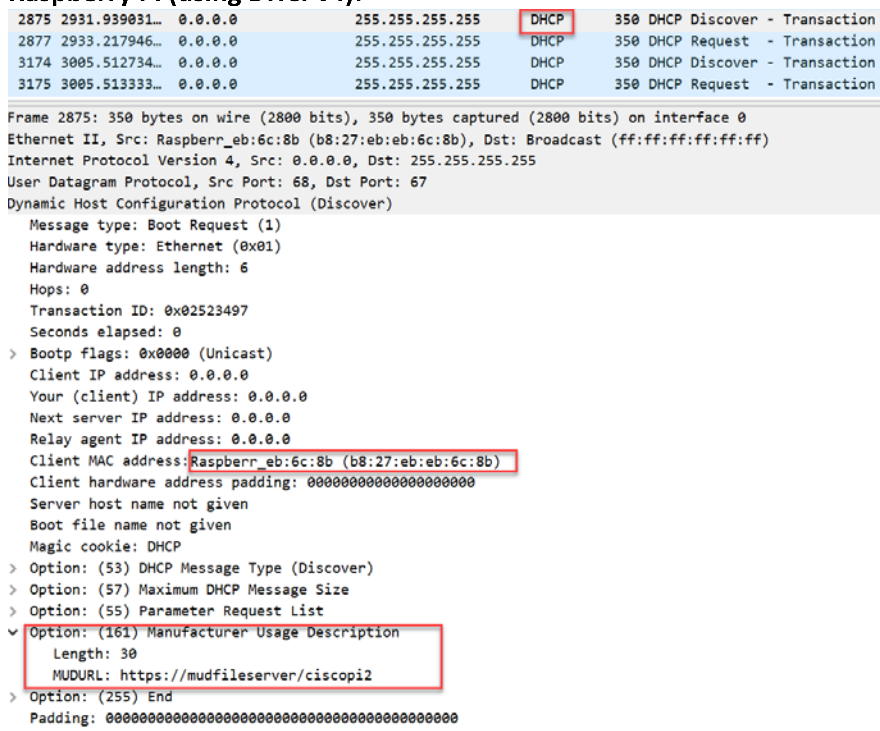
Table 2-12: Test Case IoT-11-v4

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.
Description	Shows that the IoT DDoS example implementation includes IoT devices that can emit a MUD URL via DHCP or LLDP

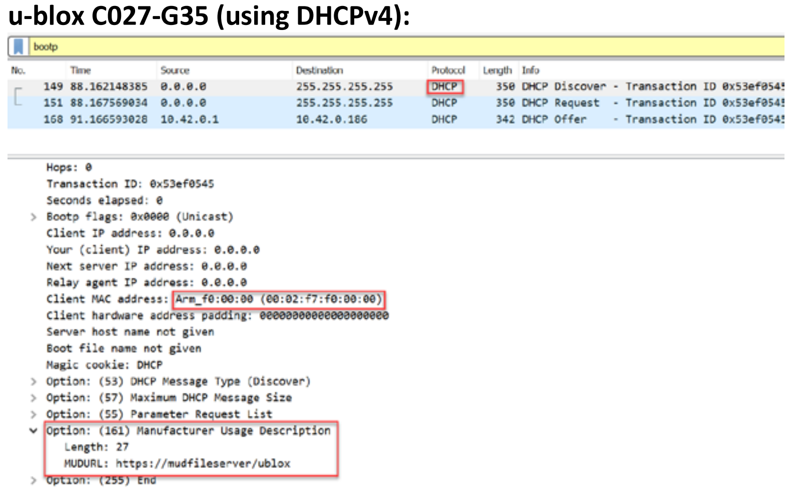
Test Case Field	Description
Associated Test Case(s)	N/A
Associated Cybersecurity Framework Subcate-gory(ies)	ID.AM-1
IoT Device(s) Under Test	Raspberry Pi, Molex light engine, u-blox C027-G35
MUD File(s) Used	*Ciscopi2.json, molex.json, ublox.json*
Preconditions	Device has been developed to emit a MUD 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):

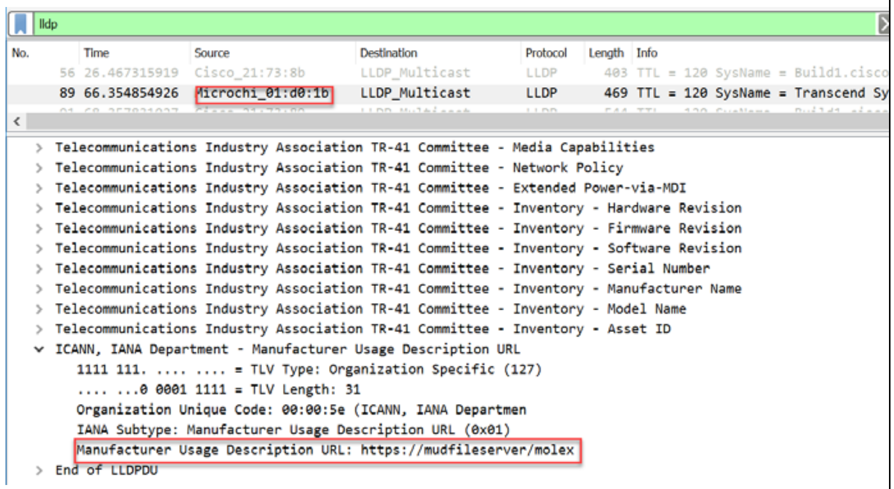


u-blox C027-G35 (using DHCPv4):



Molex light engine (using LLDP):

</td></tr>
</table>

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
</td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

252

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.

2.2.1 Non-MUD-Related Functional Capabilities Demonstrated

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

2.2.2 Exercises to Demonstrate the Above Non-MUD-Related Capabilities

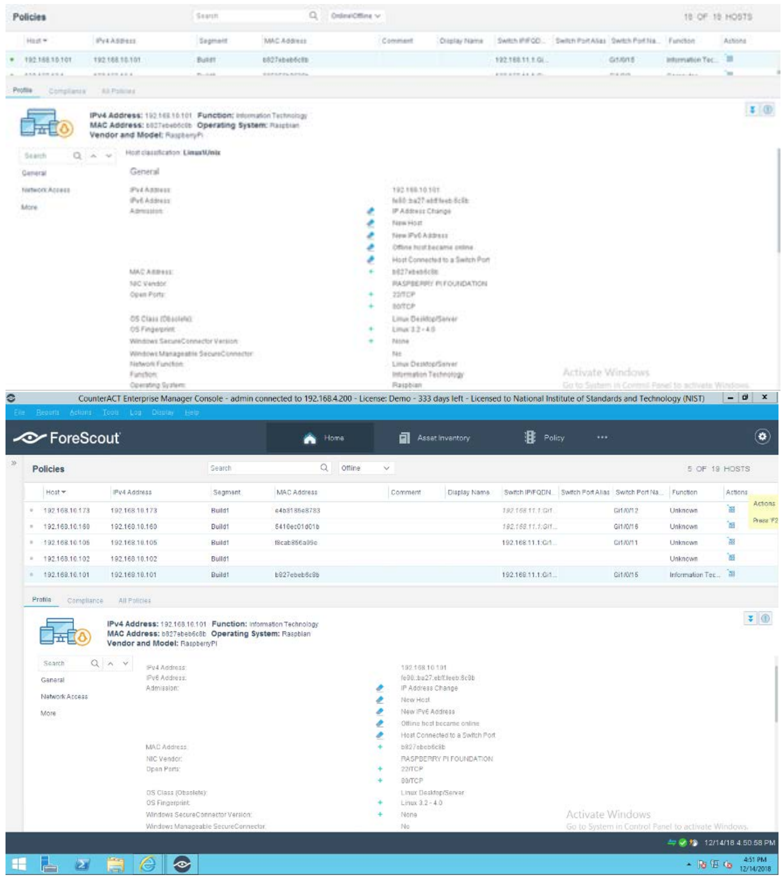
This section contains the exercises that were performed to verify that Build 1 supports the non-MUD-related capabilities listed in Table 2-13.

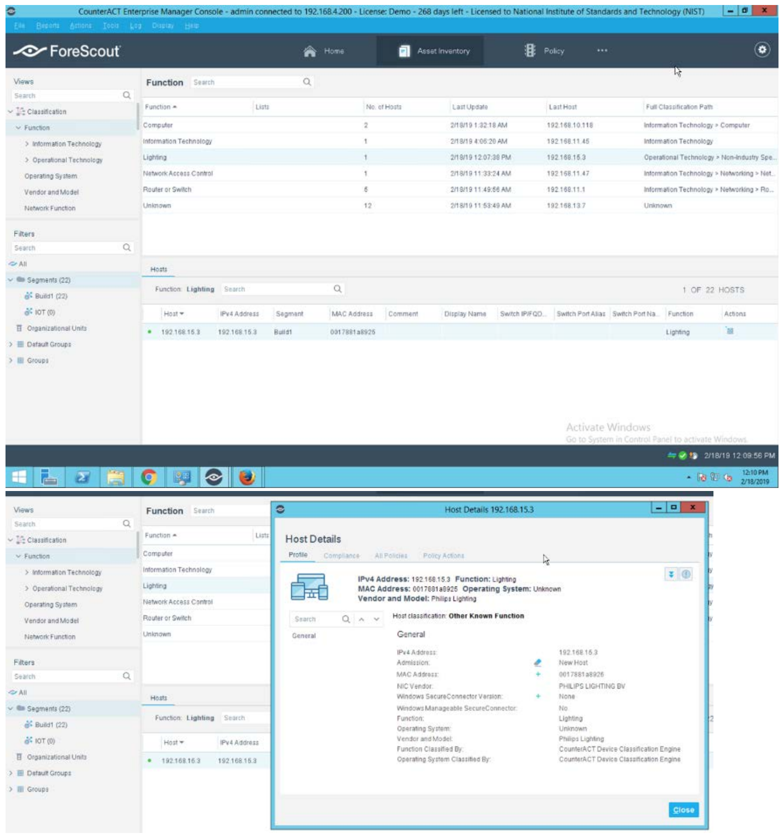
285 ### 2.2.2.1 Exercise CnMUD-13-v4

286 **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.

Test Case Field	Description
	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.
Expected Results	All expectations as enumerated in items 2, 4, 6, and 8 above are observed.
Actual Results	**At Power-On:** `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` **Screenshot from Forescout:** IoT device status is indicated by green or gray light shown in the screen capture

Test Case Field	Description
	 **Categorizing IoT Device:** We tested this function with a smart light bulb. See the example screen-shots below.

Test Case Field	Description
	
Overall Results	Pass

287 Test case CnMUD-13-v6 is identical to test case CnMUD-13-v4 except that test case CnMUD-13-v6 uses
288 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.)

Test Case Field	Description
	(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. (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

Test Case Field	Description
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 (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.

Test Case Field	Description
	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. 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`

Test Case Field	Description
	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 config rule option enabled '1' option name 'mud_192.168.20.222_main-pi-Build2_cl2-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 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 option src_ip 99.84.216.27 option dest_ip 192.168.20.222 option dest_port 443:443 config rule ```

Test Case Field	Description
	`` 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 option dest wan option proto tcp option family ipv4 option src_ip 192.168.20.222 ```

Test Case Field	Description
	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-Build2_loc0-frdev' option target ACCEPT option src lan

Test Case Field	Description
	`` 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- Build2_myctl0-frdev' option target ACCEPT ```

Test Case Field	Description
	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' option target REJECT

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
        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
</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.
</td></tr>
<tr><td>Actual Results</td><td>
Procedures 1–3:
<pre>
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.
</pre>
</td></tr>
</table>

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
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 192.168.20.222 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.
</pre>
Procedures 4–5:
dhcpmasq.txt
<pre>
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||
</pre>
Procedure 6:
MUD MANAGER:
<pre>
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|
</pre>
</td></tr>
</table>

Test Case Field	Description
	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 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

Test Case Field	Description
	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 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

Test Case Field	Description
	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 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

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 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 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

Test Case Field	Description
	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 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)

Test Case Field	Description
	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 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

Test Case Field	Description
	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' 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

Test Case Field	Description
	``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 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-```

Test Case Field	Description
	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 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

Test Case Field	Description
	`` 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 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 ```

Test Case Field	Description
	` 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-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`

Test Case Field	Description
	`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` `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`

Test Case Field	Description
	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' 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

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 IoT-1-v6 uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

3.1.2.2 Test Case IoT-2-v4

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

Test Case Field	Description
	`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

310 As explained above, test IoT-2-v6 is identical to test IoT-2-v4 except that it uses IPv6, DHCPv6, and IANA
311 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

312 *3.1.2.3 Test Case IoT-3-v4*

313 **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_perform() 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**** validateMudFileWithSig() 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 -purpose 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/ExpiredCertTest.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/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 18:03:27 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 18:03:27 DEBUG::GENERAL::/etc/osmud/create_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 ACCEPT -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 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

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
	<pre> 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</pre>

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

314 As explained above, test IoT-3-v6 is identical to test IoT-3-v4 except that it uses IPv6, DHCPv6, and IANA
315 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

316 ### 3.1.2.4 Test Case IoT-4-v4

317 **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 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`

Test Case Field	Description
	`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`
Actual Results	**Procedures 1-5:** Excluded for sake of length. **Procedure 6:** **MUD MANAGER:** `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....`

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

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>

<pre>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</pre>

Procedure 7:

Router/PEP:

<pre># 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</pre>
</td></tr>
</table>

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-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 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

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>
       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-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.

3.1.2.5 Test Case IoT-5-v4

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.

Test Case Field	Description
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	*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 strikethrough are 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 strikethrough are 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.

Test Case Field	Description
	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 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`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
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.6’

0K ....
15.7M=0.001s

2019-07-11 18:23:38 (15.7 MB/s) - ‘index.html.6’ saved [11449]
<hr>
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
</td></tr>
</table>

Test Case Field	Description
	`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/` `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):

Test Case Field	Description
	`--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` **`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)

As explained above, test IoT-5-v6 is identical to test IoT-5-v4 except that it uses IPv6, DHCPv6, and IANA code 112 instead of using IPv4, DHCPv4, and IANA code 161.

3.1.2.6 Test Case IoT-6-v4

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.

Test Case Field	Description
	(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 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

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>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 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 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</td></tr>
</table>

Test Case Field	Description
	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 (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

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>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 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%[======================================>]</pre></td></tr>
</table>

Test Case Field	Description
	`10,701      --.-K/s   in 0s` `2019-07-24 15:30:01 (139 MB/s) - 'index.html' saved [10701/10701]` 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'`

Test Case Field	Description
	`0K 100% 10.9M=0s` `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/`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
Connecting to 192.168.20.183:80... connected.

HTTP request sent, awaiting response... 200 OK

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]
</pre>
<hr>
Procedure 5 (from laptop to pi):
<pre>
[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.
</pre>
<hr>
Procedure 6 (from device):
http://www.facebook.com (unapproved):
<pre>
--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|:4
43... failed: Network is unreachable.
</pre>
<hr>
Procedure 7 (from laptop to Pi):
<pre>
[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.
</pre>
<hr>
</td></tr>
</table>

Test Case Field	Description
	Controller: Procedure 4 (from Pi to controller): 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/`*

Test Case Field	Description
	Connecting to 192.168.20.110:80... failed: Connection refused. *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.**

Test Case Field	Description
	Procedure 7 (from laptop to Pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-10 17:50:22--  http://192.168.20.222/` **`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/`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>Connecting to 192.168.20.117:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.142/ (unapproved):
<pre>--2019-07-10 18:30:26-- http://192.168.20.142/
Connecting to 192.168.20.142:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.171/ (unapproved):
<pre>--2019-07-10 18:29:55-- http://192.168.20.171/
Connecting to 192.168.20.171:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.181/ (unapproved):
<pre>--2019-07-10 18:29:08-- http://192.168.20.181/
Connecting to 192.168.20.181:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.183/ (unapproved):
<pre>--2019-07-10 18:29:23-- http://192.168.20.183/
Connecting to 192.168.20.183:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.197/ (unapproved):
<pre>--2019-07-10 18:28:32-- http://192.168.20.197/
Connecting to 192.168.20.197:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.232/ (unapproved):
<pre>--2019-07-10 18:30:36-- http://192.168.20.232/
Connecting to 192.168.20.232:80... failed: Connection refused.</pre>
<hr>
http://192.168.20.247/ (unapproved):
<pre>--2019-07-10 18:28:45-- http://192.168.20.247/
Connecting to 192.168.20.247:80... failed: Connection refused.</pre>
<hr>
Procedure 7 (from laptop to Pi):
<pre>[mud@localhost ~]$ wget 192.168.20.222
--2019-07-10 18:29:13-- http://192.168.20.222/</pre>
</td></tr>
</table>

Test Case Field	Description
	Connecting to 192.168.20.222:80... failed: Connection refused.
	Same Manufacturer 1 (.197): Procedure 4 (from device): *http://192.168.20.222/* (approved): `--2019-07-12 16:04:46--  `*`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.9'` `    0K ..........` `100%  104K=0.1s` `2019-07-12 16:04:46 (104 KB/s) - 'index.html.9' saved [10701/10701]`
	Procedure 5 (from laptop to pi): `[mud@localhost ~]$ wget 192.168.20.222` `--2019-07-12 16:08:28--` http://192.168.20.222/ **`Connecting to 192.168.20.222:80... failed: Connection refused.`**
	Procedure 6 (from device): http://192.168.20.232/ (unapproved): `--2019-07-12 16:06:35--  `*`http://192.168.20.232/`* **`Connecting to 192.168.20.232:80... failed: Connection refused.`**
	http://192.168.20.110:443/ (unapproved): `--2019-07-12 16:06:16--  `*`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 16:06:01--  `*`http://192.168.20.117/`* **`Connecting to 192.168.20.117:80... failed: Connection refused.`**

Test Case Field	Description
	http://192.168.20.181/ (unapproved): `--2019-07-12 16:05:39--  http://192.168.20.181/` **`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.`**

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
Procedure 6 (from device):
http://192.168.20.110:443/ (unapproved):
<pre>
--2019-07-12 15:58:13-- http://192.168.20.110:443/
Connecting to 192.168.20.110:443... failed: Connection refused.
</pre>
http://192.168.20.117/ (unapproved):
<pre>
--2019-07-12 15:57:19-- http://192.168.20.117/
Connecting to 192.168.20.117:80... failed: Connection refused.
</pre>
http://192.168.20.232/ (unapproved):
<pre>
--2019-07-12 15:57:29-- http://192.168.20.232/
Connecting to 192.168.20.232:80... failed: Connection refused.
</pre>
http://192.168.20.197 (unapproved):
<pre>
--2019-07-12 15:58:35-- http://192.168.20.197/
Connecting to 192.168.20.197:80... failed: Connection refused.
</pre>
Procedure 7 (from laptop to Pi):
<pre>
[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.
</pre>
Same Manufacturer:
Procedure 8 (from device):
http://192.168.20.197/ (approved):
<pre>
--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
</pre>
</td></tr>
</table>

Test Case Field	Description
	`2019-07-12 16:27:24 (3.75 MB/s) - 'index.html.43' saved [10701/10701]` 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

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.

3.1.2.7 Test Case IoT-7-v4

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*

Test Case Field	Description
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. 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.`

Test Case Field	Description
	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/*** **2019-07-11 18:57:30 DEBUG::GENERAL::cp /etc/osmud/state/ipSets/* /tmp/osmud** **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/remove_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/remove_from_ipset.sh -d /tmp/osmud -i 192.168.20.226** **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/commit_ip_fw_rules.sh -d /etc/osmud/state/ipSets -t /tmp/osmud** **2019-07-11 18:57:30 DEBUG::GENERAL::/etc/osmud/remove_mud_db_entry.sh -d /etc/osmud/state/mudfiles/mudStateFile.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

Test Case Field	Description
	`` 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 option enabled '1' ```

Test Case Field	Description
	`option name 'mud_192.168.20.197_same-manufacture-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-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`

Test Case Field	Description
	`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` `        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

330 As explained above, test IoT-7-v6 is identical to test IoT-7-v4 except that it uses IPv6, DHCPv6, and IANA
331 code 112 instead of using IPv4, DHCPv4, and IANA code 161.

332 #### *3.1.2.8 Test Case IoT-8-v4*

333 **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 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).

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 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/*

Test Case Field	Description
	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 /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

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-manufac- ture-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-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 ```

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>
        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-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

Test Case Field	Description
	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 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*.)

Test Case Field	Description
	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. 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`

Test Case Field	Description
	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 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**

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>

<pre><code>tcp -n cl2-frdev -t ACCEPT -f all -c main-pi-Build2 -k
/tmp/osmud -r 192.168.20.222</code></pre>

<pre><code>2019-07-15 20:28:33 DEBUG::GENERAL::/etc/osmud/cre-
ate_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
all -c main-pi-Build2 -k /tmp/osmud -r 192.168.20.222</code></pre>

[Logs omitted for brevity]

<pre><code>2019-07-15 20:28:33 DEBUG::GENERAL::Success returned from
for transaction</code></pre>

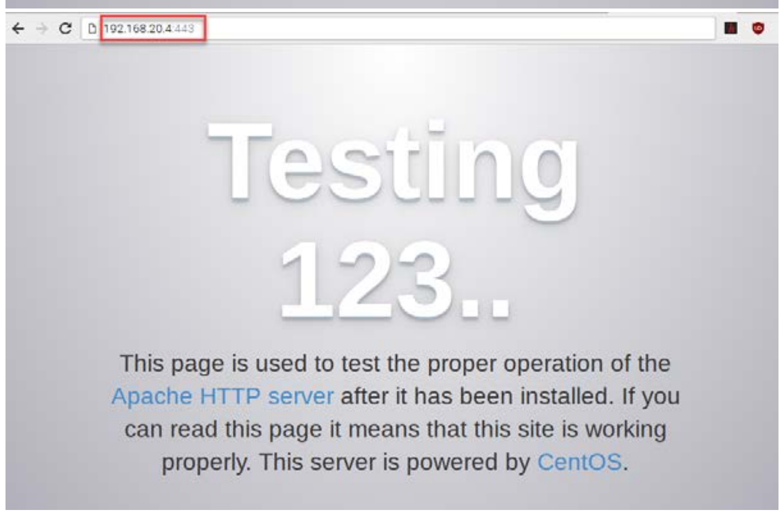
Router/PEP:

<pre><code>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</code></pre>

Procedure 4:

</td></tr>
</table>

Test Case Field	Description
	
Overall Results	Pass

338 Test case IoT-9-v6 is identical to test case IoT-9-v4 except that IoT-9-v6 uses IPv6 addresses rather than
339 IPv4 addresses.

3.1.2.10 Test Case IoT-10-v4

Table 3-111: 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

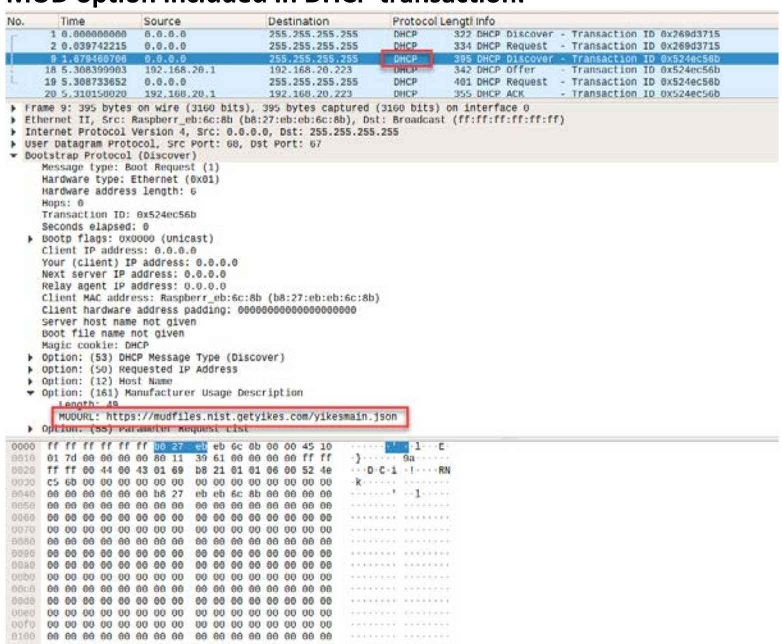
Test Case Field	Description
	file was retrieved is less than or equal to the number of hours in the cache-validity value for this MUD file. If the cache validity has been exceeded, the MUD manager will fetch a new MUD file. (Run the test both ways—with a cache-validity period that has expired and with 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

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td>Procedure</td><td>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.)</td></tr>
<tr><td>Expected Results</td><td>DHCP transaction with MUD option 161 enabled and MUD URL included</td></tr>
<tr><td>Actual Results</td><td>MUD option included in DHCP transaction:

</td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

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.

440 **Table 3-133: 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.			
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, Yn-MUD-1-v6
Y-1.b		The non-MUD-capable **device's make and model cannot be identified.**		YnMUD-1-v4, Yn-MUD-2-v6
Y-1.c		The non-MUD-capable **device's make and model can be assigned manually.**		YnMUD-2-v4, Yn-MUD-3-v6
Y-2	**Device Categorization–**The device is correctly categorized according to its type (e.g., phone, printer, computer, watch)			

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
	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, Yn-MUD-1-v6
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.**			

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
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, YnMUD-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 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**	YnMUD-3-v4, YnMUD-3-v6

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
			initiate communications with all internet sites.	
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 from receiving communications from all other internet sites.**	**N/A for IPv4** due to NAT
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**			

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
	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
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.**			

Functional Capability	Parent Capability	Subrequirement 1	Subrequirement 2	Exercise ID
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
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):

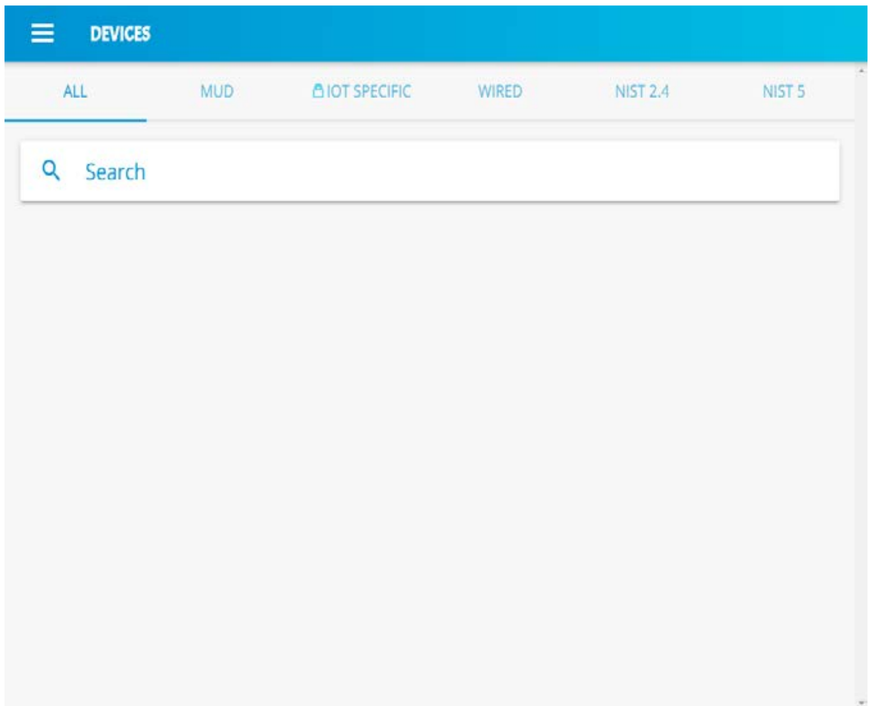
- www.google.com
- www.osmud.org
- www.trytechy.com

3.2.4.1 Exercise YnMUD-1-v4

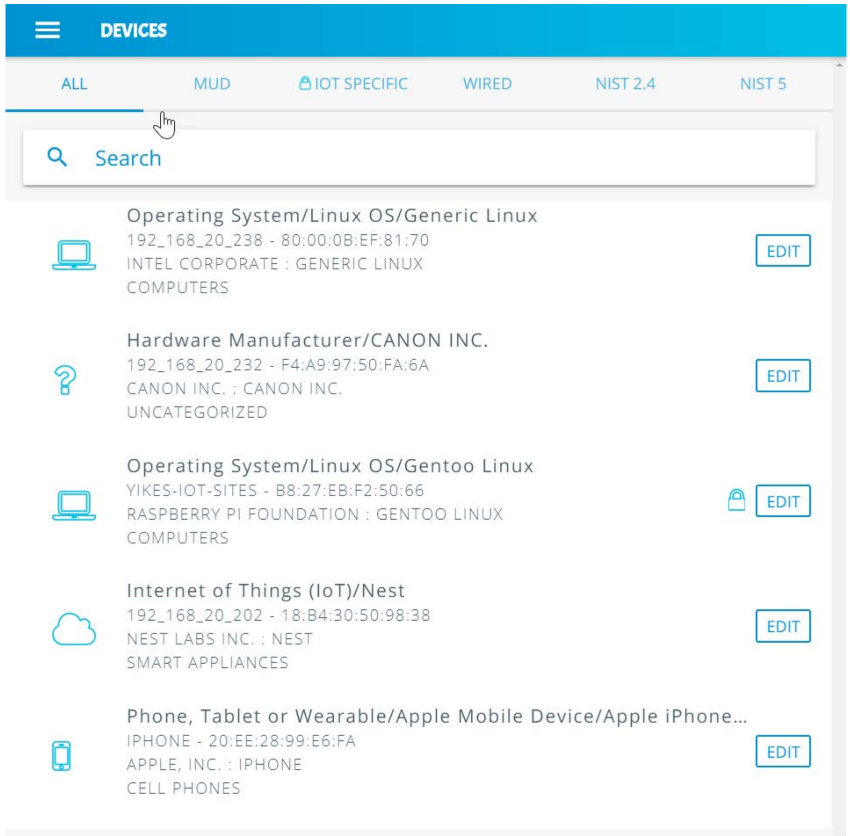
Table 3-144: Exercise YnMUD-1-v4

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 info, and lookup

Exercise Field	Description
	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
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

Exercise Field	Description
	and model correctly identified (if possible), and has been categorized appropriately: **Procedures 1–2:**  **Procedures 3–4:**

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>



<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>iPhone 7</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 Device</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-1-v6 is identical to exercise YnMUD-1-v4 except that it uses IPv6 instead of IPv4.

452 ### 3.2.4.2 Exercise YnMUD-2-v4

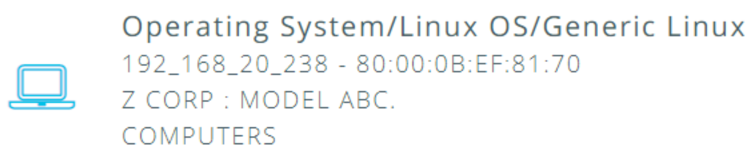
453 **Table 3-15: Exercise YnMUD-2-v4**

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.

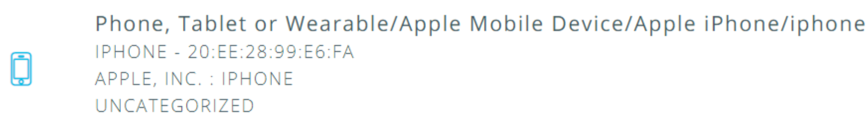
Exercise Field	Description
Demonstrated Results	Access the Yikes! UI, go to the Device tab, and verify that the following information is present:

Procedure 1: Completed; excluded for brevity

Procedures 2–3:



Procedure 4:



Device	Device ID	Make	Model	Category
Laptop	ID_1	Dell	E6540	Computer
Cell Phone	ID_2	Apple	iPhone7	Cell phone
Printer	ID_3	Canon	MX922	Uncategorized
Camera	ID_4	Nest	Indoor Cam	Smart Device
Test-PI	ID_5	Raspberry	Pi B+	Computer

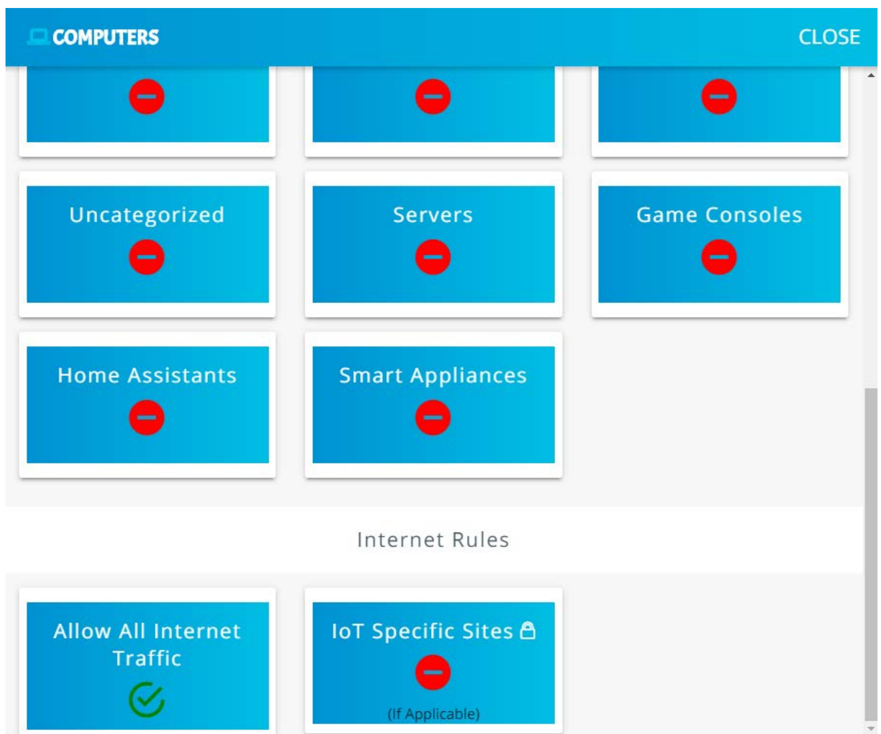
454 Exercise YnMUD-2-v6 is identical to exercise YnMUD-2-v4 except that it uses IPv6 instead of IPv4.

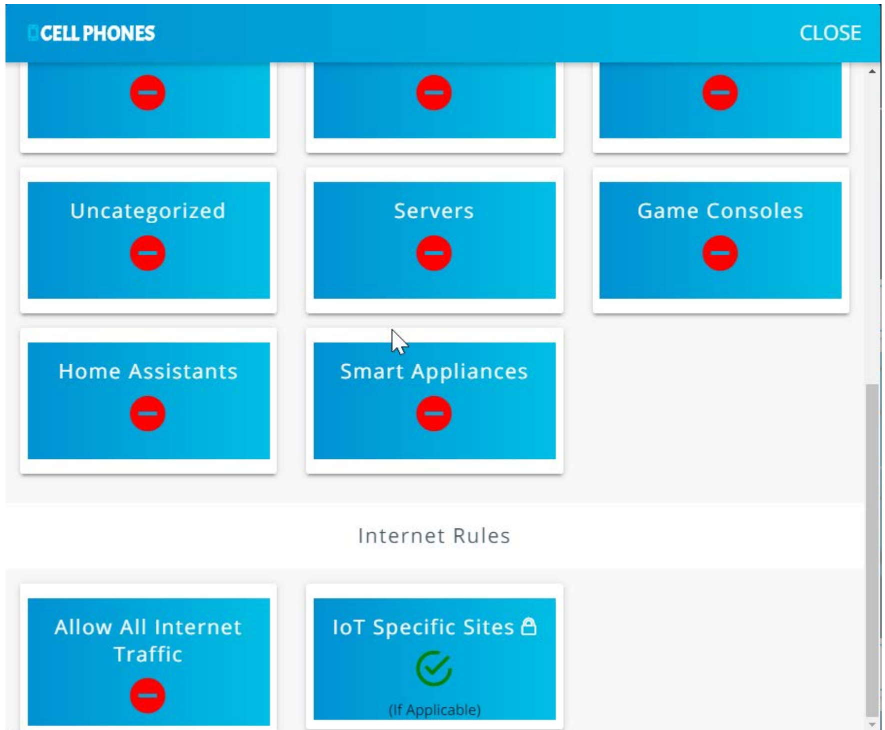
455 ### 3.2.4.3 Exercise YnMUD-3-v4

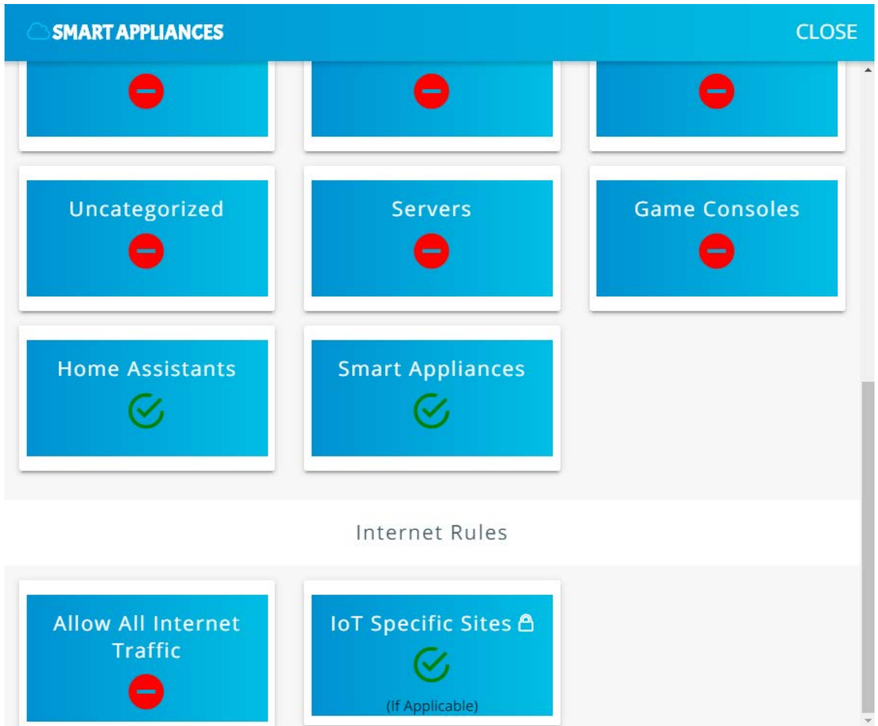
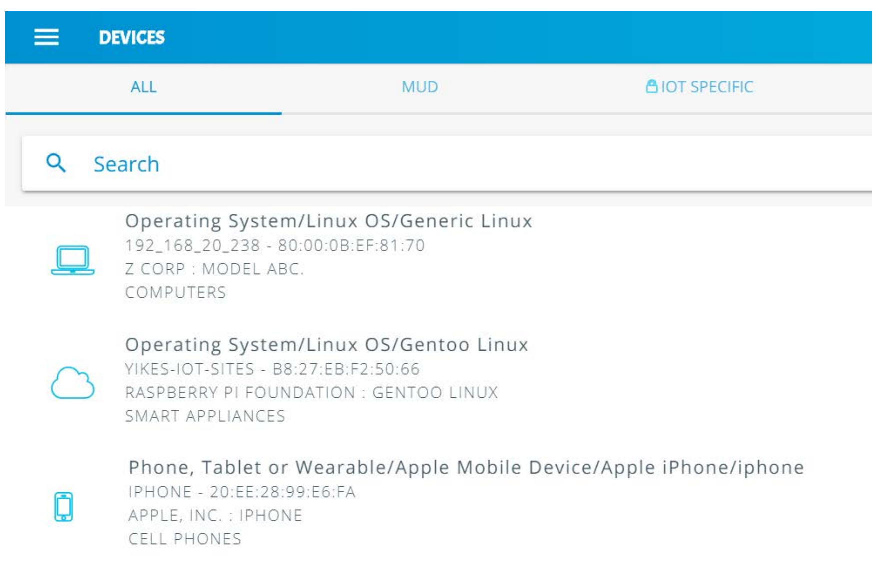
456 **Table 3-16: Exercise YnMUD-3-v4**

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 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.

Exercise Field	Description
	- 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
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 smart 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.

Exercise Field	Description
	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. 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**

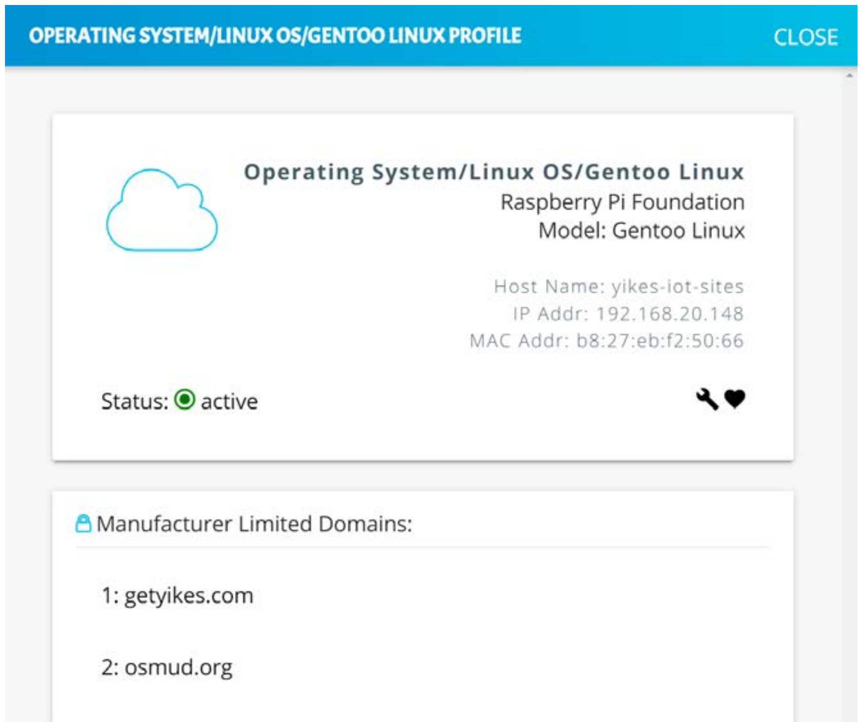
Exercise Field	Description
	 **Smart Appliances**

Exercise Field	Description
	 **Procedure 2:** 

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>

Procedure 3:

Smart Appliance



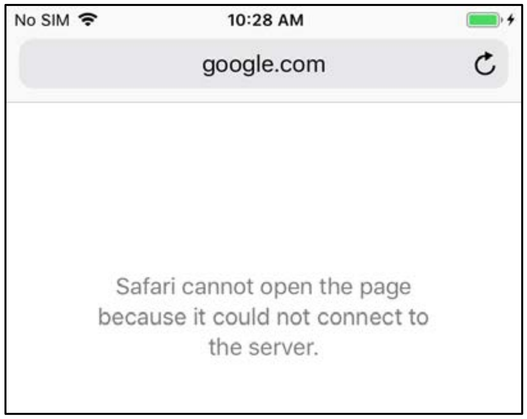
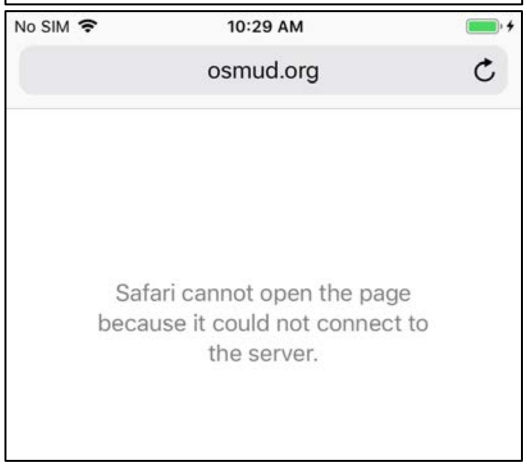
Yikes! approved communication:

<pre>
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]
</pre>
</td></tr>
</table>

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

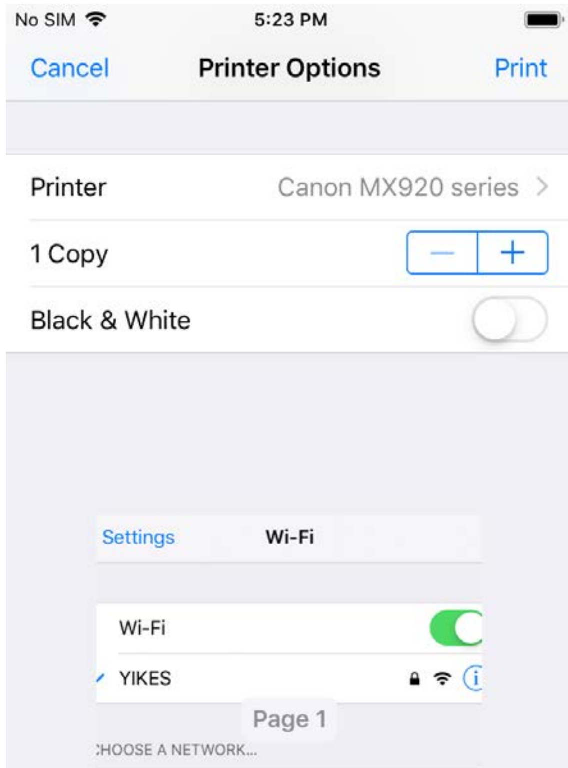
Table 3-17: Exercise YnMUD-4-v4

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

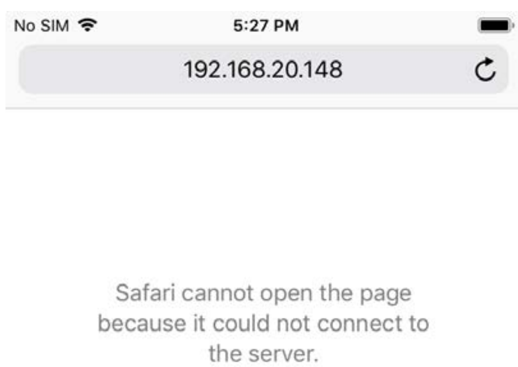
Exercise Field	Description
Associated Cybersecurity Framework Subcategory(ies)	ID.AM-3, ID.AM-4, PR.AC-1, PR.AC-3, PR.AC-4, PR.AC-5
IoT Device(s) Used	Same as for exercise YnMUD-1-v4
Policy Used	In the Yikes! UI: - The Cell Phone local rules are set to allow cell phones to initiate communications to 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 smart 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**

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>

Procedure 2:



Procedure 3:



</td></tr>
</table>

Exercise Field	Description
	Procedure 4: Operating System/Linux OS/Gentoo Linux MY-CONTROLLER-PI - B8:27:EB:2B:39:B1 RASPBERRY PI FOUNDATION : GENTOO LINUX PRINTERS
	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]`

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'`

Exercise Field	Description
	<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]** **Procedure 2:** `--2019-07-24 10:50:53--` http://www.google.com/

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

Exercise Field	Description
	$ 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'

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

<table>
<tr><th>Exercise Field</th><th>Description</th></tr>
<tr><td></td><td>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.</td></tr>
<tr><td>Demonstrated Results</td><td>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<pre>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</pre></td></tr>
</table>

Exercise Field	Description
	`NCCoEs-MBP:results nccoe$` **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'` `        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`

469 As explained above, exercise YnMUD-6-v6 is identical to exercise YnMUD-6-v4 except that it uses IPv6
470 instead of IPv4.

471 ### 3.2.4.7 Exercise YnMUD-7-v4

472 **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	1. 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.

Exercise Field	Description
	2. 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. 3. Let 24 hours elapse. 4. 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.
Demonstrated Results	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:** `# Q9THREATRULES start` `#` `# DO NOT EDIT THESE LINES. Q9THRT WILL REPLACE WITH ITS CONFIGURATION` `#` `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`

Exercise Field	Description
	`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` **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`

Exercise Field	Description
	`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` `option family ipv4` `option target ACCEPT` **[Omitted for brevity]** **`# 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` `#` **[Omitted for brevity]** `# OSMUD end` `# AYIKES start` `#` `# DO NOT EDIT THESE LINES. AYIKES WILL REPLACE WITH ITS CON-FIGURATION` `#` `# Begin YIKES ipset firewall declarations` **[Omitted for brevity]** `# AYIKES end`

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 is still under development by CableLabs. Therefore, it has not yet been fully demonstrated. Documentation of Build 3's functional evaluation and demonstration is planned for inclusion in the next phase of this project.

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 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-11-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
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
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

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
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
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		IoT-2-v4

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, 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.	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 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**		IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		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-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
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	IoT-3-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			handles whether to allow or block traffic to and from the MUD-enabled IoT device.	
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
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		IoT-4-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		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.**	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 **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
CR-6.a		**The MUD manager shall install a router**		IoT-1-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		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.**	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
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**		IoT-5-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
		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.	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
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

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
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
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

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
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 to initiate lateral traffic to approved devices.**		IoT-6-v4
CR-9.a.1			**The router or switch shall receive the at-**	IoT-6-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			tempt 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.**		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
CR-10.a		The MUD-enabled IoT device shall **attempt to initiate lateral traffic to unapproved** (implicitly denied) **devices.**		IoT-6-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
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 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.**			No test needed because the DHCP server does not forward the MUD URL to the MUD manager, as intended.

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
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
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**	N/A

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			policies associated with the affected IoT device that had been configured on the MUD PEP router/switch.	
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
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**	IoT-10-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
			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.	IoT-10-v4
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**			IoT-9-v4

Capability Requirement (CR)-ID	Parent Requirement	Subrequirement 1	Subrequirement 2	Test Case
	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. (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.

Test Case Field	Description
	(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. (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.

Test Case Field	Description
	(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.
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

Test Case Field	Description
	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	**Flow rules on router/switch:** As seen below, tables zero and one classify the packets based on source and destination address, and tables two and three implement the MUD 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.

Test Case Field	Description
	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 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

Test Case Field	Description
	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=0x4000000000000000/0x4000000000000000 actions=goto_table:5 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=CON-

Test Case Field	Description
	TROL-LER: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=CONTROL-LER: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=CONTROL-LER: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 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

Test Case Field	Description
	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_src=8080 actions=write_metadata:0xffffffffffffffff/0,goto_table:4

Test Case Field	Description
	cookie=0xde7f1, duration=40.921s, table=3, n_packets=0, n_bytes=0, priority=40,udp,metadata=0x4000/0xfff000,nw_src=10.0.41.225,tp_src=4000 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0x6d587, duration=40.896s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_dst=80 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0x6d587, duration=40.799s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_dst=888 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0x6d587, duration=40.852s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_src=80 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0x6d587, duration=40.825s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x100000000004000/0x100000000fff000,tp_src=888 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0xd0bd1, duration=40.729s, table=3, n_packets=0, n_bytes=0, priority=40,tcp,metadata=0x400004000/0xfff00fff000,tp_src=800 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0xecf6, duration=40.565s, table=3, n_packets=0, n_bytes=0, priority=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, priority=40,tcp,metadata=0x400004000/0xfff00fff000,tp_dst=800 actions=write_metadata:0xffffffffffffffff/0,goto_table:4 cookie=0xecf6, duration=40.543s, table=3, n_packets=0, n_bytes=0, priority=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 actions=write_metadata:0xffffffffffffffff/0,goto_table:5 cookie=0x29a94, duration=80.685s, table=3, n_packets=282, n_bytes=22446, priority=30 actions=write_metadata:0x29a94,goto_table:4 cookie=0x64f19, duration=79.686s, table=4, n_packets=281, n_bytes=24670, priority=41 actions=NORMAL,IN_PORT

Test Case Field	Description
	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": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=888,dstPort=-1,targetTable=3)", "packet-count": 0

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td><pre>
          },
          {
              "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
          },
          {
</pre></td></tr>
</table>

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": "MetadataProtocolAndSrcDstPortMatchGoToTable(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": "MetadataProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=800,targetTable=3)", "packet-count": 0 }, { "flow-id": "https://sensor.nist.local/nist-mud1/mud-31931-v4fr/cl0-frdev",

Test Case Field	Description
	"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*", "byte-count": 0, "table-id": 2,

Test Case Field	Description
	"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, "dst-model": "*https://sensor.nist.local/nist-mud1*",

Test Case Field	Description
	"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, "flow-name": "MetadaPro-tocolAndSrcDstPortMatchGoToTable(proto-col=6,srcPort=888,dstPort=-1,targetTable=4)", "packet-count": 0

Test Case Field	Description
	}, { "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)", "packet-count": 0 }, {

Test Case Field	Description
	"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 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/loc0-todev/2*",

Test Case Field	Description
	"src-local-networks-flag": true, "byte-count": 0, "table-id": 3, "dst-model": "*https://sensor.nist.local/nist-mud1*", "priority": 40, "flow-name": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=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": "MetadaProtocolAndSrcDstPortMatchGoToTable(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": "MetadaProtocolAndSrcDstPortMatchGoToTable(protocol=6,srcPort=-1,dstPort=80,targetTable=4)", "packet-count": 0 }, { "flow-id": "*https://sensor.nist.local/nist-mud1/mud-31931-v4to/cl0-todev/TCP_DIRECTION_CHECK*", "byte-count": 0,

Test Case Field	Description
	"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

512 IPv6 is not supported in this implementation.

513 *5.1.2.2 Test Case IoT-2-v4*

514 **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.

Test Case Field	Description
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. 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]).

Test Case Field	Description
	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).
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` `{`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>

<pre>
    "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://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]
       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-
</pre>
</td></tr>
</table>

Test Case Field	Description
	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

515 IPv6 is not supported in this implementation.

516 *5.1.2.3 Test Case IoT-3-v4*

517 **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,

Test Case Field	Description
	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. 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.

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. 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.
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": {`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
        "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://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]
        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-
</pre>
</td></tr>
</table>

Test Case Field	Description
	`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

518 IPv6 is not supported in this implementation.

519 5.1.2.4 Test Case IoT-4-v4

520 **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

Test Case Field	Description
	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 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.

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. 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 allowed (DHCP, DNS, NTP)—this is the standard policy on MUD file verification failures.
Actual Results	**IoT device before DHCP request:** `python get-src-mac-metadata.py -m 00:13:EF:20:1D:6B` `{` `    "input": {` `        "mac-address": "00:13:EF:20:1D:6B"`

Test Case Field	Description
	<code> } } { "output": { "src-local-networks-flag": true, "src-quarantine-flag": false, **"src-blocked-flag": false,** "src-model": "UNCLASSIFIED", "src-manufacturer": "UNCLASSIFIED", "metadata": "100300300000000" } }</code> **<u>MUD manager logs—exception when there is an issue with MUD file</u>:** **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_nist-mud_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_nist-mud_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.execchain.MainClientExec.establishRoute(MainClientExec.java:381)[379:wrap_file__home_mudmanager_nist-mud_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.execchain.MainClientExec.execute(MainClientExec.java:237)[379:wrap_file__home_mudmanager_nist-mud_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.execchain.ProtocolExec.execute(ProtocolExec.java:185)[379:wrap_file__home_mudmanager_nist-mud_sdnmud-aggregator_karaf_target_assembly_system_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.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>

IPv6 is not supported in this implementation.

5.1.2.5 Test Case IoT-5-v4

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).

Test Case Field	Description
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.
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

Test Case Field	Description
	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)
Procedure	Note: Procedure steps with strikethrough are not tested due to NAT. 1. As stipulated in the preconditions, right before this test, test IoT-1-v4 must have been run successfully.

Test Case Field	Description
	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.` `HTTP request sent, awaiting response... 200 OK` `Length: 116855 (114K) [text/html]` `Saving to: ‘index.html.51’`

Test Case Field	Description
	`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.`
Overall Results	Pass

524 IPv6 is not supported in this implementation.

525 ##### 5.1.2.6 Test Case IoT-6-v4

526 **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 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

Test Case Field	Description
	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 the MUD URL [mudfileserver] of the IoT device in question),** and further constrained by source port: any; destination port: 80; and protocol: TCP.

Test Case Field	Description
	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 this traffic is received at the MUD PEP, but it **is not forwarded** by the MUD PEP, nor is it received at *anyhost-from*.

Test Case Field	Description
	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 (in-gress): **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 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*.

Test Case Field	Description
Expected Results	Each of the results that is listed as needing to be verified in the procedure steps above occurs as expected.
Actual Results	2. Local-network (ingress)—allowed: `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... con-nected.` `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]`
	3. Local-network (egress)—blocked: `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.`
	4. Local-network, controller, my-controller, manufacturer class (egress)—allowed: Local-network: `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.`

Test Case Field	Description
	`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` `PING 2 03:32:00` `RTT = 0.812637805939` `PING 3 03:32:01` `RTT = 0.652308940887` `PING 4 03:32:02`

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
RTT = 0.784868001938
PING 5 03:32:02
RTT = 0.573136806488
PING 6 03:32:03
RTT = 0.481912136078
[rc=6]
</pre>
<hr>
Manufacturer:
<pre>
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]
</pre>
<hr>
5. Local-network, controller, my-controller, manufacturer class (in-gress)—blocked:
Local-network:
<pre>
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.

--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.
</pre>
</td></tr>
</table>

Test Case Field	Description
	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/` `Resolving sensor (sensor)... 10.0.41.190` `Connecting to sensor (sensor)|10.0.41.190|:800... failed: Connection timed out.` `Retrying.`

Test Case Field	Description
	`--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]`
	9. Same-manufacturer class (egress)—blocked: `sensor ]` **`ping sameman -c 10`** `PING sameman (10.0.41.220) 56(84) bytes of data.`

Test Case Field	Description
	`--- sameman ping statistics ---` `10 packets transmitted, 0 received, 100% packet loss, time 9383ms`
Overall Results	Pass

527 IPv6 is not supported in this implementation.

528 ##### *5.1.2.7 Test Case IoT-9-v4*

529 **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 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

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	*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. 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*.

Test Case Field	Description
	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` `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 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	*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. 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.

<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.
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 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.</td></tr>
<tr><td>Actual Results</td><td><u>IoT device initial DHCP event</u>:

<code>For the first DHCLient request:
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.</code></td></tr>
</table>

Test Case Field	Description
	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-time":1567537376711,"mud-url":"*https://sensor.nist.local/nistmud1*"}]}** 2019-09-03 15:02:56,739 | INFO | on-dispatcher-99 | Datas-toreUpdater | 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*"}]} **IoT device—second DHCP request:**

<table>
<tr><th>Test Case Field</th><th>Description</th></tr>
<tr><td></td><td>
<pre>
 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/

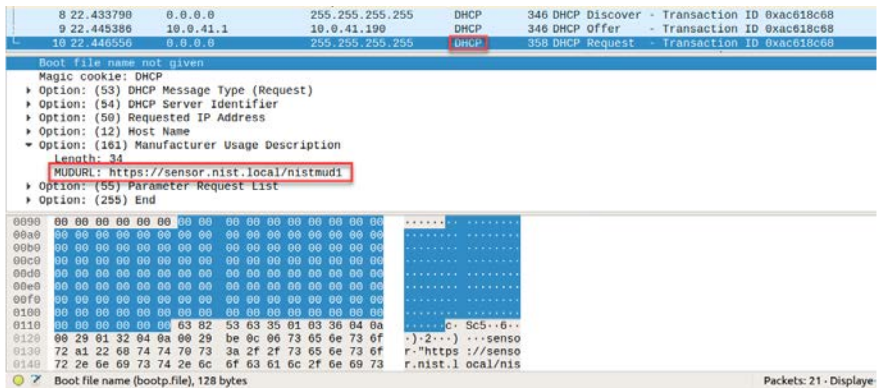
Listening on LPF/wlan0/00:13:ef:20:1d:6b
Sending on   LPF/wlan0/00:13:ef:20:1d:6b
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 8
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 19
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 12
DHCPREQUEST 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.
</pre>
<u>MUD manager—log file showing cached file in use</u>:
<pre>
2019-09-03 15:03:51,666 | INFO | on-dispatcher-99 | Mud-
FileFetcher | 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 | Mud-
FileFetcher | 93 - gov.nist.antd.sdnmud-impl
- 0.1.0 | read 9548 characters
</pre>
<u>MUD file server—log after second fetch (no change in output)</u>:
<pre>
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
</pre>
</td></tr>
<tr><td>Overall Results</td><td>Pass</td></tr>
</table>

533 IPv6 is not supported in this implementation.

534 ### 5.1.2.9 Test Case IoT-11-v4

535 **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
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

Test Case Field	Description
Actual Results	
Overall Results	Pass

536 ## 5.1.3 MUD Files

537 This section contains the MUD files that were used in the Build 4 functional demonstration.

538 ### *5.1.3.1 mudfile-sensor.json*

539 The complete mudfile-sensor.json MUD file has been linked to this document. To access this MUD file
540 please click the link below.

541 mudfile-sensor.json

542 ### *5.1.3.2 mudfile-otherman.json*

543 The complete mudfile-otherman.json MUD file has been linked to this document. To access this MUD
544 file please click the link below.

545 mudfile-otherman.json