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.

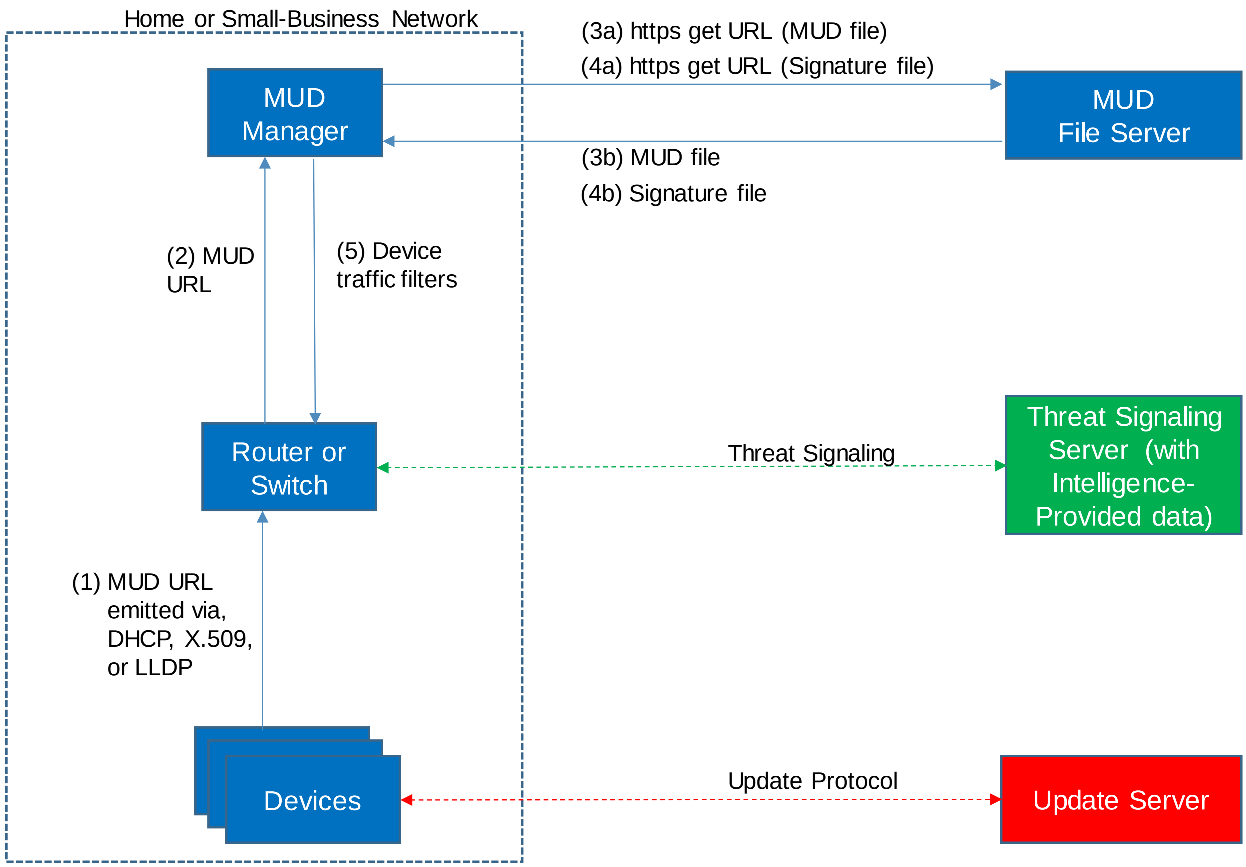
1.2.1 Usage Scenarios

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.

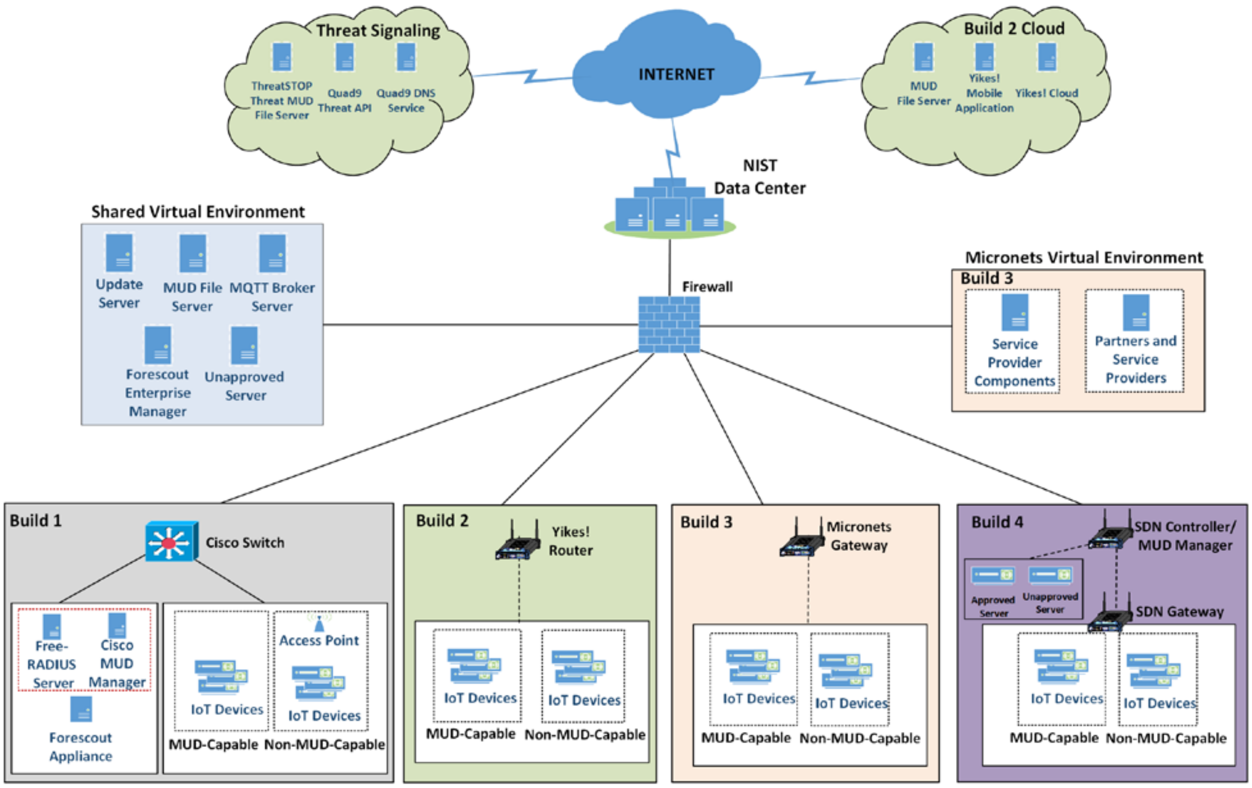
The following four builds, as depicted in the diagram, are supported within the physical architecture:

- 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.

Figure 1-2 NCCoE Physical Architecture



1.3 Typographic Conventions

The following table presents typographic conventions used in this volume.

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

2 Build 1 Product Installation Guides

This section of the practice guide contains detailed instructions for installing and configuring all 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:

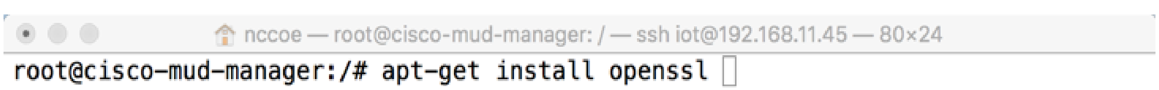
```
sudo su
```

2. Change to the root directory:

```
cd /
```

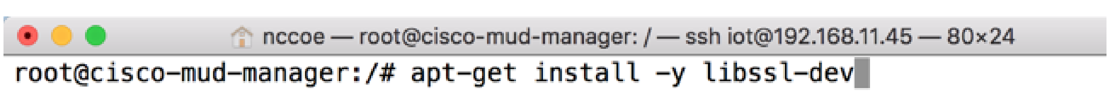
3. To install OpenSSL from the terminal, enter the following command:

```
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:

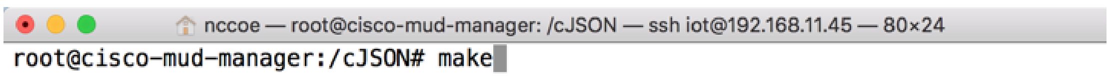
```
git clone https://github.com/DaveGamble/cJSON
```

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

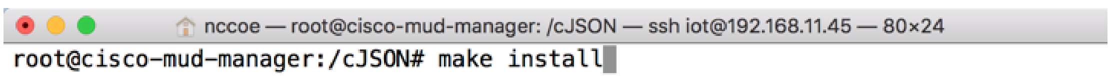
```
cd cJSON
```

   b. Build cJSON by entering the following commands:

```
make
```

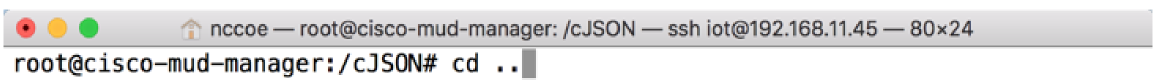



```
make install
```



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

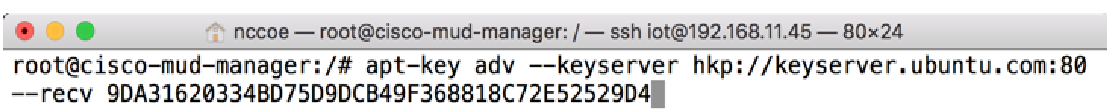
```
cd ..
```



6. To install MongoDB, enter the following commands:

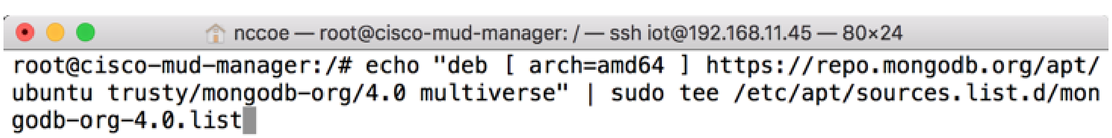
   a. Import the public key:

```
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
```

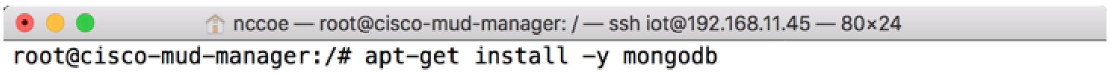


c. Reload the local package database:

```
apt-get update
```

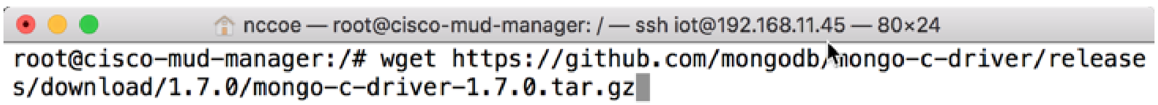
d. Install the MongoDB packages:

```
apt-get install -y mongodb
```



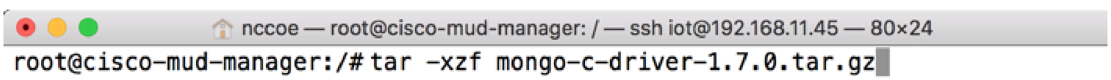
7. To install the Mongo C driver, enter the following command:

```
wget https://github.com/mongodb/mongo-c-driver/releases/download/1.7.0/mongo-c-driver-1.7.0.tar.gz
```



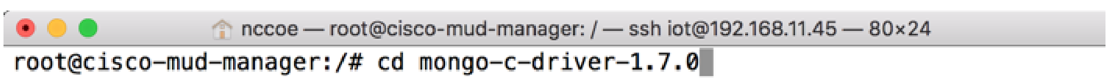
a. Untar the file by entering the following command:

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



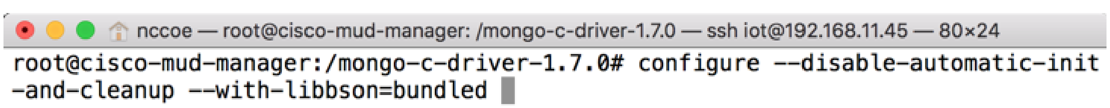
b. Change into the mongo-c-driver-1.7.0 directory by entering the following command:

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

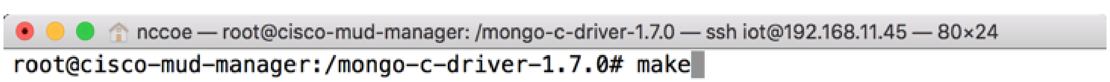


c. Build the Mongo C driver by entering the following commands:

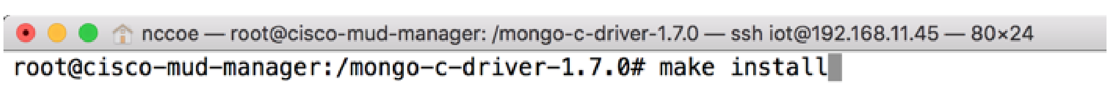
```
./configure --disable-automatic-init-and-cleanup --with-libbson=bundled
```




```
make
```

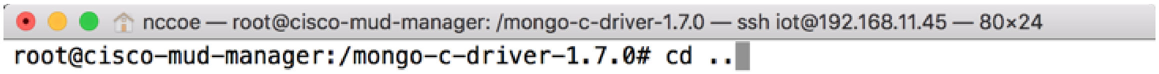



```
make install
```



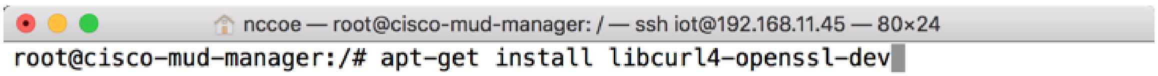
8. Change directories back a folder by entering the following command:

```
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:

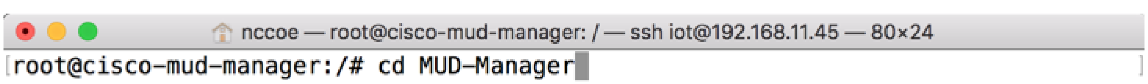
```
cd /
```

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
```

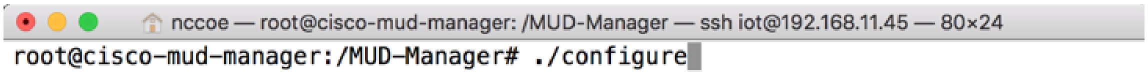
4. Change into the MUD manager directory:

```
cd MUD-Manager
```



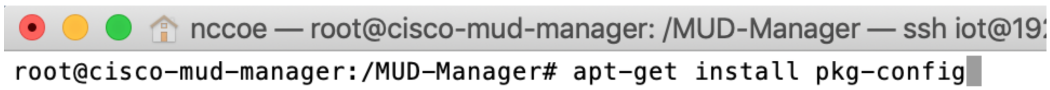
5. Build the MUD manager by entering the following commands:

```
./configure
```

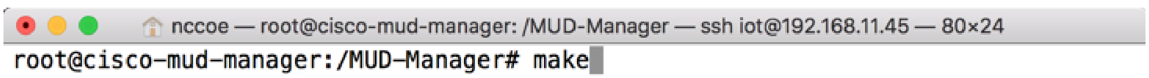


Note: If a “pkg-config error” is thrown, run the command below to install the missing package:

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

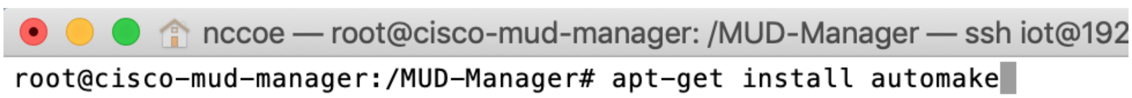



```
make
```

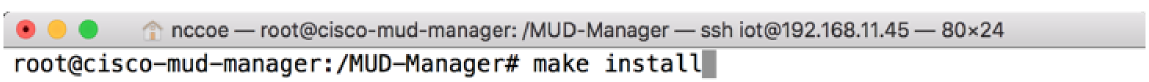


Note: If an “ac.local error” is thrown, run the command below to install the missing package:

```
apt-get install automake
```




```
make install
```



2.1.3.3 MUD Manager Configuration

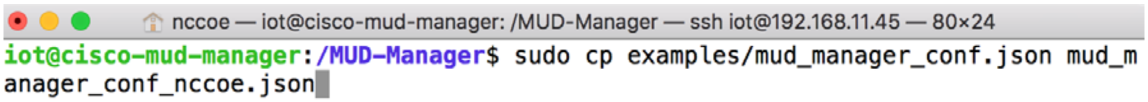
This section describes configuring the MUD manager to communicate with the NCCoE MUD file server and defining the attributes used for translating the fetched MUD files. Details about the configuration file and additional fields that can be set within this file can be accessed here: https://github.com/CiscoDevNet/MUD-Manager#editing-the-configuration-file.

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

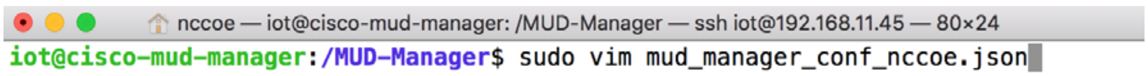
   ```
   cd /MUD-Manager
   ```

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",
```

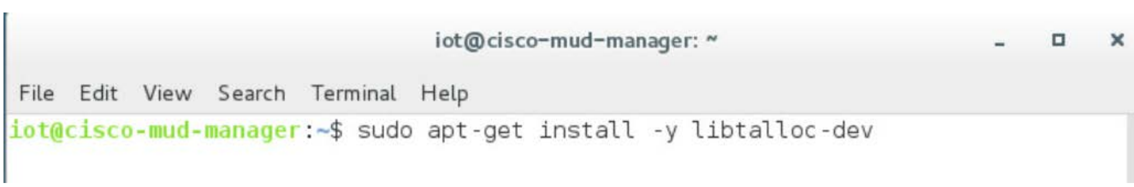
```
                "www.trytechy.com" : "99.84.104.21"
        },

        "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.

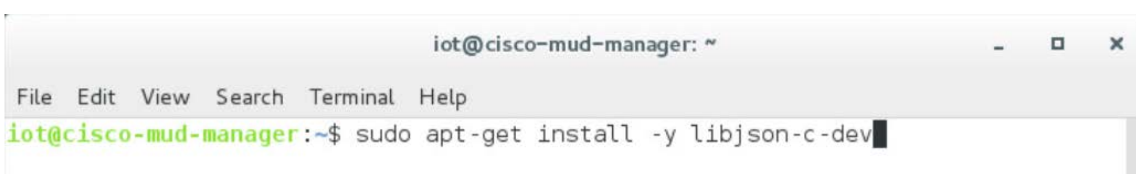
2.1.3.4 FreeRADIUS Installation

1. Install the dependencies for FreeRADIUS:

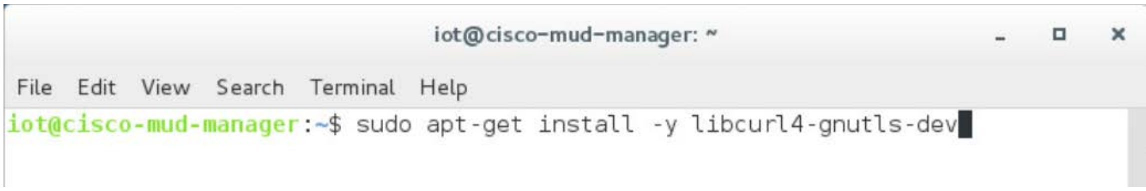
   a. `sudo apt-get install -y libtalloc-dev`



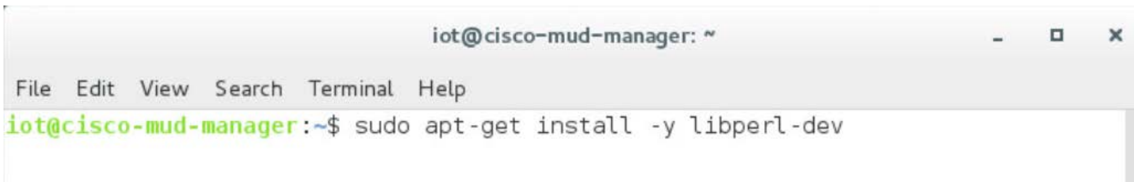
   b. `sudo apt-get install -y libjson-c-dev`



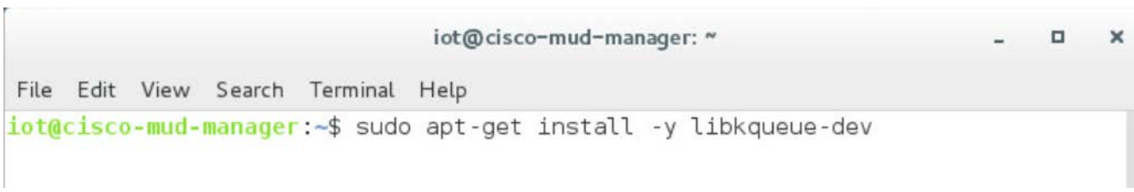
   c. `sudo apt-get install -y libcurl4-gnutls-dev`



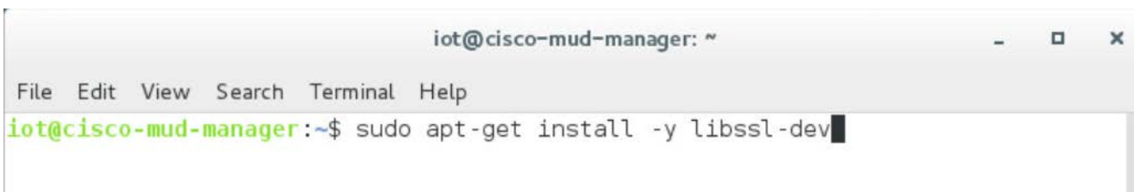
d. `sudo apt-get install -y libperl-dev`



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

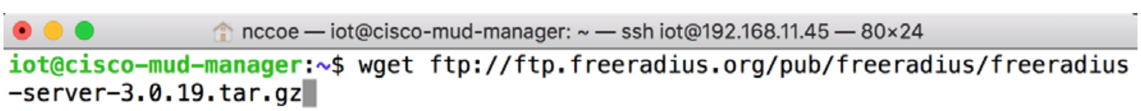


f. `sudo apt-get install -y libssl-dev`



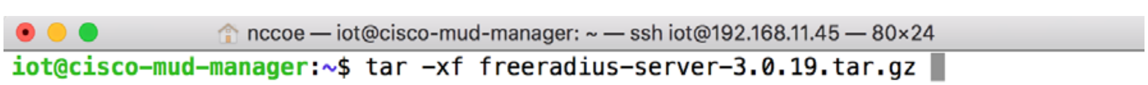
2. Download the source by entering the following command (Note: Version 3.0.19 and later are recommended):

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



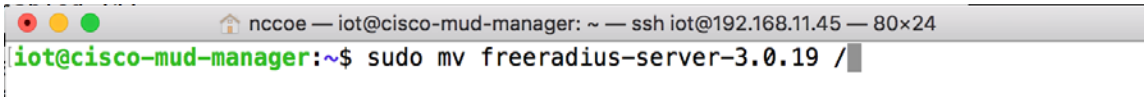
3. Untar the downloaded file by entering the following command:

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



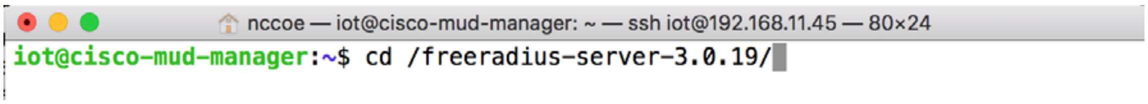
4. Move the FreeRADIUS directory to the root directory:

`sudo mv freeradius-server-3.0.19/ /`



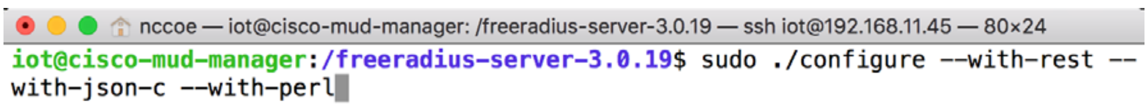
5. Change to the FreeRADIUS directory:

```
cd /freeradius-server-3.0.19/
```

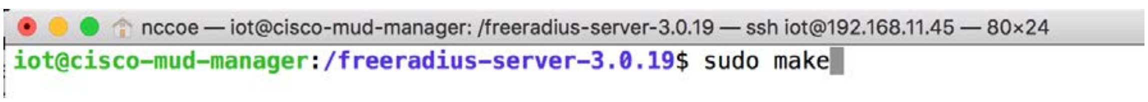


6. Make and install the source by entering the following:

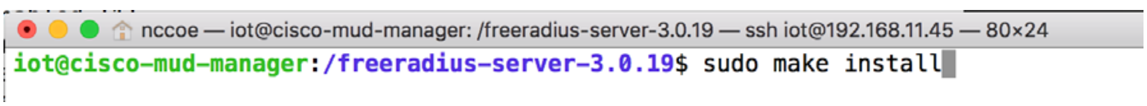
   a. `sudo ./configure --with-rest --with-json-c --with-perl`



   b. `sudo make`



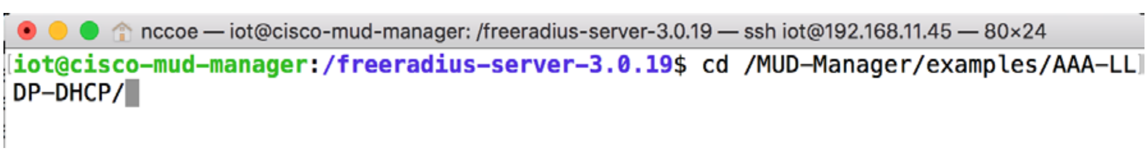
   c. `sudo make install`



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:

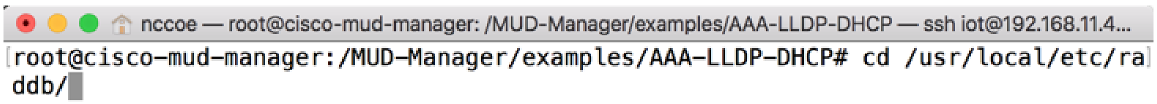
```
sudo ./FR-setup.sh
```

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

```
sudo su
```

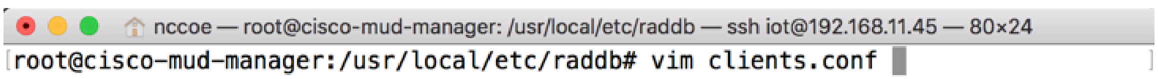
4. Change to the radius directory:

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



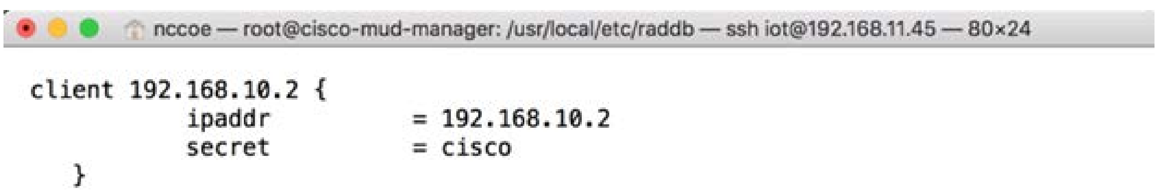
5. Open the *clients.conf* file:

```
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
   }
```

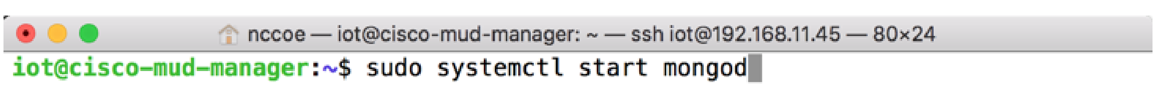


7. Save and close the file.

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
```

```
nccoe — iot@cisco-mud-manager: ~ — ssh iot@192.168.11.45 — 80×24
iot@cisco-mud-manager:~$ 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
   ```

   ```
   nccoe — iot@cisco-mud-manager: ~ — ssh iot@192.168.11.45 — 80×24
   iot@cisco-mud-manager:~$ sudo mud_manager -f /MUD-Manager/mud_manager_conf_nccoe
   .json -l 3
   ```

   The following output should appear if the service started successfully:

   ```
   nccoe — iot@cisco-mud-manager: ~ — ssh iot@192.168.11.45 — 80×24
   iot@cisco-mud-manager:~$ sudo mud_manager -f /MUD-Manager/mud_manager_conf_nccoe
   .json -l 3
   ***MUDC [INFO][main:2939]--> Using configuration file: /MUD-Manager/mud_manager_
   conf_nccoe.json

   ***MUDC [INFO][read_mudmgr_config:322]--> Successfully read Manufacture 0 cert
   ***MUDC [INFO][read_mudmgr_config:353]--> Successfully read Manufacture web 0 ce
   rt
   ***MUDC [INFO][read_mudmgr_config:322]--> Successfully read Manufacture 1 cert
   ***MUDC [INFO][read_mudmgr_config:353]--> Successfully read Manufacture web 1 ce
   rt
   ***MUDC [INFO][read_mudmgr_config:383]--> Certificate read ok:  Continue reading
    domain list
   ***MUDC [INFO][read_mudmgr_config:389]--> JSON is read succesfully
   ***MUDC [INFO][read_mudmgr_config:402]--> JSON is read succesfully
   ***MUDC [INFO][main:2992]--> Starting RESTful server on port 8000
   ```

3. Start the FreeRADIUS service in the foreground with logging enabled by entering the following command:

   ```
   sudo radiusd -Xxx
   ```

   ```
   nccoe — iot@cisco-mud-manager: ~ — ssh iot@192.168.11.45 — 80×24
   iot@cisco-mud-manager:~$ 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

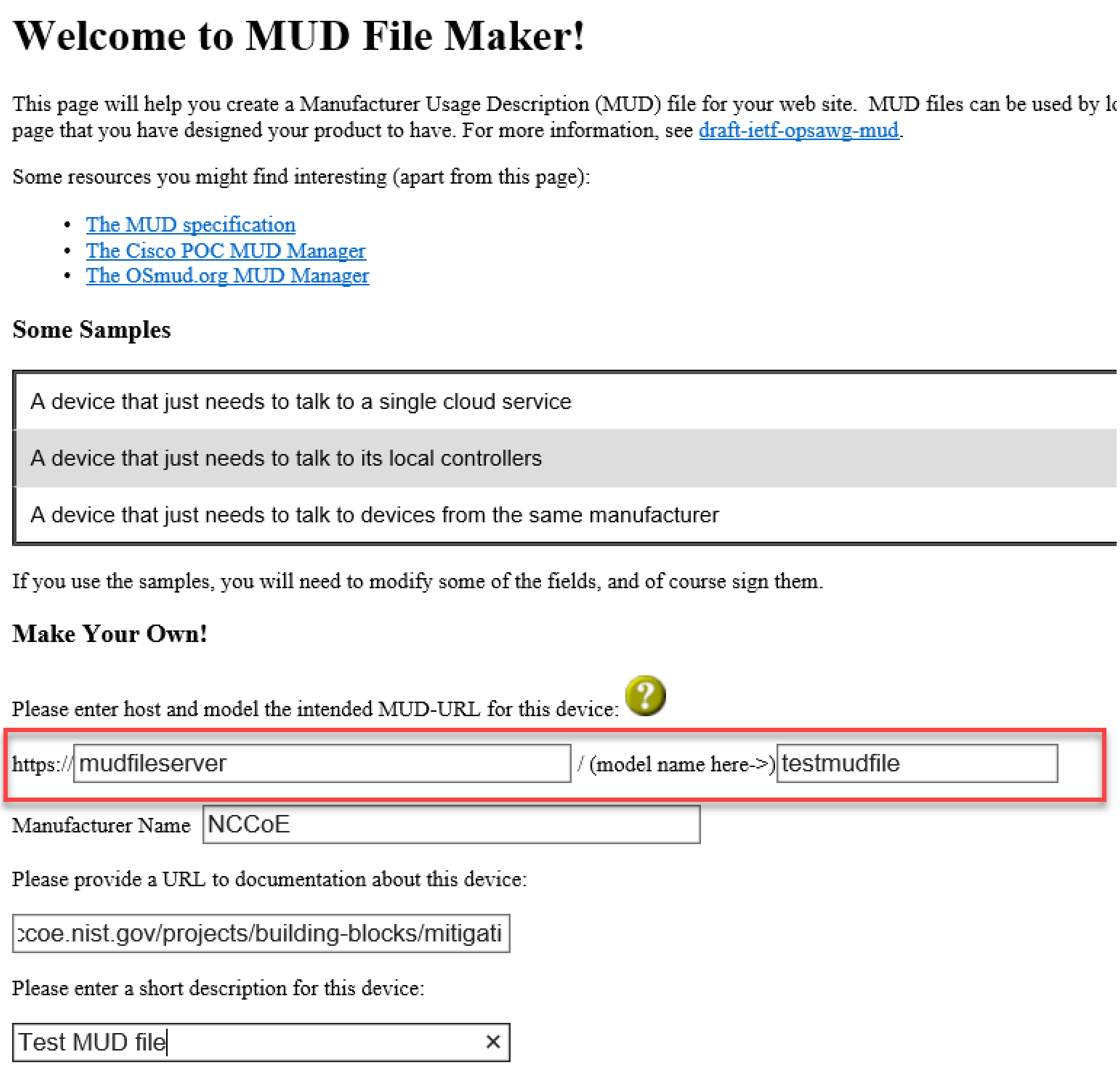
This section details creating and signing a MUD file on the MUD file server. The MUD specification does not mandate that this signing process be performed on the MUD file server itself.

2.2.3.2.1 MUD File Creation

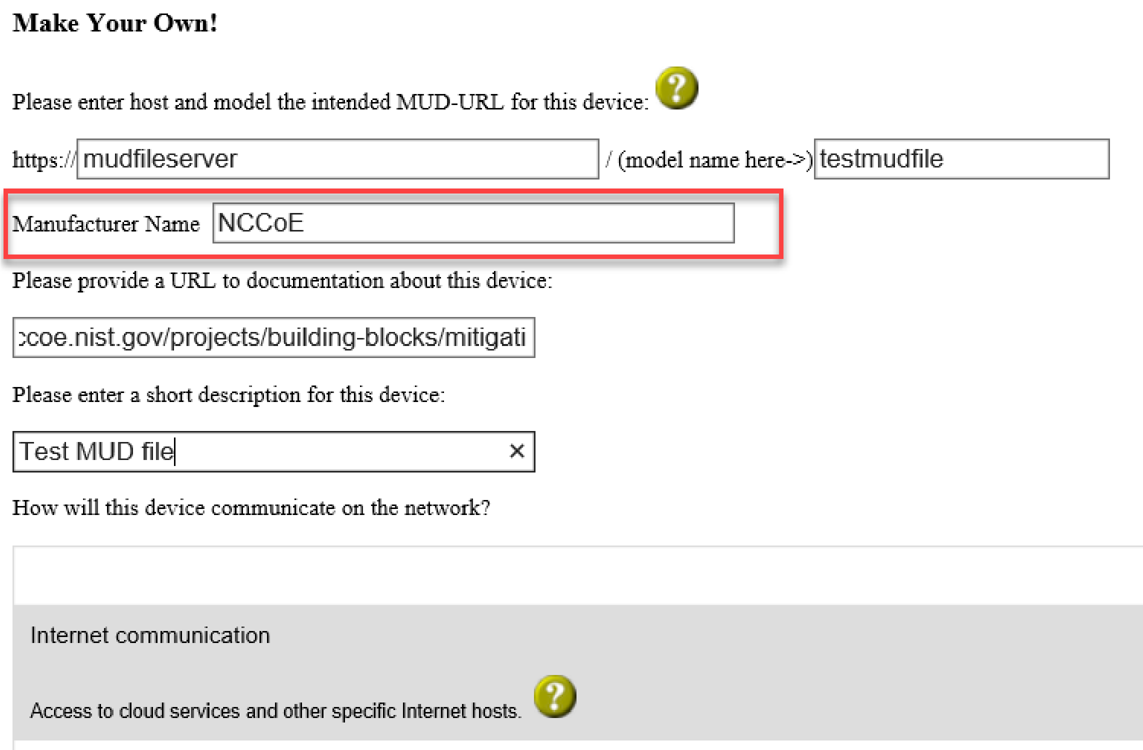
An online tool called MUD Maker was used to build MUD files. Once the permitted communications have been defined for the IoT device, proceed to www.mudmaker.org to leverage the online tool. There is also a list of sample MUD files on the site, which can be used as a reference. Upon navigating to www.mudmaker.org,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

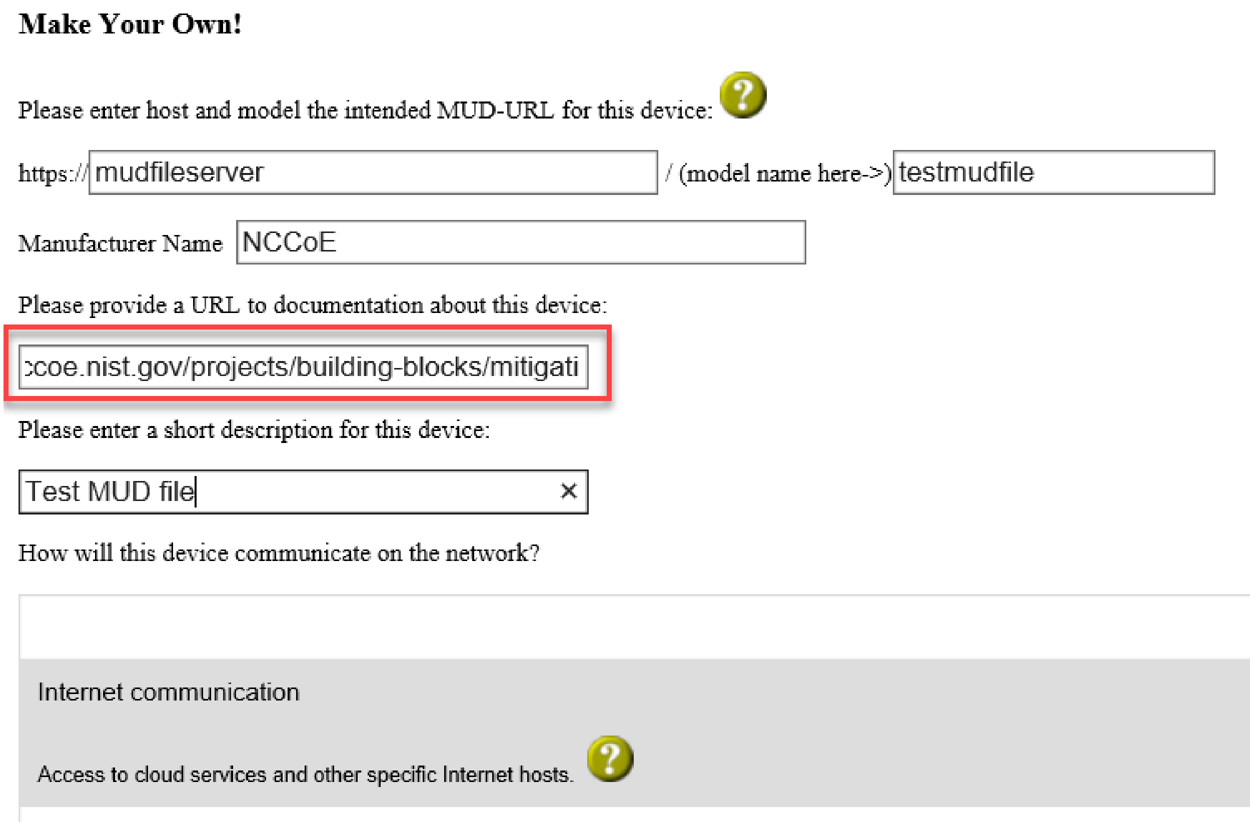
   Sample input: mudfileserver, testmudfile



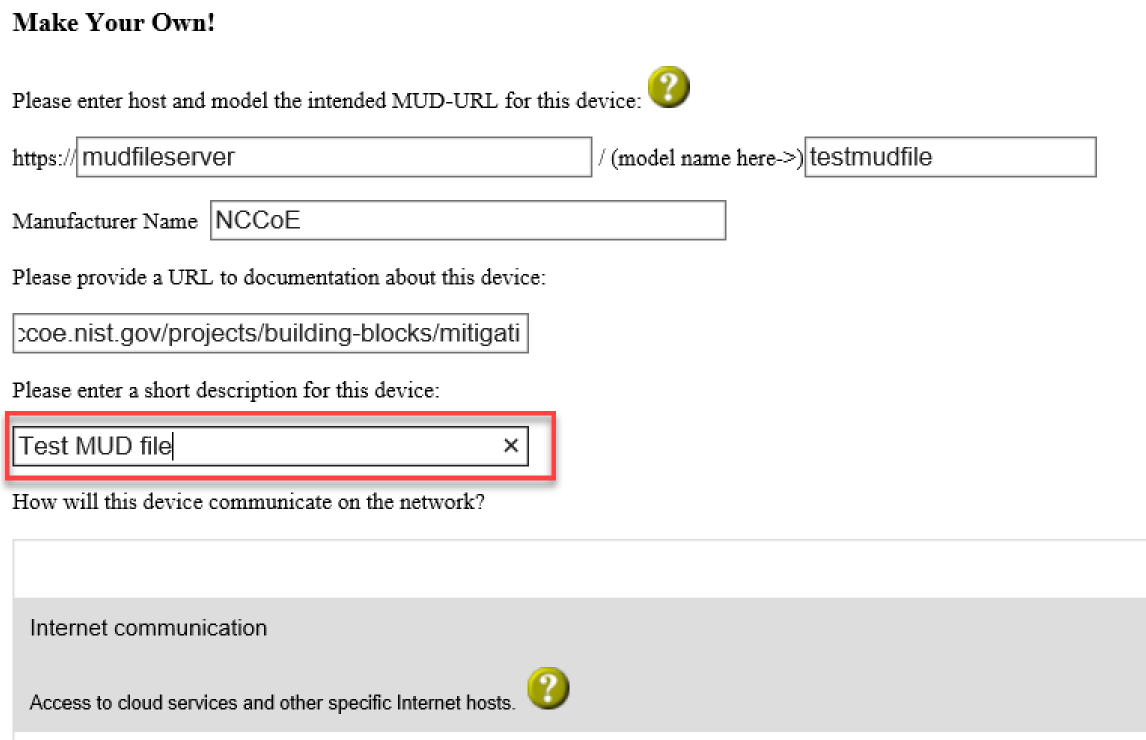
2. Specify the Manufacturer Name of the device in the appropriate input field, which is outlined in red in the screenshot below:



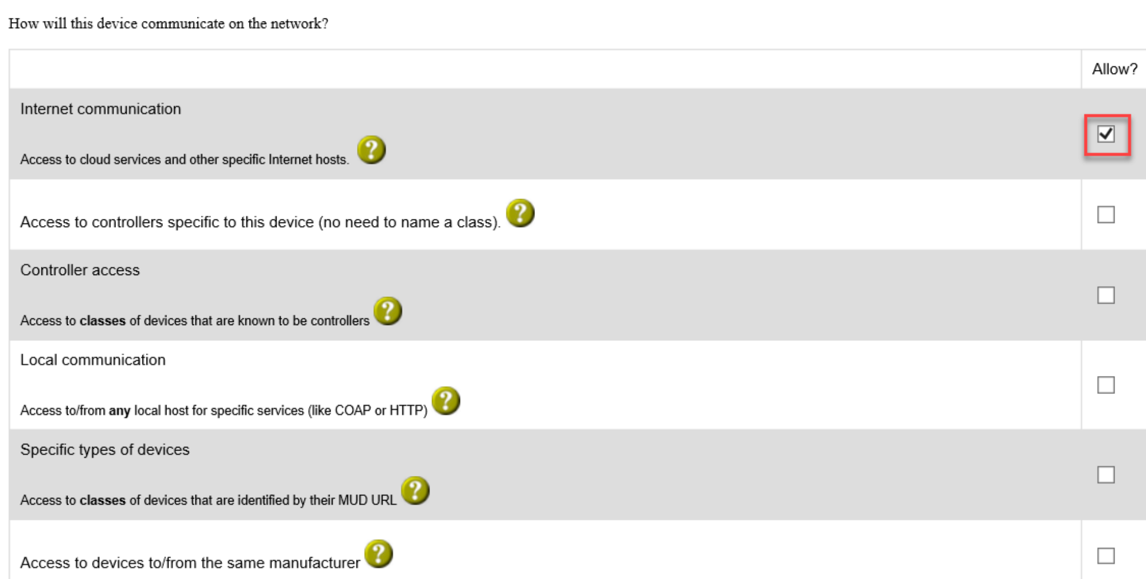
3. Include a URL to provide documentation about this device in the appropriate input field, which is outlined in red in the screenshot below:



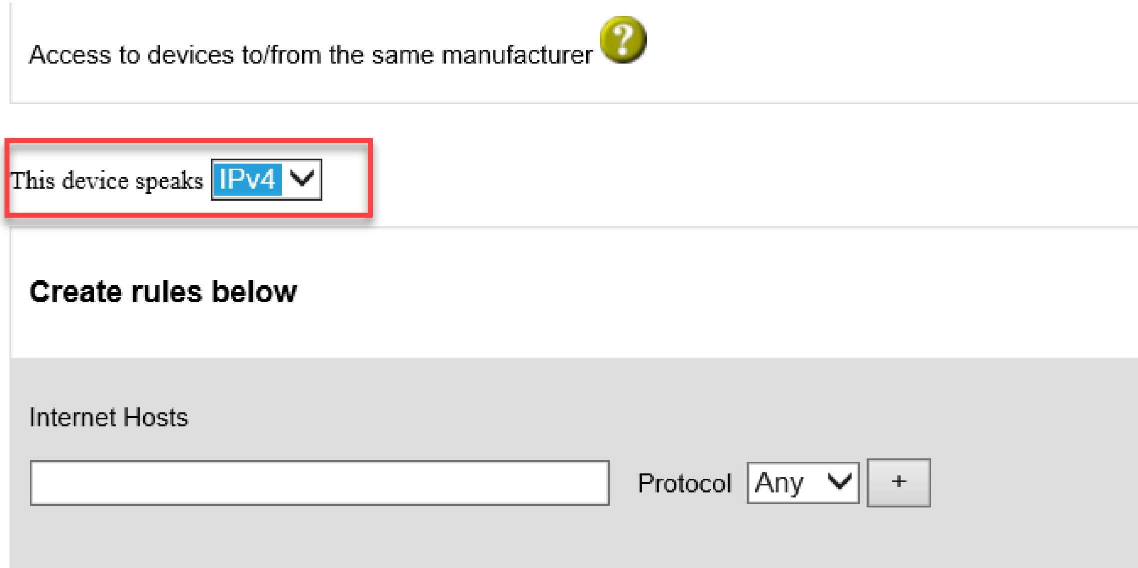
4. Include a short description of the device in the appropriate input field, which is outlined in red in the screenshot below:



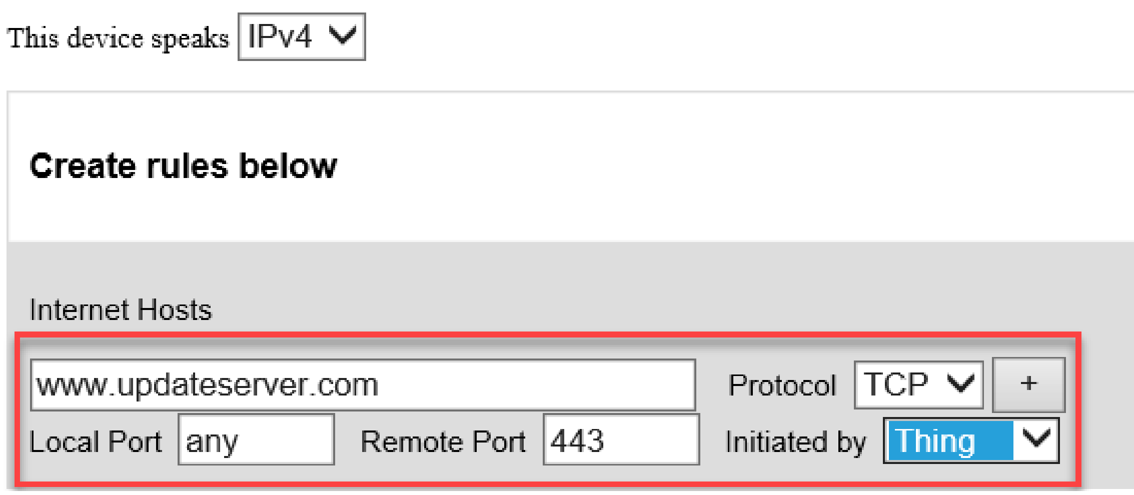
5. Check the boxes for the types of network communication that are allowed for the device:



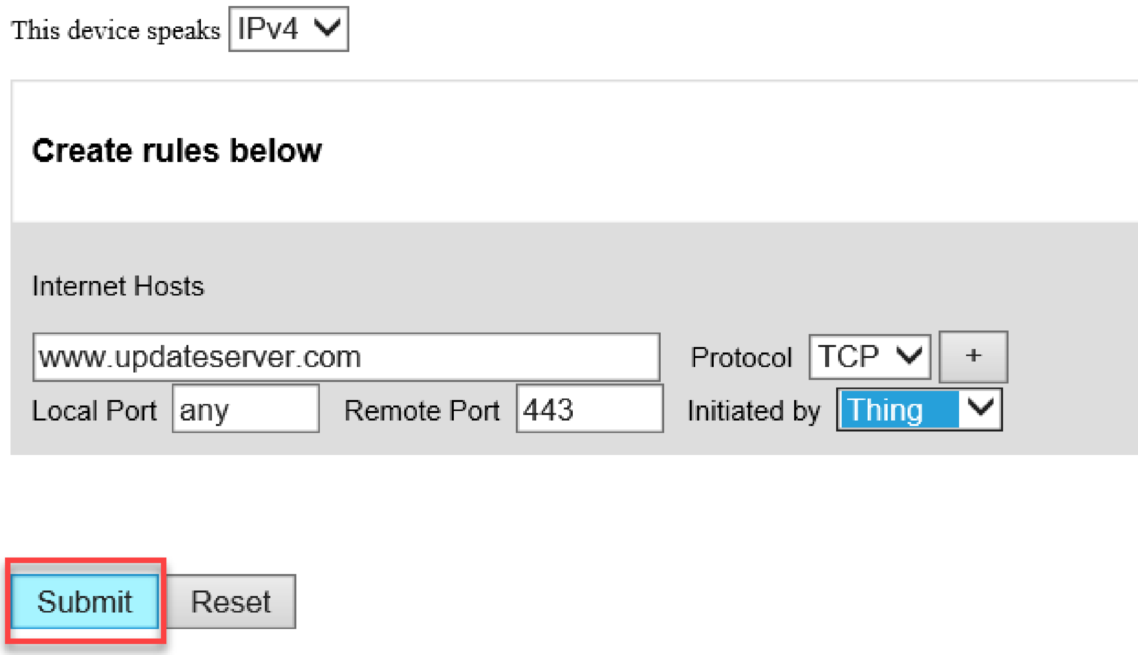
6. Specify the internet protocol version that the device leverages:



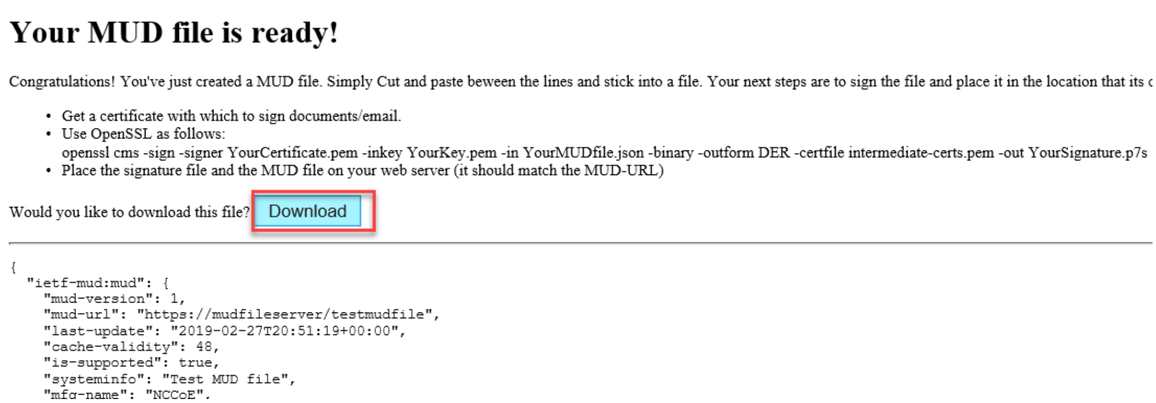
7. Specify values for the fields (Internet Hosts, Protocol, Local Port, Remote Port, and Initiated by) that describe the communications that will be permitted for the device:



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:



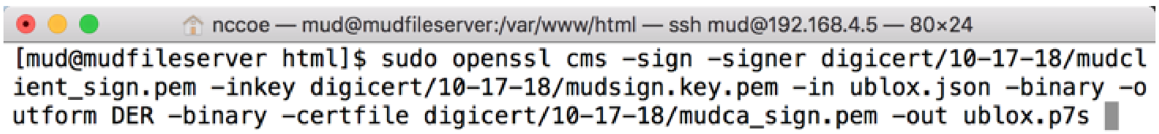
10. Click **Save** to store a copy of the MUD file:

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:

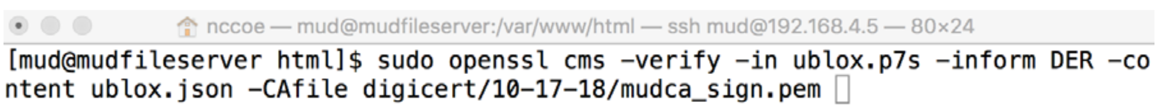
```
sudo openssl cms -sign -signer <Signing Certificate> -inkey <Private Key for
Signing Certificate> -in <Name of MUD File> -binary -outform DER -binary -
certfile <Intermediate Certificate for Signing Certificate> -out <Name of MUD
File without the .json file extension>.p7s
```



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.

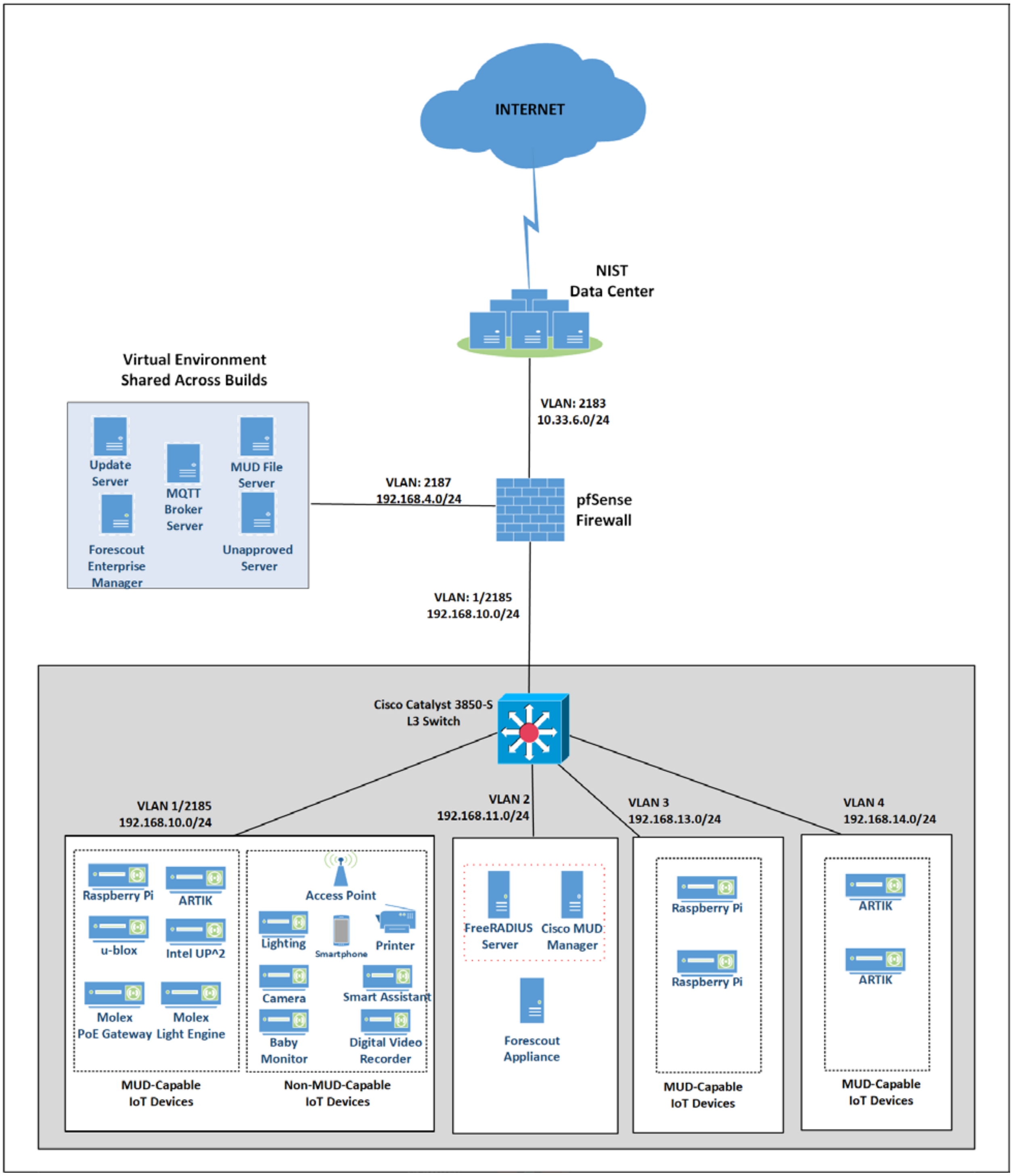
2.3.2 Configuration Overview

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).

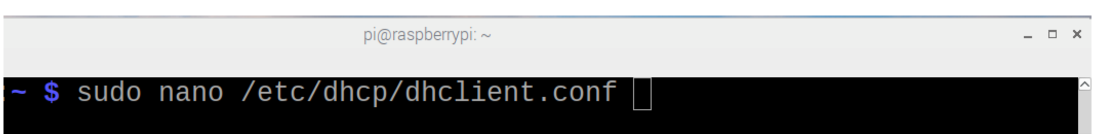
2.5.2.2.1 DHCP Client Configuration

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:

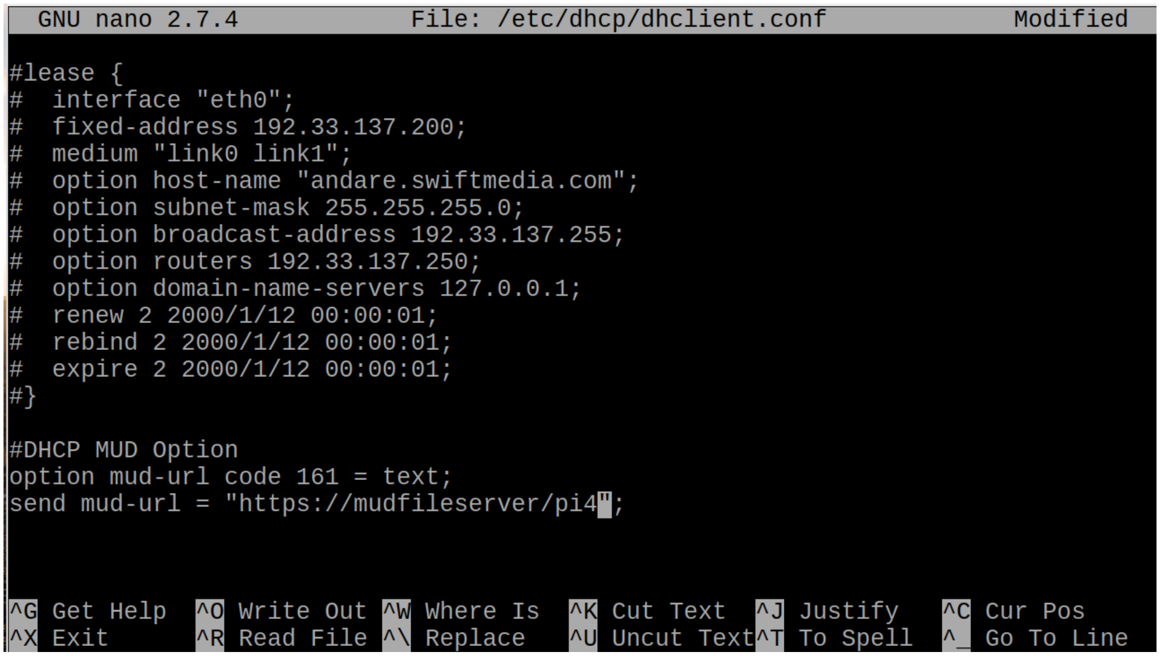
1. Open a terminal on the device.
2. Ensure that any other conflicting DHCP clients are disabled or removed.
3. Install the dhclient package (if needed).
4. Edit the *dhclient.conf* file by entering the following command:

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



5. Add the following lines:

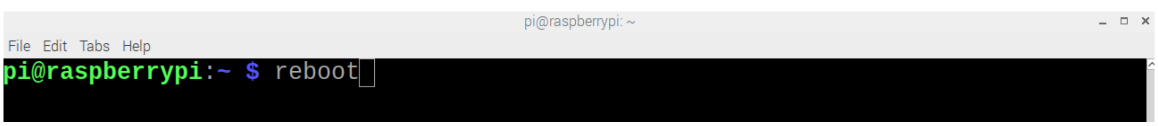
```
option mud-url code 161 = text;
send mud-url = "<insert URL for MUD File here>";
```



6. Save and close the file.

7. Reboot the device:

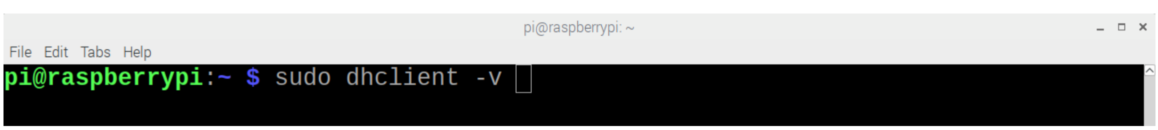
```
reboot
```



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.

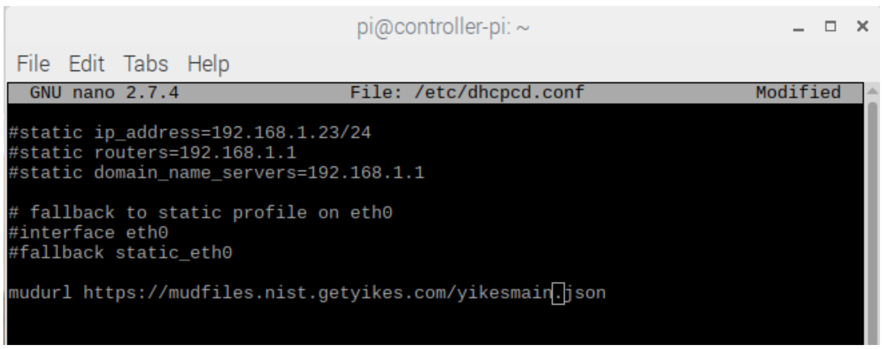
3.6.1.2 Setup

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."

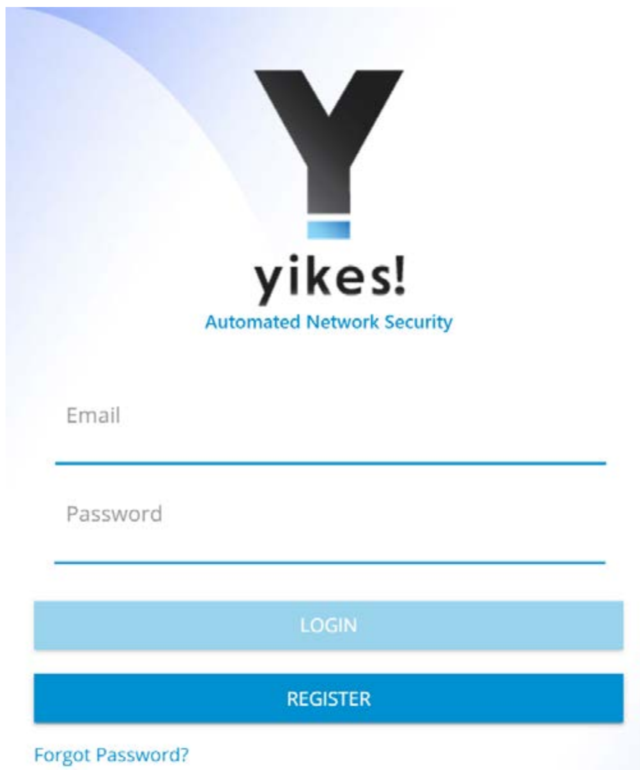
3.9.3 Setup

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.

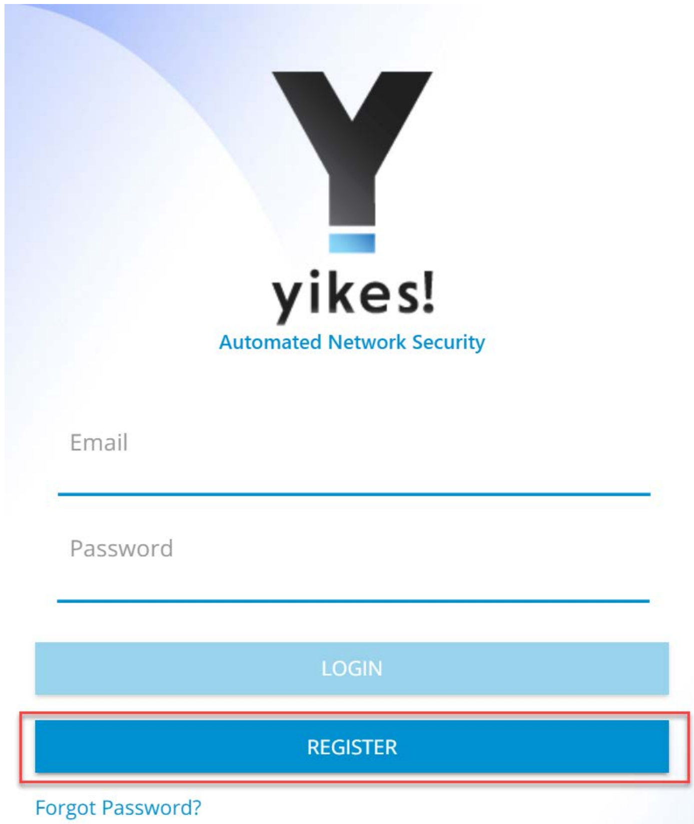
3.9.3.1 Yikes! Router and Account Cloud Registration

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.

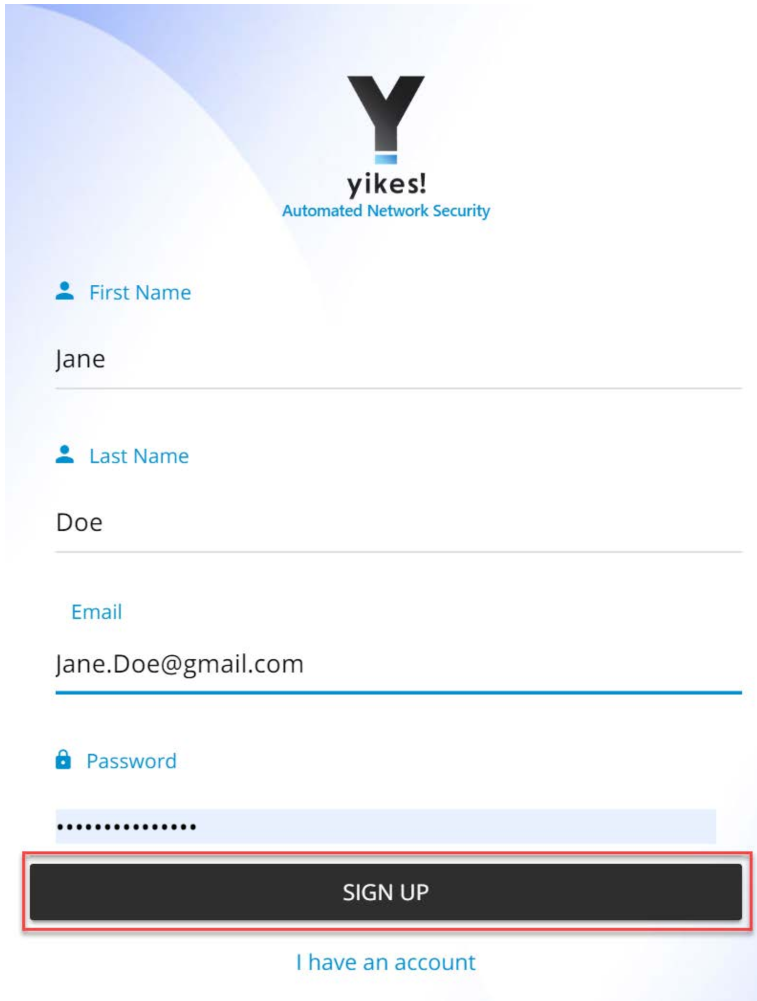
1. Open a browser and access the Yikes! UI. (In the preproduction version of the router, accessing the UI required inputting a URL provided by MasterPeace.):



1466 2. Click on the **Register** button to sign up for an account:

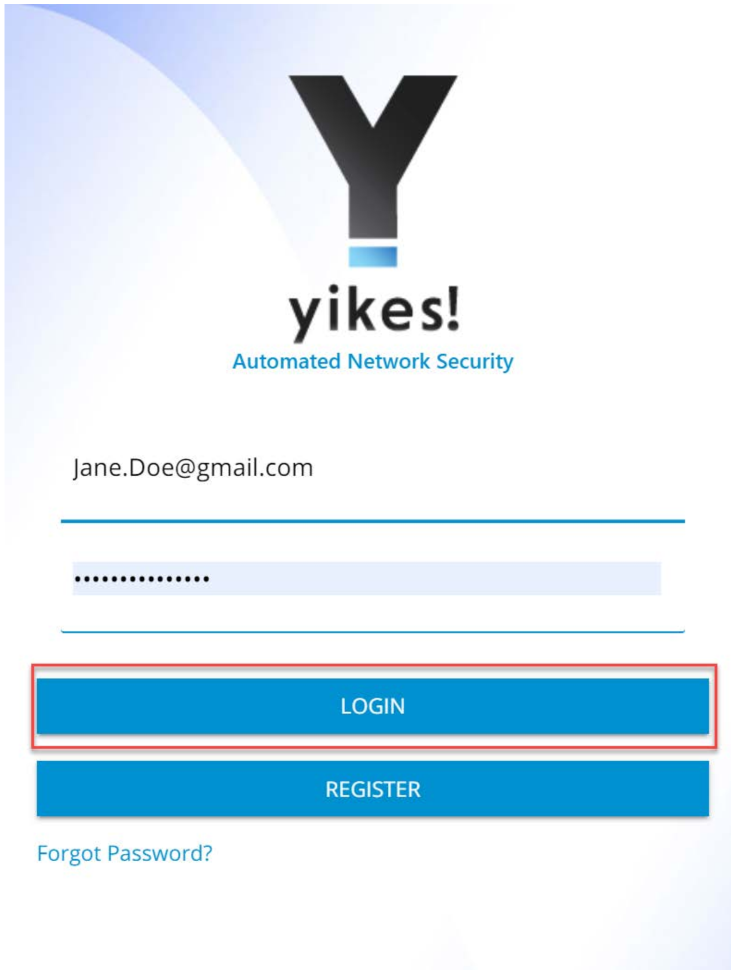
1467 

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

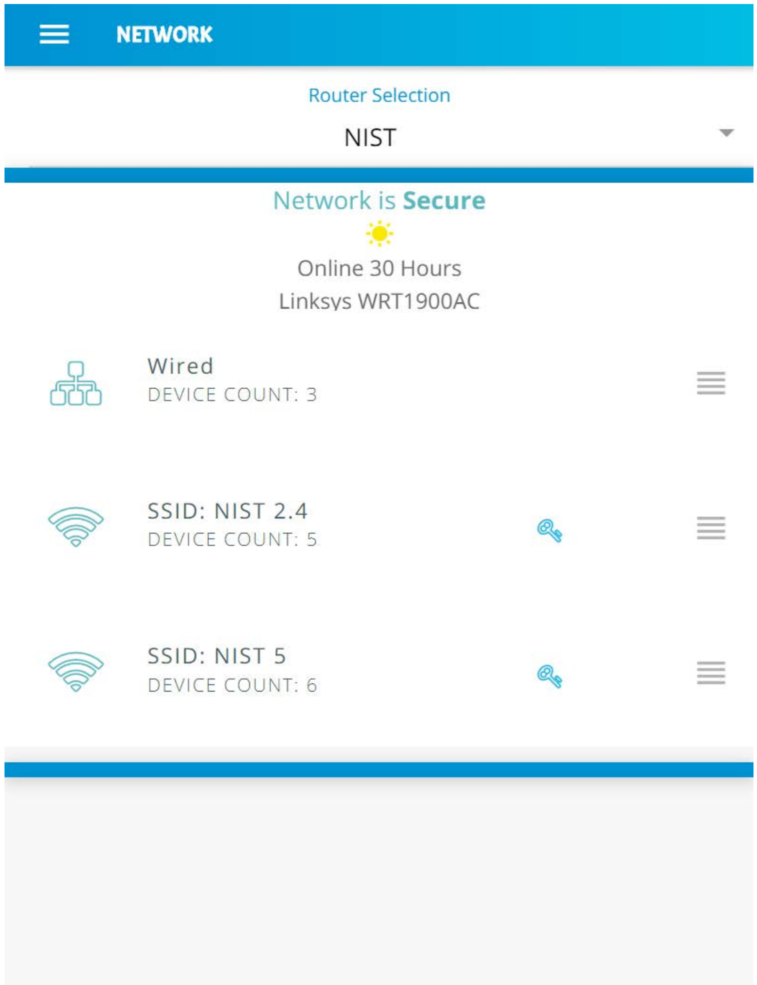


Note: There will be additional steps related to associating the Yikes! router with the Yikes! account being created. However, at this implementation, this process was still under development.

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



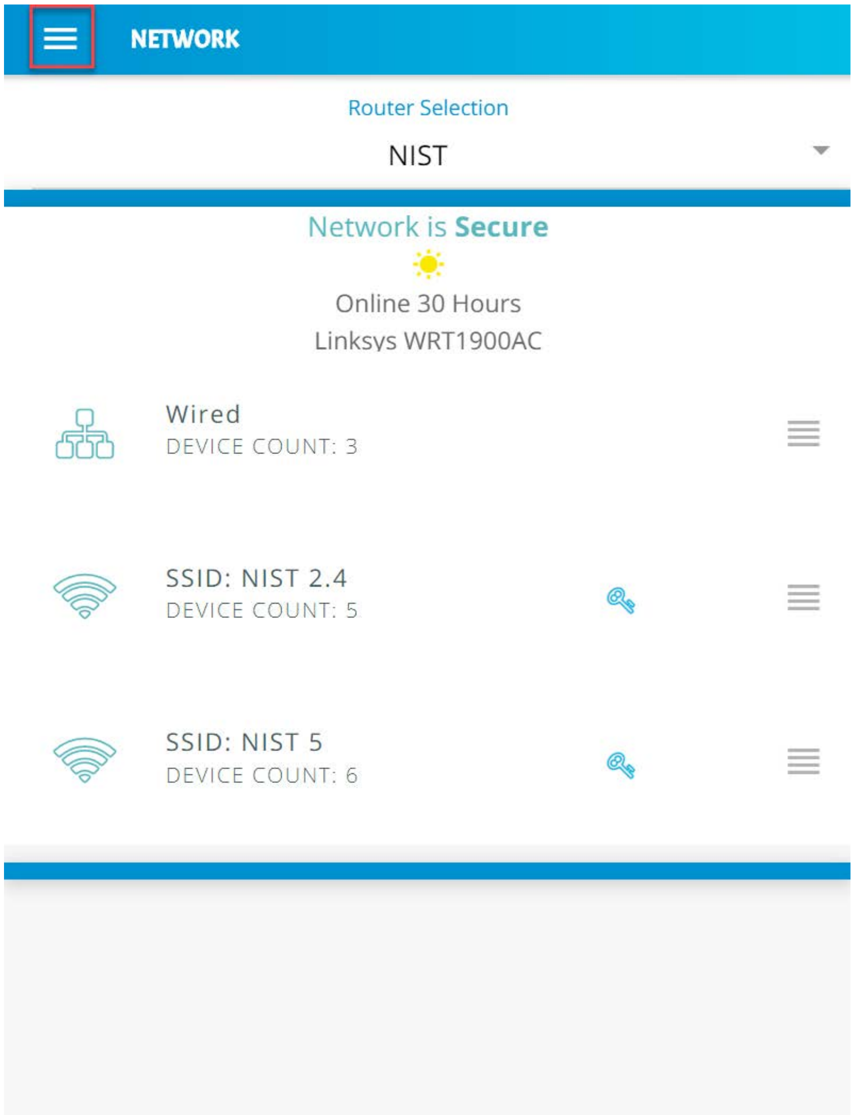
5. The home screen will show the network overview:



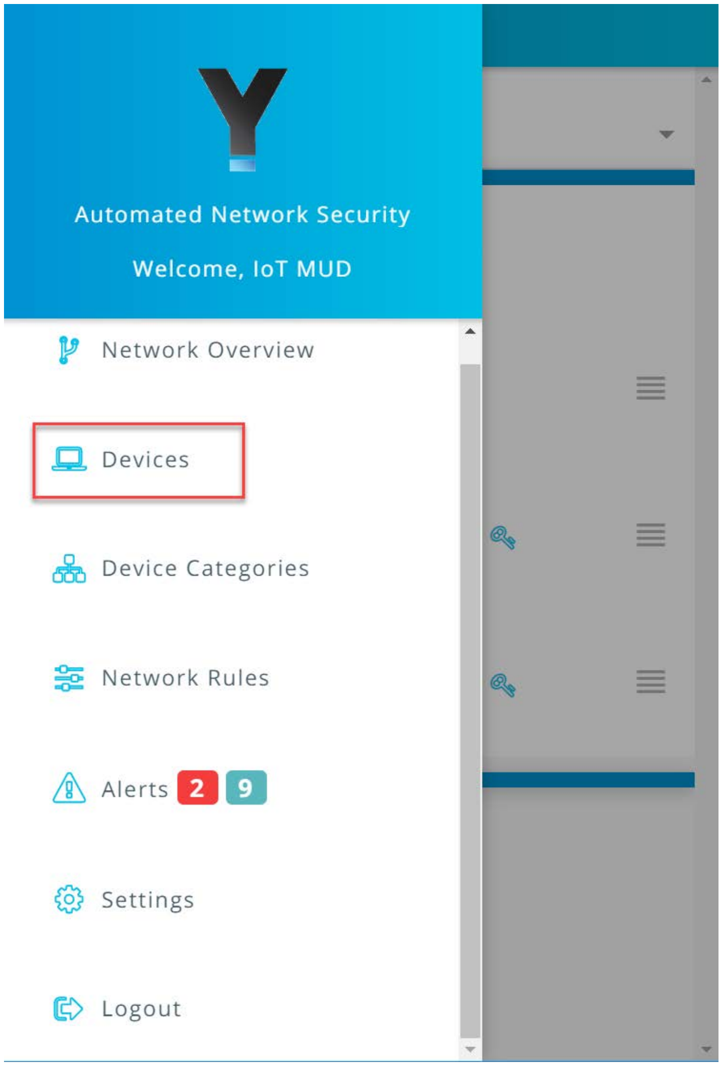
3.9.3.2 Yikes! MUD-Capable IoT Device Discovery

This section details the Yikes! MUD-capable IoT device discovery capability. This feature is accessible through the Yikes! mobile application and identifies all MUD-capable IoT devices that are connected to the network.

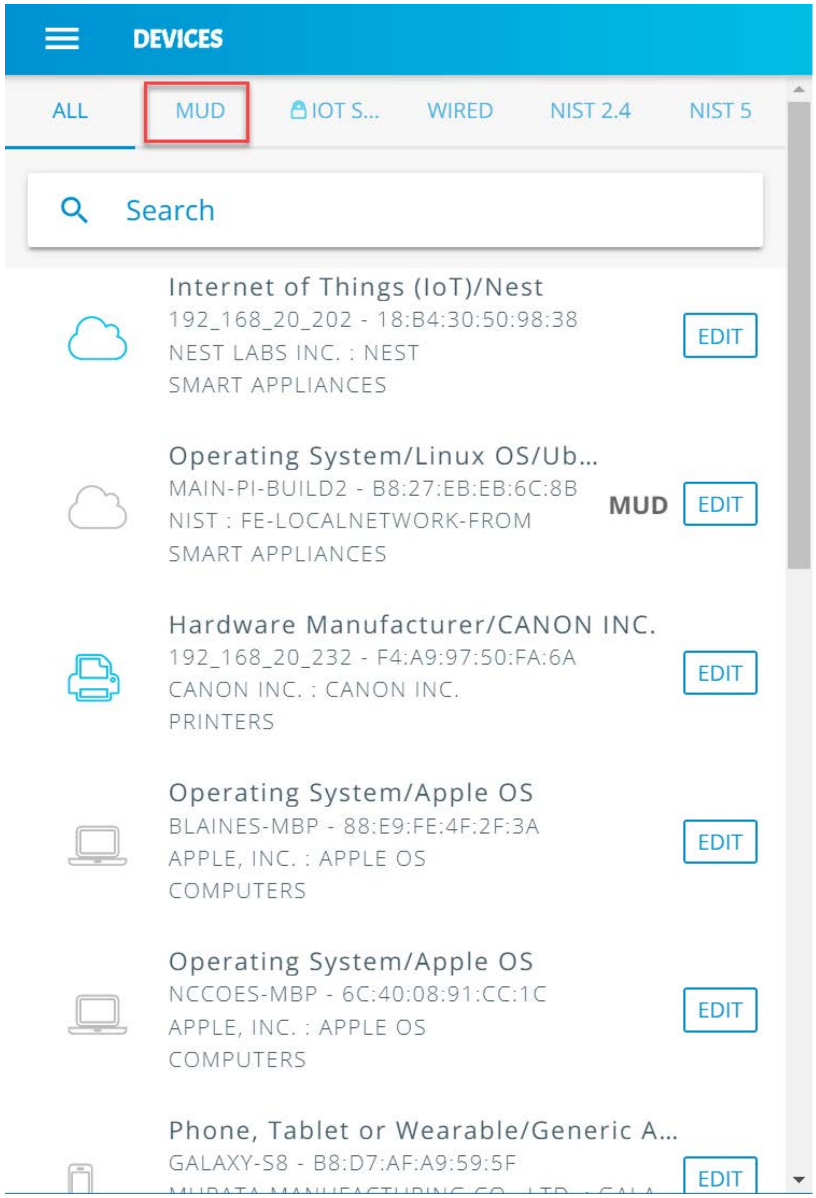
1. Open the menu pane in the UI:



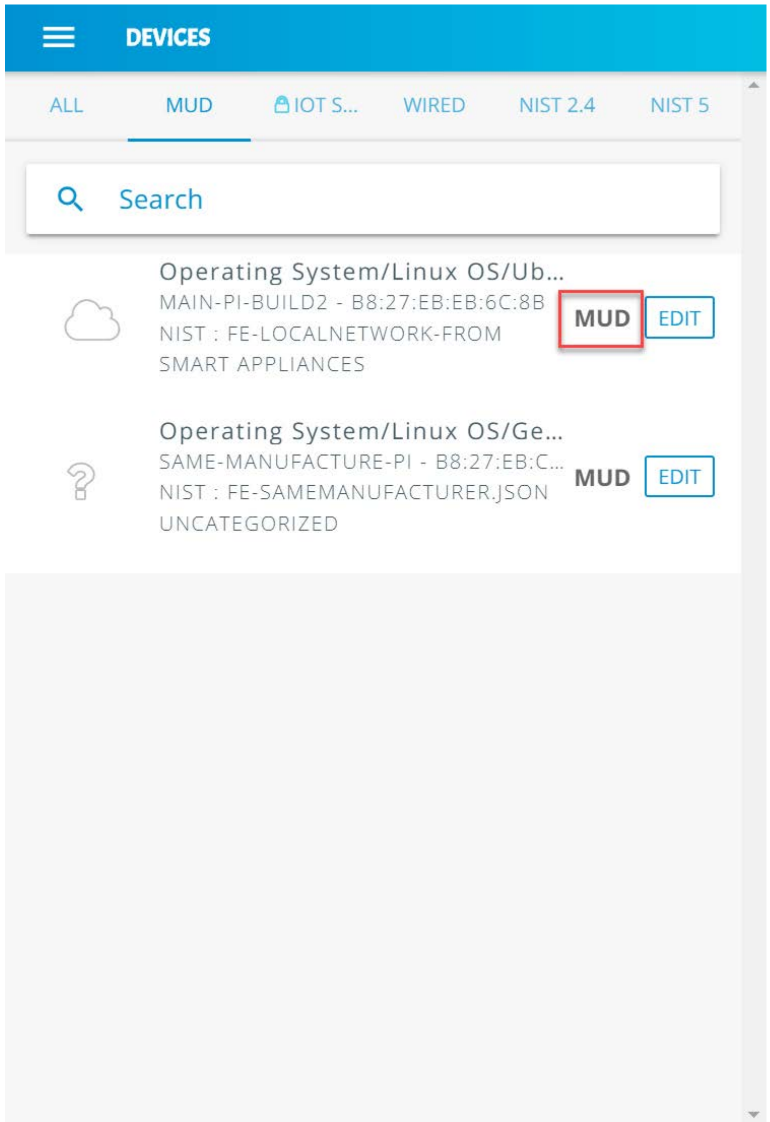
1485 2. Click the **Devices** button to open the devices menu:

1486 

3. Click the **MUD** tab to switch from the **ALL** device view to review the MUD-capable IoT devices connected to the network:



4. All MUD-capable devices on the network will have the **MUD** label, as seen below:

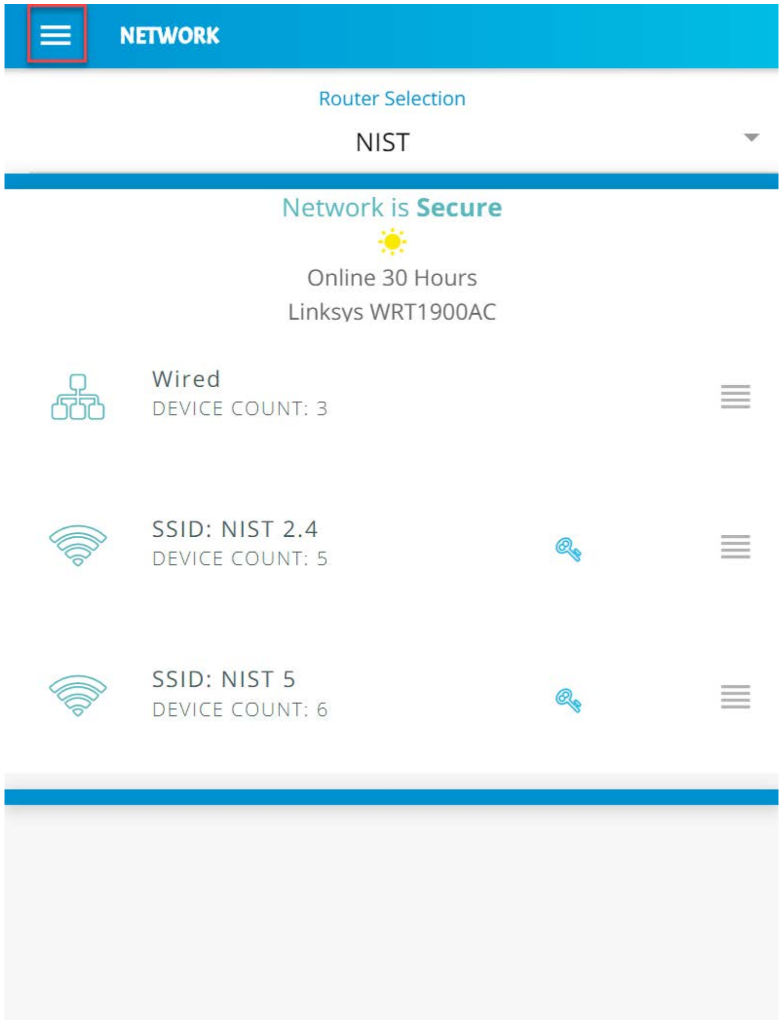


3.9.3.3 Yikes! Alerts

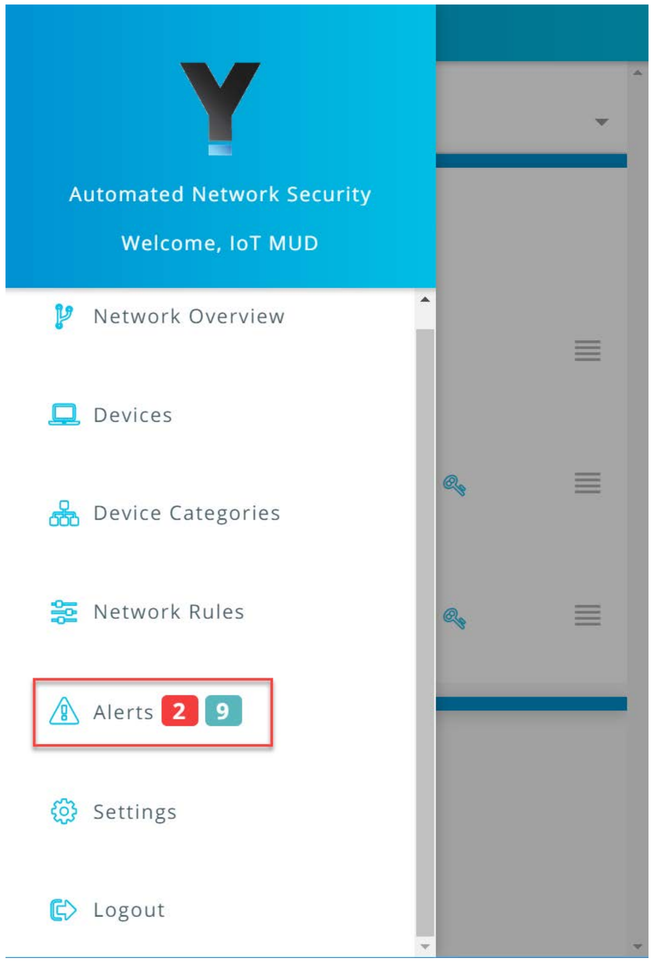
This section details the Yikes! alerting capability. This feature is accessible through the Yikes! mobile application and notifies users when new devices have been connected to the network. Additionally, this feature alerts the user when new devices are not recognized as known devices and are placed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

From the Yikes! mobile application, the user can edit the information about the device (e.g., name, make, and model) and modify the device's category or can choose to ignore the alert by removing the notification.

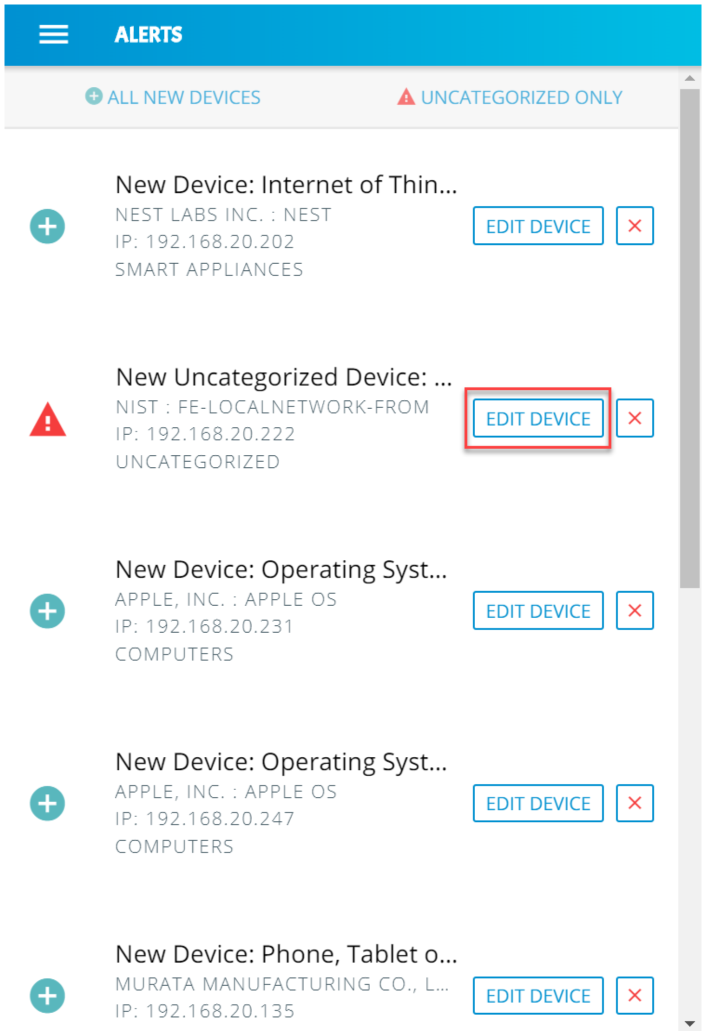
1. Open the menu pane in the UI:



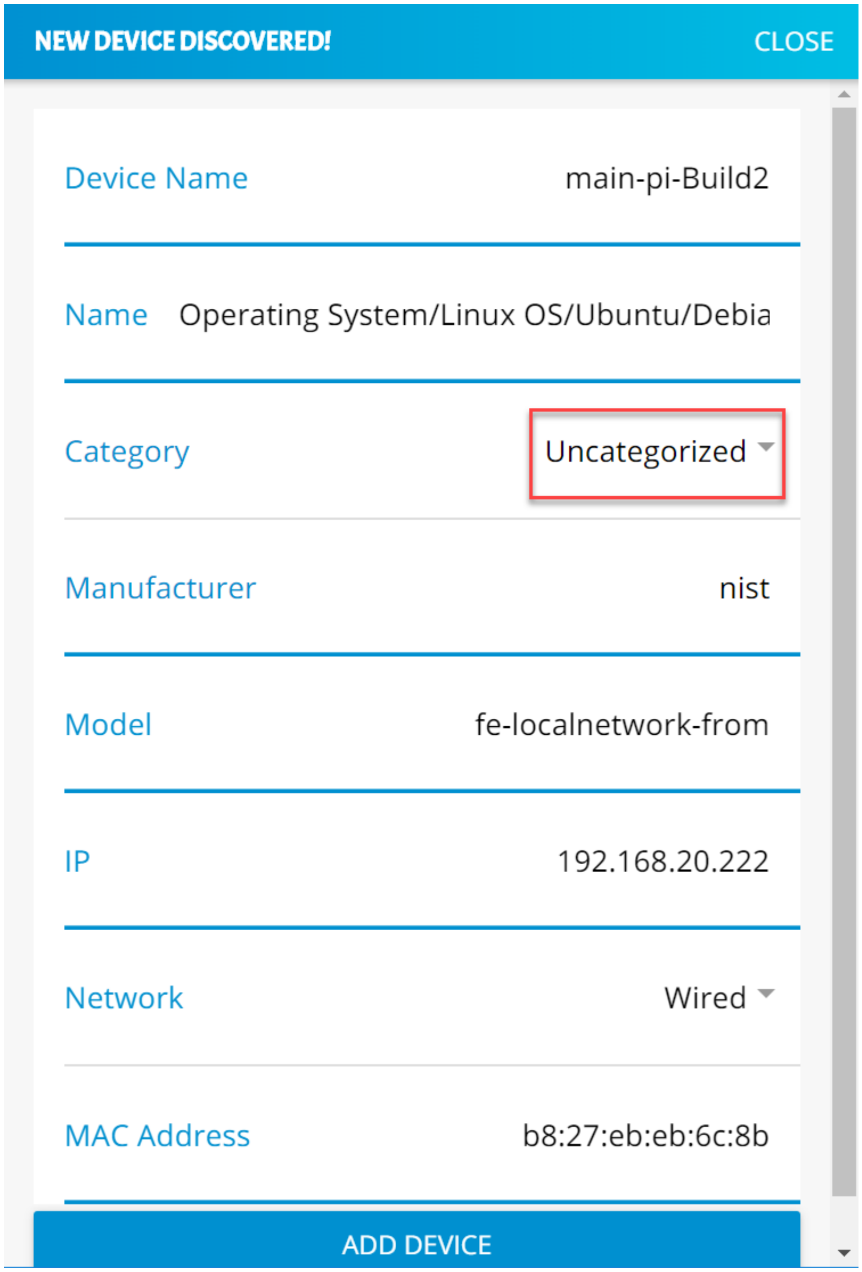
2. Click the **Alerts** to open the Alerts menu:



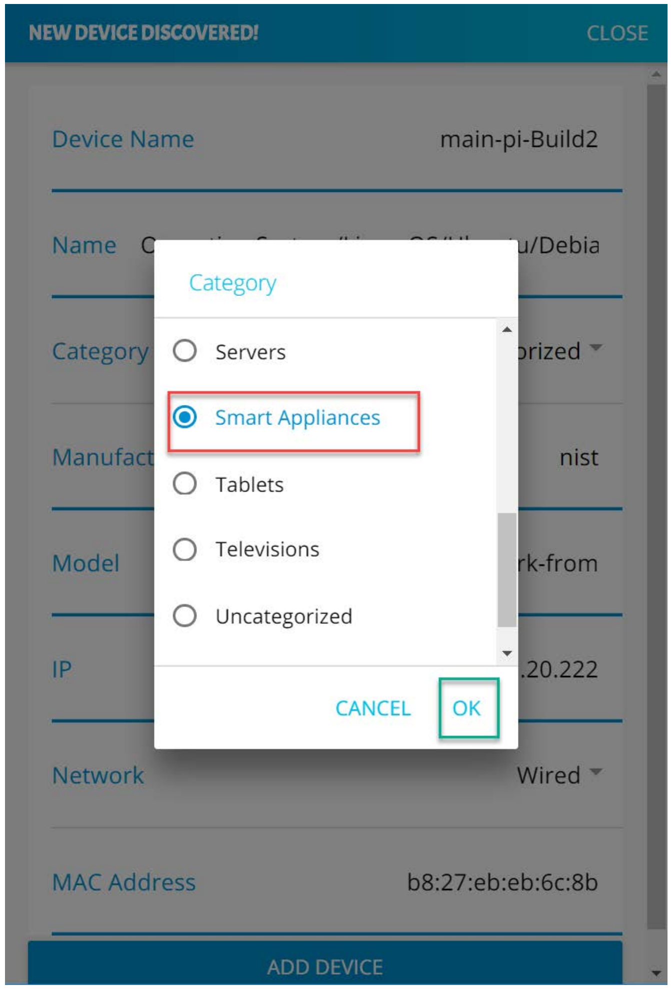
3. Select a device to edit the device information and category by clicking **Edit Device:**



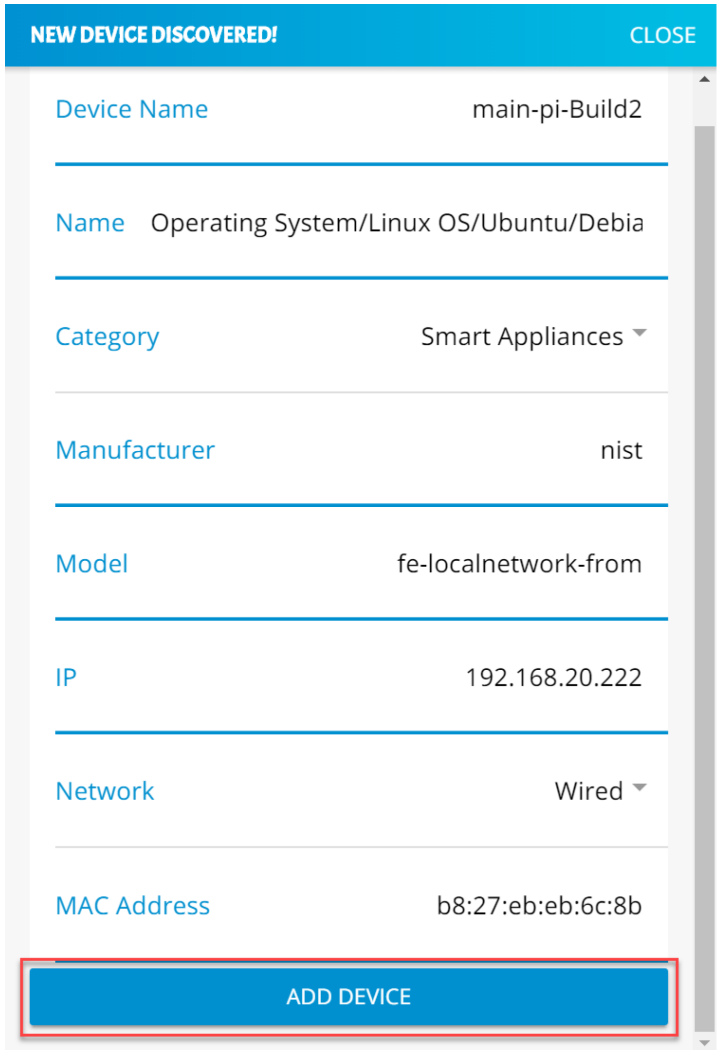
1506 4. Modify the **Category** of the device by clicking the device's current category:

1507 

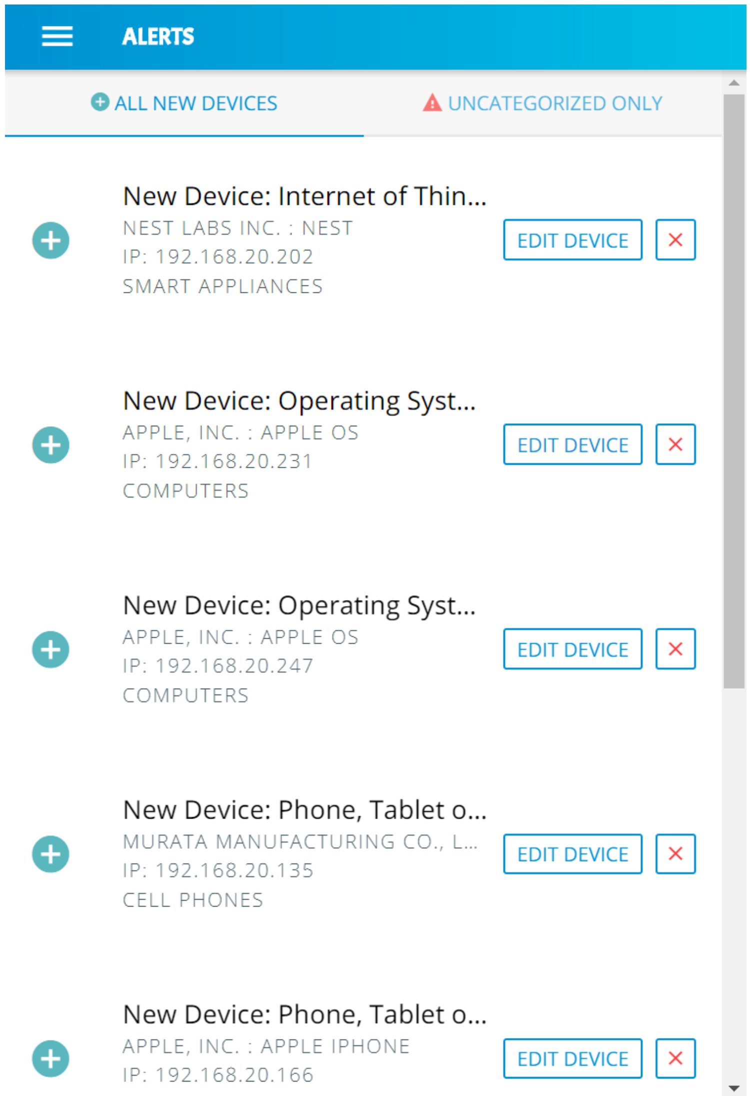
5. Select the desired category, in this case **Smart Appliances,** and click **OK:**



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



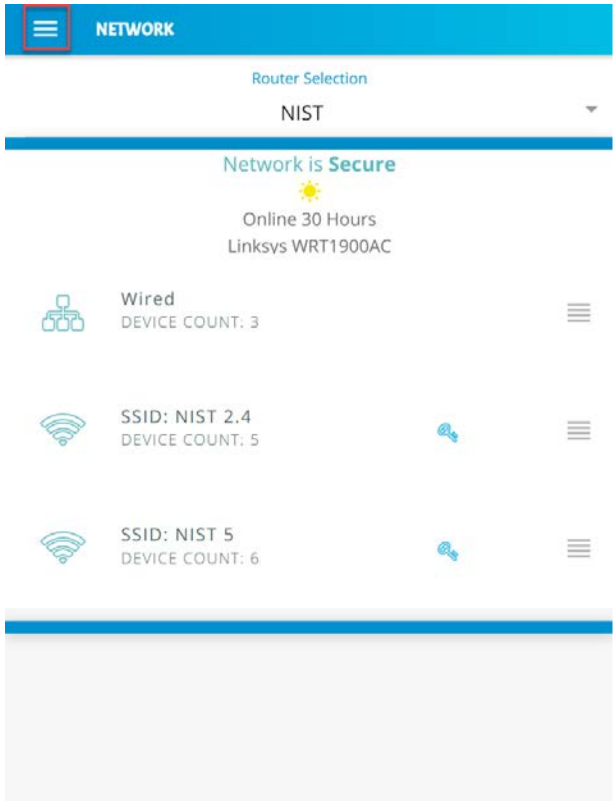
7. The alerts menu will update and no longer include the device that was just modified and added:



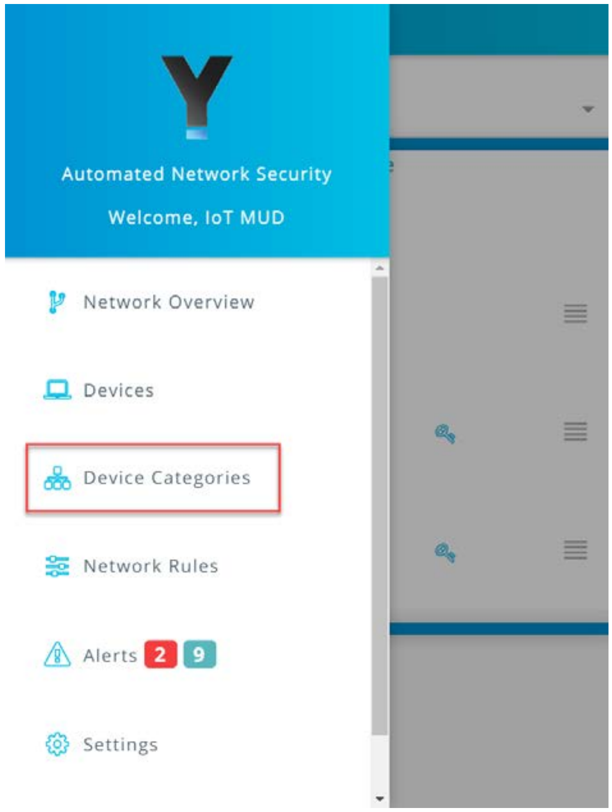
3.9.3.4 Yikes! Device Categories and Setting Rules

The Yikes! mobile application provides the capability to view predefined device categories and set rules for local communication between categories of devices on the local network and internet rules for all devices in a selected category.

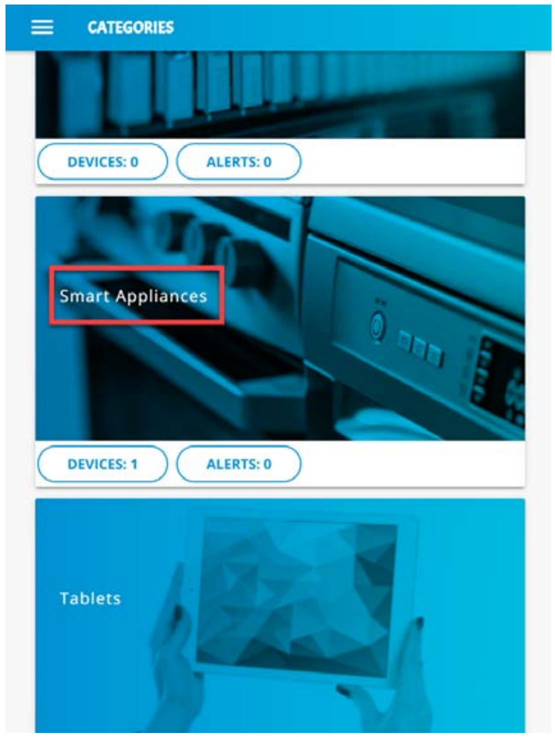
1. Click the menu bar to open the menu pane:



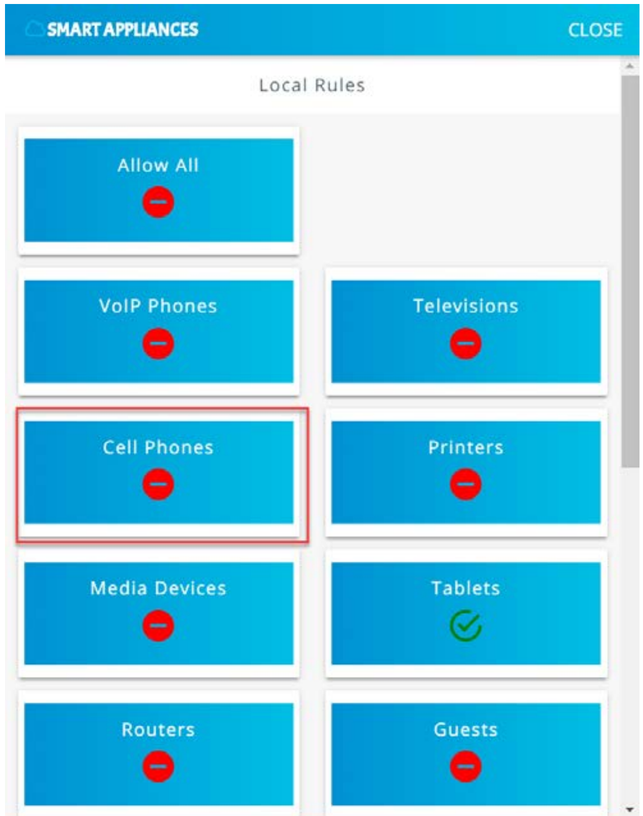
2. Click the **Device Categories** option to view all device categories:

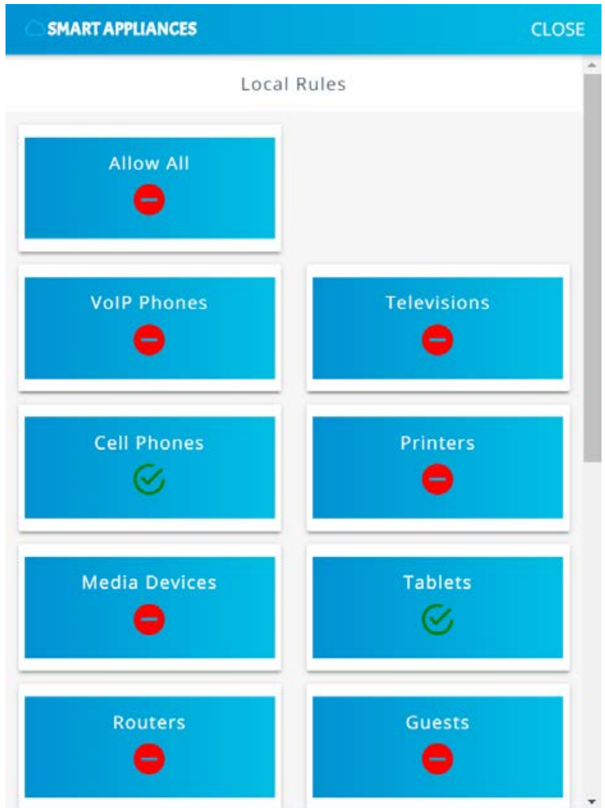


1523 3. Select the category of device to view and configure rules:

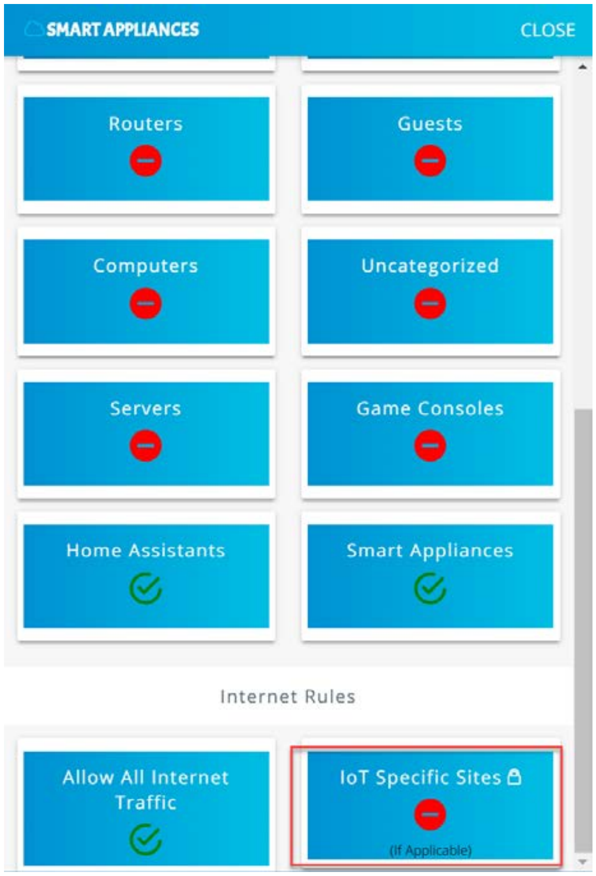
1524 

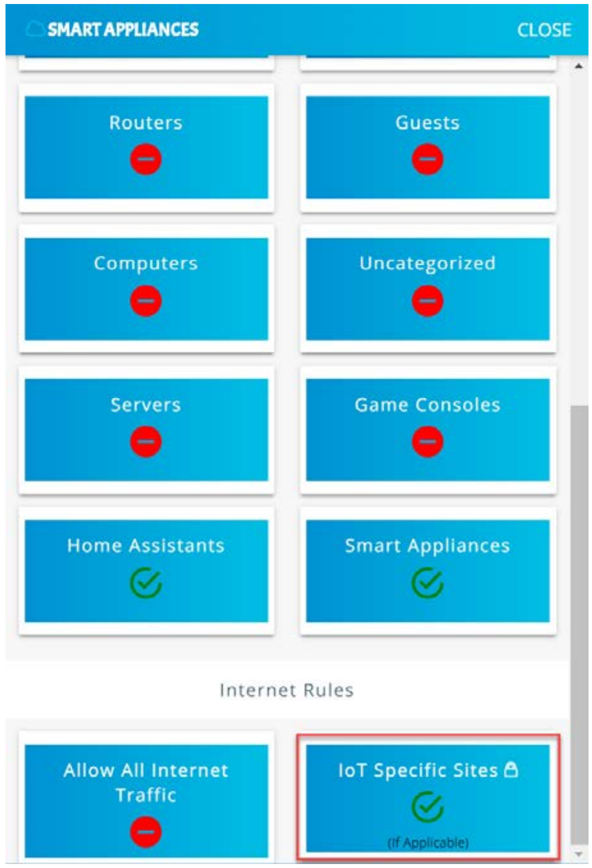
4. Modify local rules by clicking on the category of devices with which the selected category is permitted to communicate:




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:**

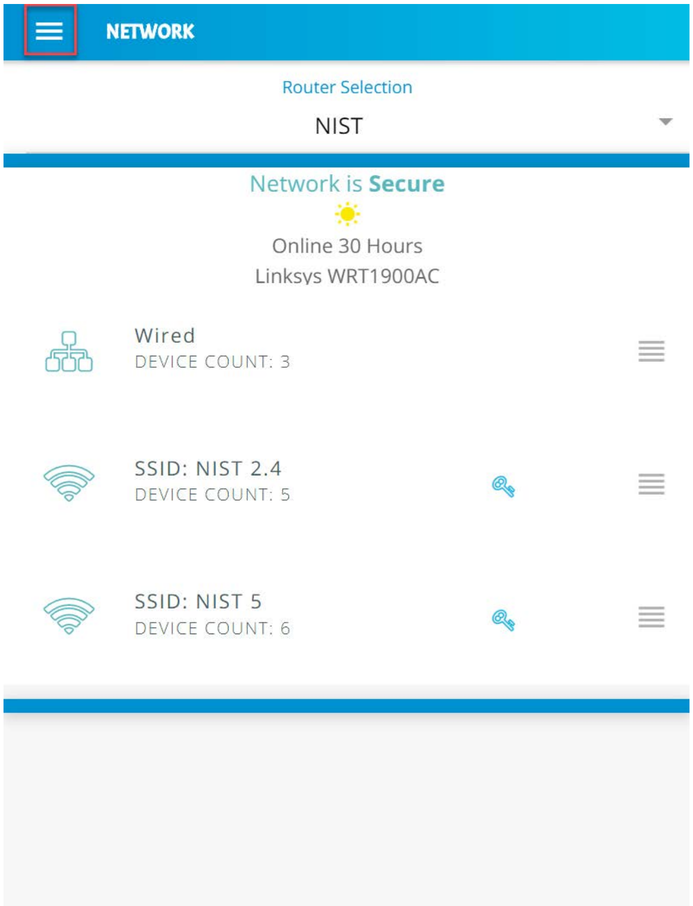
1531 
SMART APPLIANCES
CLOSE
Routers
Guests
Computers
Uncategorized
Servers
Game Consoles
Home Assistants
Smart Appliances
Internet Rules
Allow All Internet Traffic
IoT Specific Sites
(If Applicable)



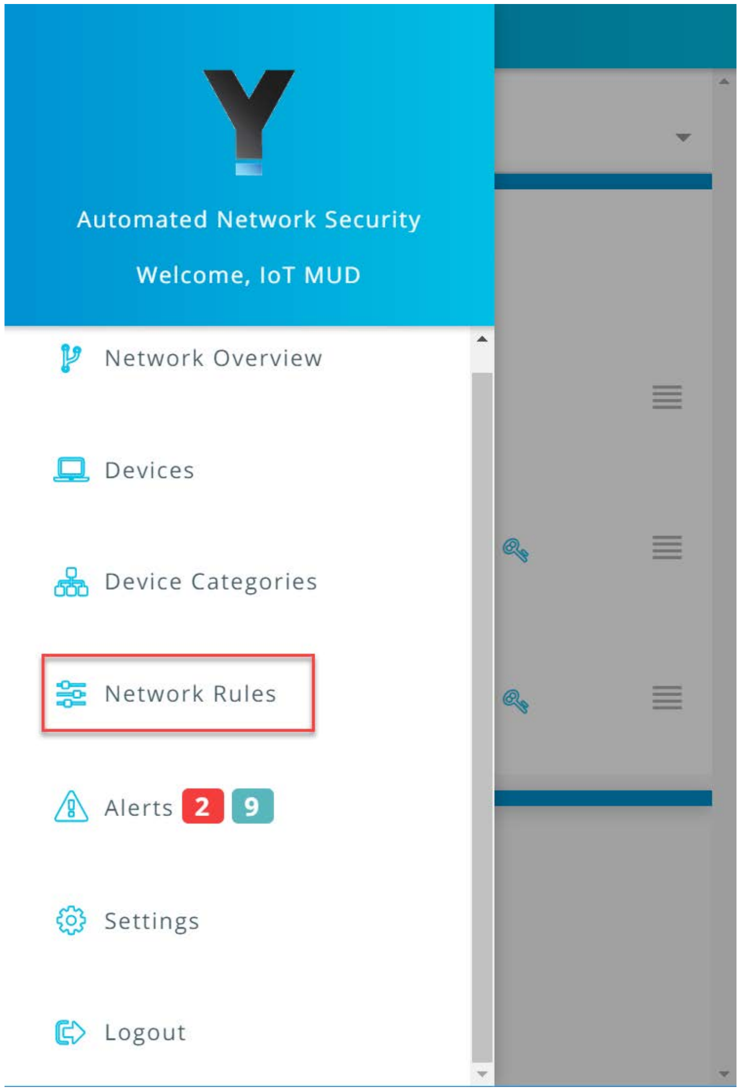
Smart appliances should now be permitted to communicate locally to Smart Appliances, Home Assistants, Tablets, Cell Phones, and, externally, to IoT Specific Sites.

3.9.3.5 Yikes! Network Rules

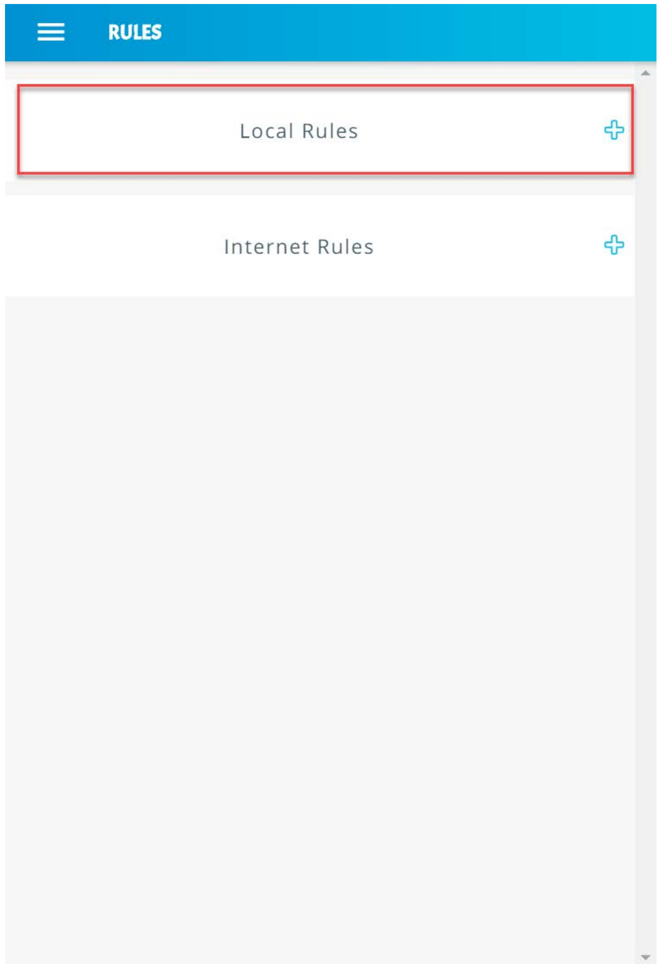
1. The Yikes! mobile application allows reviewing the rules that have been implemented on the network. These rules are divided into two main sections: Local Rules and Internet Rules. Local rules display the local communications permitted for each category of devices. Internet rules display the internet 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

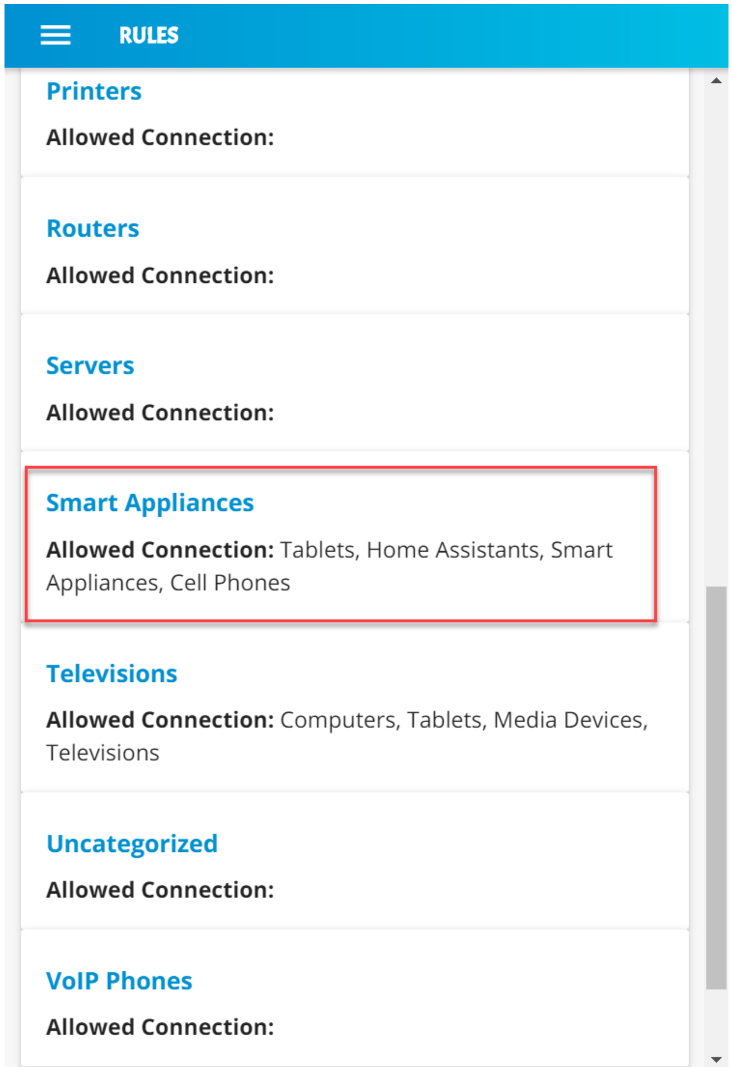
2. Click **Network Rules** to navigate to the rules menu:



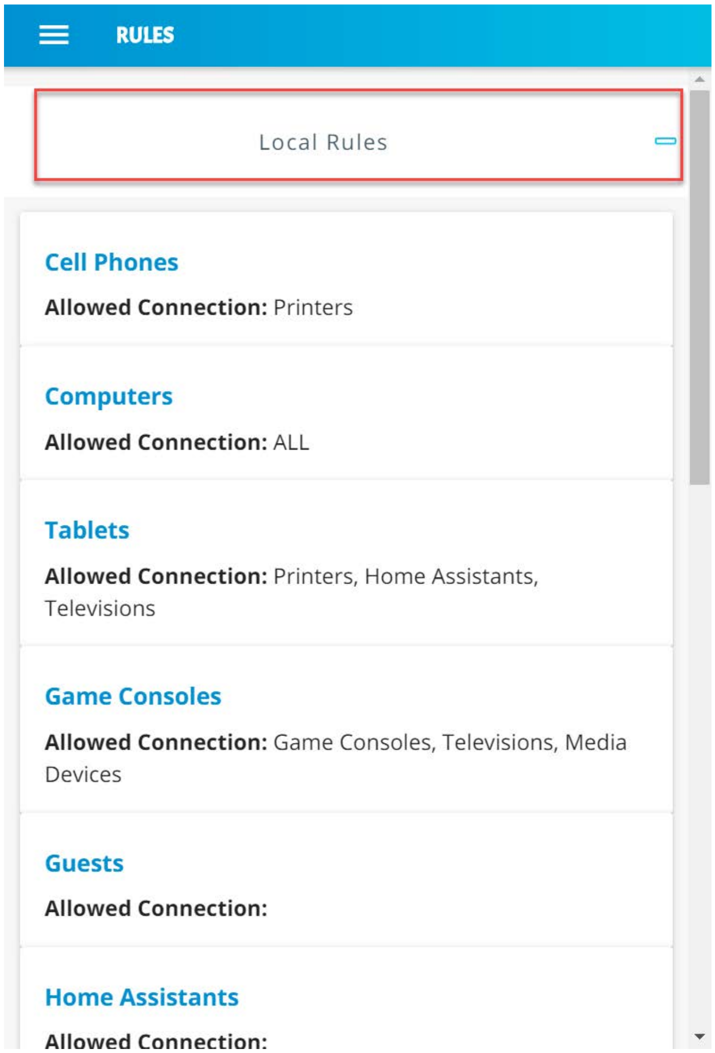
3. Click **Local Rules** to view the permitted local communications for each device category:



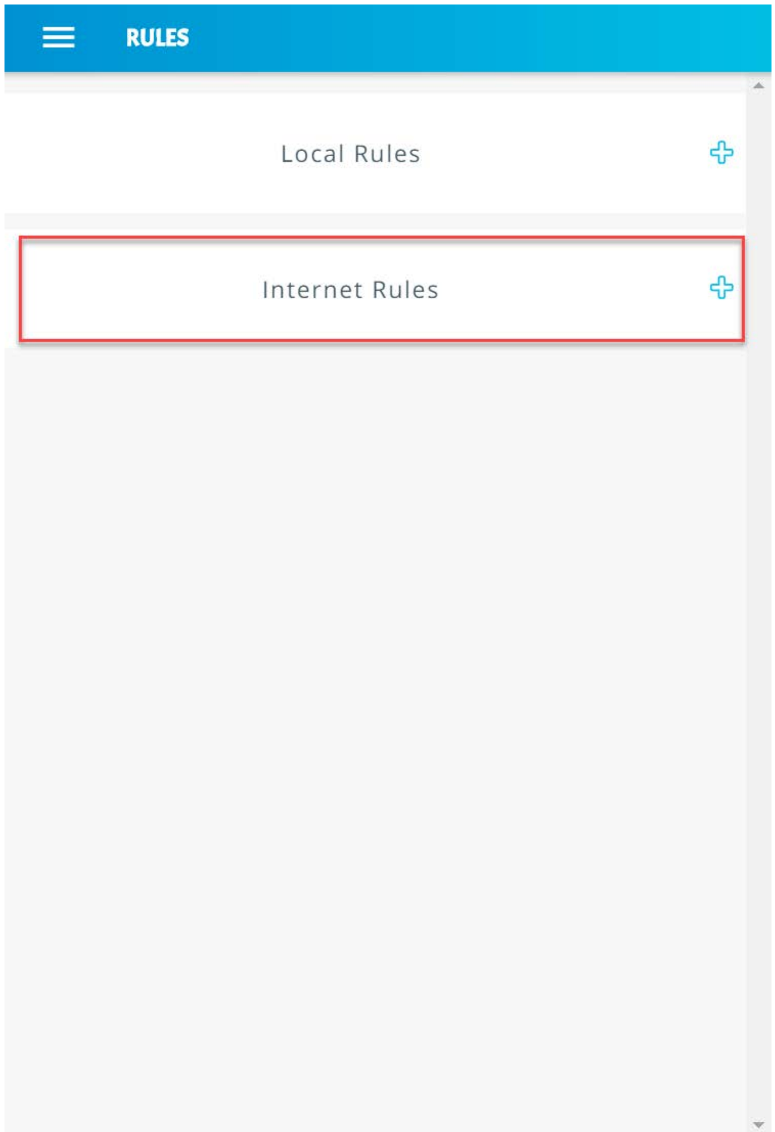
4. Scroll down to view the local rules for the **Smart Appliances** category:



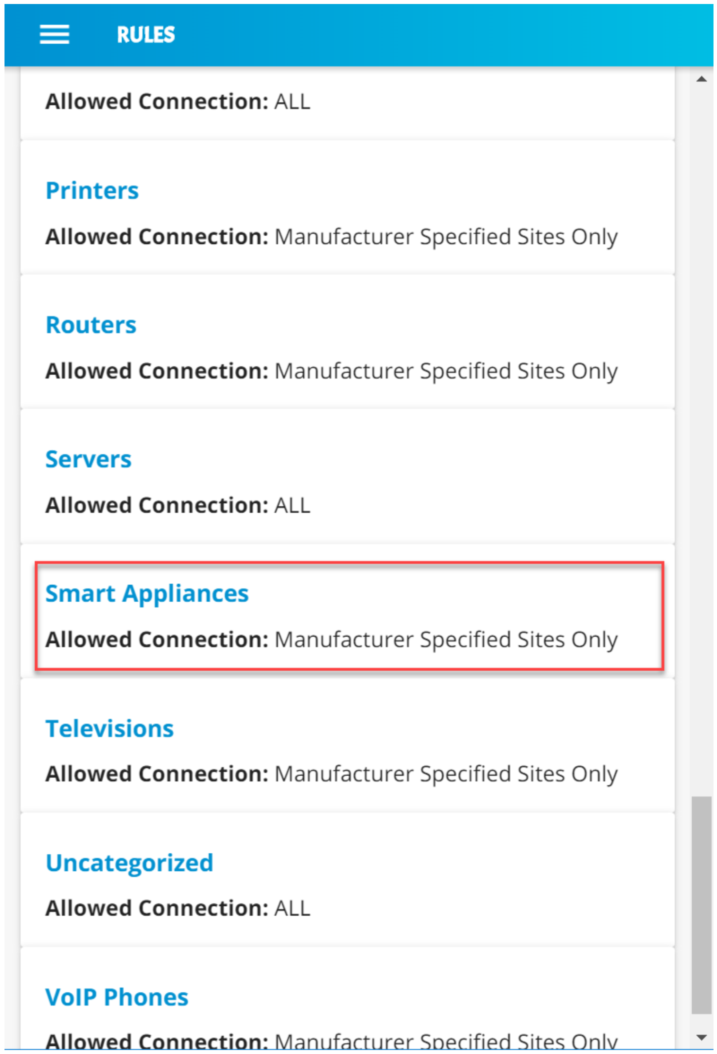
5. Minimize the rules by clicking on the **Local Rules** button:



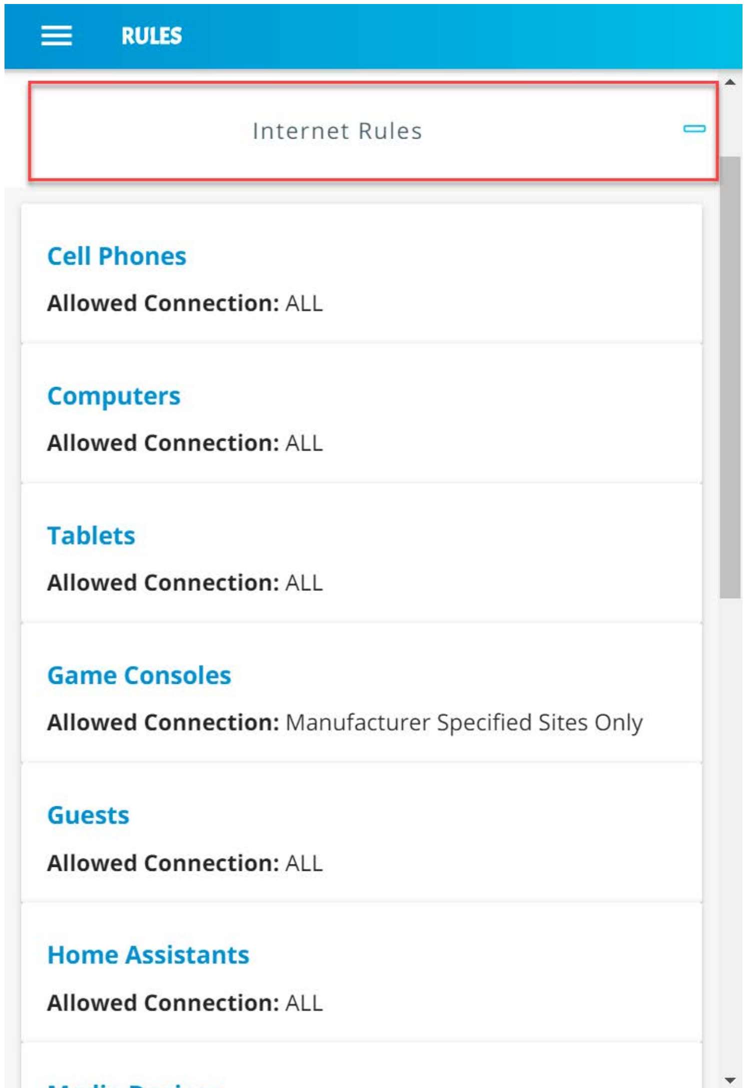
6. Expand the rules that show internet rules for device categories by clicking **Internet Rules:**



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



8. Minimize the rules by clicking 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

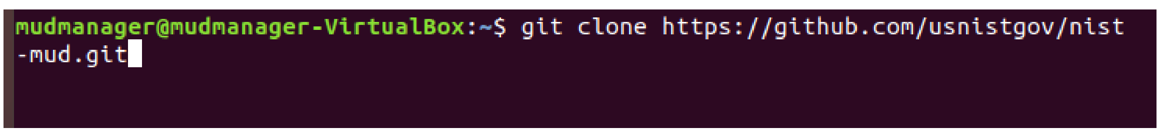
Build 4's GitHub page provides documentation that was followed to complete this section: https://github.com/usnistgov/nist-mud.

- 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

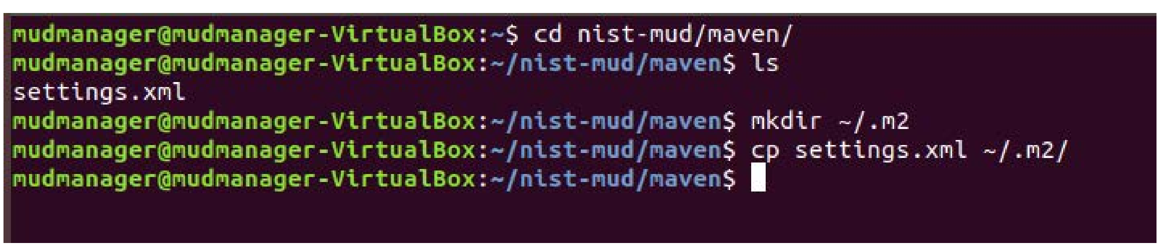
1. Execute the following command to clone the Git project:

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



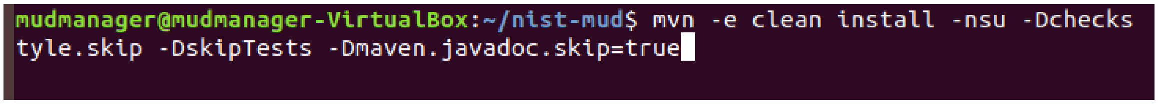
2. Copy the contents of `nist-mud/maven/settings.xml` to `~/.m2` by executing the commands below:

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



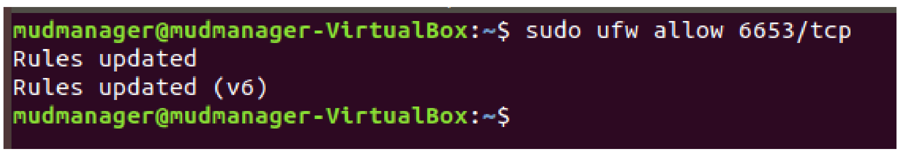
3. In the nist-mud directory, run the commands below:

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



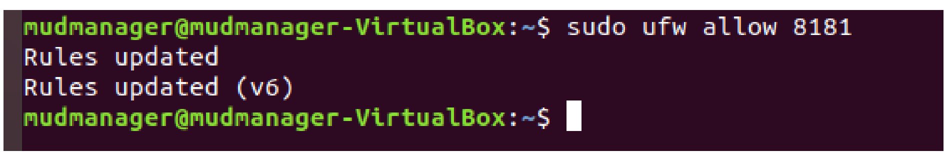
4. Open port 6653 on the controller stack for TCP access so the switches can connect by executing the command below:

```
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
```

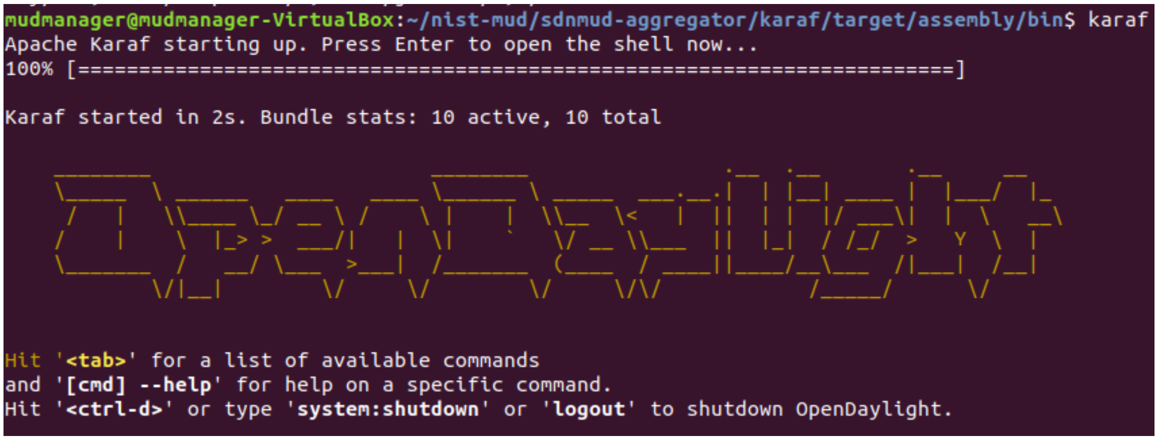


6. Change to the bin directory by executing the command below:

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

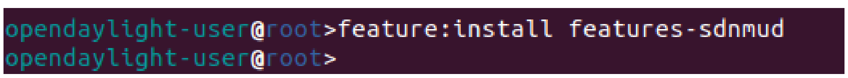
7. Run the command below:

```
./karaf clean
```

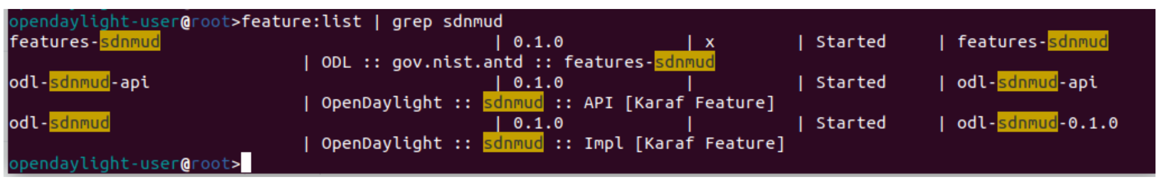


8. At the Karaf prompt, install MUD capabilities using:

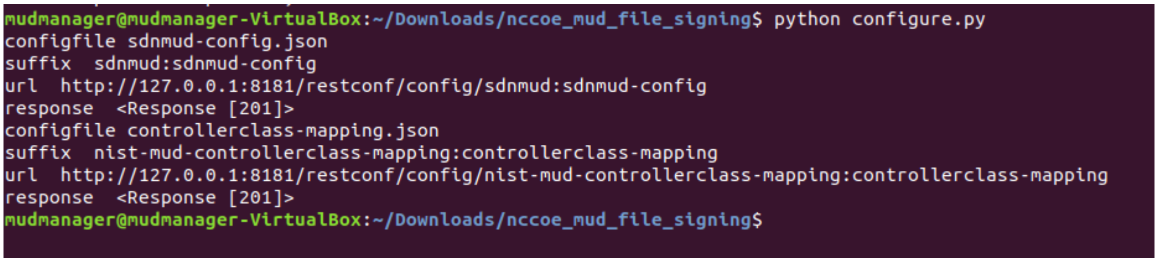
```
feature:install features-sdnmud
```



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.



```
mudmanager@mudmanager-VirtualBox:~/Downloads/nccoe_mud_file_signing$ python configure.py
configfile sdnmud-config.json
suffix  sdnmud:sdnmud-config
url  http://127.0.0.1:8181/restconf/config/sdnmud:sdnmud-config
response  <Response [201]>
configfile controllerclass-mapping.json
suffix  nist-mud-controllerclass-mapping:controllerclass-mapping
url  http://127.0.0.1:8181/restconf/config/nist-mud-controllerclass-mapping:controllerclass-mapping
response  <Response [201]>
mudmanager@mudmanager-VirtualBox:~/Downloads/nccoe_mud_file_signing$
```

Example Python script (`configure.py`):

```
import requests
import json
import argparse
import os

if __name__=="__main__":
    if os.environ.get("CONTROLLER_ADDR") is None:
       print "Please set environment variable CONTROLLER_ADDR to the address of the
opendaylight controller"

    controller_addr = os.environ.get("CONTROLLER_ADDR")

    headers= {"Content-Type":"application/json"}
    for (configfile,suffix) in {
        ("sdnmud-config.json", "sdnmud:sdnmud-config"),
        ("controllerclass-mapping.json","nist-mud-controllerclass-
mapping:controllerclass-mapping") }:
        data = json.load(open(configfile))
        print "configfile", configfile
        print "suffix ", suffix
        url = "http://" + controller_addr + ":8181/restconf/config/" + suffix
        print "url ", url
        r = requests.put(url, data=json.dumps(data), headers=headers , auth=('admin',
'admin'))
        print "response ", r
```

Example controller class mapping (`controllerclass-mapping.json`):


```
{
"controllerclass-mapping" :  {
    "switch-id" : "openflow:123917682138002",
    "controller" : [
        {
                "uri" :  "urn:ietf:params:mud:dns",
                "address-list" : [ "10.0.41.1" ]
        },
        {
                "uri" :  "urn:ietf:params:mud:dhcp",
                "address-list" : [ "10.0.41.1" ]
        },
        {
                "uri" :  "https://controller.nist.local",
                "address-list" : [ "10.0.41.225" ]
        },
```

```
        {
               "uri" :  "https://sensor.nist.local/nistmud1",
               "address-list" : [ "10.0.41.225" ]
        }
    ],
    "local-networks": [ "10.0.41.0/24" ],
    "wireless" : true
 }
}
```

Example SDN MUD configuration (`sdnmud-config.json`):

```
{
 "sdnmud-config" : {
       "ca-certs": "lib/security/cacerts",
       "key-pass" : "changeit",
       "trust-self-signed-cert" : true,
       "mfg-id-rule-cache-timeout": 120,
       "relaxed-acl" : false
  }
}
```

5.2 MUD File Server

5.2.1 MUD File Sever Overview

The MUD file server is responsible for serving the MUD file and the corresponding signature file upon request from the MUD manager. For testing purposes, the MUD file server is run on 127.0.0.1 on the same machine as the MUD manager. This allows us to examine the logs to check if the MUD file has been retrieved. For testing purposes, host name verification for the TLS connection to the MUD file server is disabled in the configuration of the MUD manager.

5.2.2 Configuration Overview

The following subsections document the software, hardware, and network configurations for the MUD file server.

5.2.2.1 Hardware Configuration

The MUD file server was hosted on the same machine as the SDN controller.

5.2.2.2 Network Configuration

The MUD file server was hosted on the same machine as the SDN controller. To direct the MUD manager to retrieve the MUD files from the MUD file server, the host name of the two manufacturers that are present in the MUD URLs used for testing are both mapped to 127.0.0.1 in the */etc/hosts* file of the Java Virtual Machine in which the MUD manager is running. This static configuration is read by the MUD manager when it starts. The name resolution information in the `/etc/hosts` file directs the MUD manager to retrieve the test MUD files from the MUD file server.

5.2.2.3 Software Configuration

In this build, serving MUD files requires Python 2.7 and the Python requests package. These may be installed using *apt* and *pip*. After creation of the MUD files by using mudmaker.org, the MUD files were signed, and the certificates used for signing were imported into the trust store of the Java Virtual Machine in which the MUD manager is running.

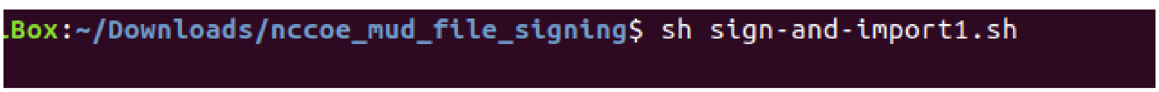
5.2.3 Setup

5.2.3.1 MUD File Creation

This build also leveraged the MUD Maker online tool found at www.mudmaker.org. For detailed instructions on creating a MUD file using this online tool, please refer to Build 1's [MUD File Creation](#) section.

5.2.3.2 MUD File Signing

1. Sign and import the desired MUD files. An example script (`sign-and-import1.sh`) can be found below.



The shell script that was used in this build is shown below. This script generates a signature based on the private key of a DigiCert-issued certificate and imports the certificate into the trust store of the Java Virtual Machine. This is done for both MUD files.

```
CACERT=DigiCertCA.crt
MANUFACTURER_CRT=nccoe_mud_file_signing.crt
MANUFACTURER_KEY=mudsign.key.pem
MANUFACTURER_ALIAS=sensor.nist.local
MANUFACTURER_SIGNATURE=mudfile-sensor.p7s
MUDFILE=mudfile-sensor.json

openssl cms -sign -signer $MANUFACTURER_CRT -inkey $MANUFACTURER_KEY -in $MUDFILE -
binary -noattr -outform DER -certfile $CACERT  -out $MANUFACTURER_SIGNATURE
openssl cms -verify -binary  -in $MANUFACTURER_SIGNATURE  -signer $MANUFACTURER_CRT -
inform DER  -content $MUDFILE

MANUFACTURER_ALIAS=otherman.nist.local
MUDFILE=mudfile-otherman.json
MANUFACTURER_SIGNATURE=mudfile-otherman.p7s
openssl cms -sign -signer $MANUFACTURER_CRT -inkey $MANUFACTURER_KEY -in $MUDFILE -
binary -noattr -outform DER -certfile $CACERT  -out $MANUFACTURER_SIGNATURE
openssl cms -verify -binary  -in $MANUFACTURER_SIGNATURE  -signer $MANUFACTURER_CRT -
inform DER  -content $MUDFILE
```

```
sudo -E $JAVA_HOME/bin/keytool -delete -alias digicert -keystore
$JAVA_HOME/jre/lib/security/cacerts -storepass changeit
sudo -E $JAVA_HOME/bin/keytool -importcert -file $CACERT -alias digicert -keystore
$JAVA_HOME/jre/lib/security/cacerts -storepass changeit
```

5.2.3.3 MUD File Serving

Run a script that serves desired MUD files and signatures. An example Python script (`mudfile-server.py`) can be found below.

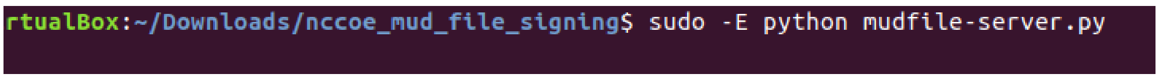
1. Save a copy of the **mudfile-server.py** Python script onto the NIST SDN controller/MUD manager configured in Section 5.1:

```
import BaseHTTPServer, SimpleHTTPServer
import ssl
import urlparse
# Dummy manufacturer server for testing

class MyHTTPRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        print ("DoGET " + self.path)
        self.send_response(200)
        if self.path == "/nistmud1" :
           with  open("mudfile-sensor.json", mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        elif self.path == "/nistmud2" :
           with  open("mudfile-otherman.json", mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        elif self.path == "/nistmud1/mudfile-sensor.p7s":
           with open("mudfile-sensor.p7s",mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        elif self.path == "/nistmud2/mudfile-otherman.p7s":
           with open("mudfile-otherman.p7s",mode="r") as f:
                data = f.read()
          print("Read " + str(len(data)) + " chars ")
                self.send_header("Content-Length", len(data))
                self.end_headers()
                self.wfile.write(data)
        else:
           print("UNKNOWN URL!!")
           self.wfile.write(b'Hello, world!')
```

```
httpd = BaseHTTPServer.HTTPServer(('0.0.0.0', 443), MyHTTPRequestHandler)
httpd.socket = ssl.wrap_socket (httpd.socket, keyfile='./mudsigner.key',
certfile='./mudsigner.crt', server_side=True)
httpd.serve_forever()
```

2. From the same directory as the previous step, execute the command below to start the MUD file server:

   ```
   sudo -E python mudfile-server.py
   ```



5.3 Northbound Networks Zodiac WX Access Point

5.3.1 Northbound Networks Zodiac WX Access Point Overview

The Zodiac WX, in addition to being a wireless access point, includes the following logical components: an SDN switch, a NAT router, a DHCP server, and a DNS server. The Zodiac WX is powered by OpenWRT and Open vSwitch. Open vSwitch directly integrates into the wireless configuration. The Zodiac WX works with any standard OpenFlow-compatible controllers and requires no modifications because it appears to the controller as a standard OpenFlow switch.

5.3.2 Configuration Overview

The following subsections document the network, software, and hardware configurations for the SDN-capable Northbound Networks Zodiac WX.

5.3.2.1 Network Configuration

The access point is configured to have a static public address on the public side of the NAT. For purposes of testing, we use 203.0.113.x addresses on the public network. The public side of the NAT is given the address of 203.0.113.1. The DHCP server is set up to allocate addresses to wireless devices on the LAN. The SDN controller/MUD manager is connected to the public side of the NAT. The Open vSwitch configuration for the access point is given the address of the SDN controller, which is shown in the setup below.

5.3.2.2 Software Configuration

At this implementation, no additional software configuration was required.

5.3.2.3 Hardware Configuration

At this implementation, no additional hardware configuration was required.

5.3.3 Setup

On the Zodiac WX, DNSmasq supports both DHCP and DNS. For testing purposes, it will be necessary to access several web servers (two update servers called www.nist.local and an unapproved server called www.antd.local). The following commands enable the Zodiac WX to resolve the web server host names to their IP addresses.

1. Set up the access point to resolve the addresses for the web server host names by opening the file */etc/dnsmasq.conf* on the access point.

2. Add the following line to the *dnsmasq.conf* file:

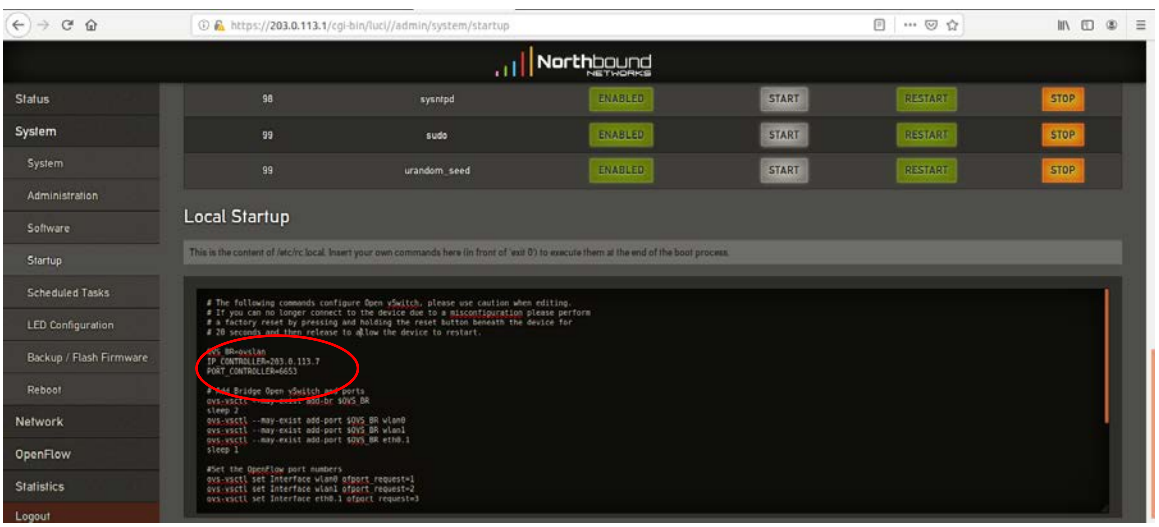
```
addn-hosts=/etc/hosts.nist.local
```

```
addn-hosts=/etc/hosts.nist.local
- /etc/dnsmasq.conf [Readonly] 38/38 100%
```

3. The file */etc/hosts.nist.local* has the host name to address mapping. The mapping used for our tests is shown below (Note that the host www.nist.local maps to two addresses on the public side).

```
203.0.113.13 www.nist.local
203.0.113.15 www.nist.local
203.0.113.14 www.antd.local
~
```

4. On the Zodiac WX configuration web page in the System->Startup tab, indicate where (IP address and port) the Open vSwitch Daemon connects to the controller.



5.4 DigiCert Certificates

DigiCert's CertCentral web-based platform allows provisioning and management of publicly trusted X.509 certificates for a variety of purposes. After establishing an account, clients can log in, request, renew, and revoke certificates by using only a browser. For Build 4, the Premium Certificate created in Build 1 was leveraged for signing the MUD files. To request and implement DigiCert certificates, follow the documentation in Build 1's DigiCert Certificates section and subsequent sections.

5.5 IoT Devices

5.5.1 IoT Devices Overview

This section provides configuration details for the Linux-based Raspberry Pis used in the build, which emit MUD URLs by using DHCP.

5.5.2 Configuration Overview

The devices used in this build were multiple Raspberry Pi development kits that were configured to act as IoT devices. The devices run Raspbian 9, a Linux-based operating system, and are configured to emit a MUD URL during a typical DHCP transaction. These devices were used to test interactions related to MUD capabilities.

5.5.2.1 Network Configuration

The kits are connected to the network over a wireless connection. Their IP addresses are assigned dynamically by the DHCP server on the Zodiac WX access point.

5.5.2.2 Software Configuration

The Raspberry Pis are configured on Raspbian. They also utilized dhclient as their default DHCP clients to manually initiate a DHCP interaction. This DHCP client is installed natively on many Linux distributions and can be installed using a preferred package manager if not currently present. Dhclient uses a configuration file: `/etc/dhclient.conf`. This needs to be modified to include the MUD URL that the device will emit in its DHCP requests. (The modification details are provided in the setup information below.)

5.5.2.3 Hardware Configuration

Multiple Raspberry Pi 3 Model B devices were used.

5.5.3 Setup

Each Raspberry Pi used in this build was intended to represent a different class of device (manufacturer, other manufacturer, local networks, controller classes). The type of device was determined by the MUD

URL being emitted by the device. If no MUD URL is emitted, the device is an unclassified local network device.

1. On each Pi, changes were made to `/etc/network/interfaces` to add a line that allows the Pi to authenticate to the access point. The following line is added to the network interface as shown below:

   `wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf.northbound`

```
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf.northbound
```

   The file (*/etc/wpa_supplicant/wpa_supplicant.conf.northbound*) is shown below:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
        ssid="ZodiacWX_24GHz"
        psk="66666666"
}
```

2. A dhclient configuration file can be altered (by adding information) to allow for emission of a MUD URL in the DHCP transaction. Modify the *dhclient.conf* file with the command:

   `vi /etc/dhcp/dhclient.conf`

3. A send MUD URL line must be added as well as a `mud-url` in the request line. In this build, multiple MUD URLs were transmitted, depending on the type of the device. Example alterations made to dhclient configuration files can be seen below:

   send mud-url = "*https://sensor.nist.local/nistmud1*";

   send mud-url = "*https://otherman.nist.local/nistmud2*";

```
send mud-url = "https://sensor.nist.local/nistmud1";

request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, domain-search, host-name, mud-url,
        dhcp6.name-servers, dhcp6.domain-search,
        netbios-name-servers, netbios-scope, interface-mtu,
        rfc3442-classless-static-routes, ntp-servers,
        dhcp6.fqdn, dhcp6.sntp-servers;
```

4. To control the time at which the MUD URL is emitted, we manually reacquire the DHCP address rather than have the device acquire the MUD URL on boot. Emit the MUD URL and attain an IP address by sending the altered dhclient configuration file manually with the following commands:

```
sudo rm /var/lib/dhcp/dhclient.leases
sudo ifconfig wlan0 0.0.0.0
sudo dhclient -v wlan0 -cf /etc/dhcp/dhclient.conf.toaster
```

```
 sensor ] sudo rm /var/lib/dhcp/dhclient.leases; sudo ifconfig wlan0 0.0.0.0;  sudo dhclient -v wlan0 -cf /etc/dhcp/dhclient.conf.toaster
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/wlan0/b8:27:eb:3d:65:78
Sending on   LPF/wlan0/b8:27:eb:3d:65:78
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 4
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 10
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 11
DHCPREQUEST of 10.0.41.190 on wlan0 to 255.255.255.255 port 67
DHCPOFFER of 10.0.41.190 from 10.0.41.1
DHCPACK of 10.0.41.190 from 10.0.41.1
bound to 10.0.41.190 -- renewal in 21068 seconds.
 sensor ]
```

5.6 Update Server

5.6.1 Update Server Overview

This section provides configuration details for the Linux-based IoT development kit used in the build, which acts as an update server. This update server will attempt to access and be accessed by the IoT device, which, in this case, is one of the development kits built in the lab. The update server is a web server that hosts mock software update files to be served as software updates to our IoT device devkits. When the server receives an http request, it sends the corresponding update file.

5.6.2 Configuration Overview

The devkit runs Raspbian 9, a Linux-based operating system, and is configured to act as an update server. This host was used to test approved internet interactions related to MUD capabilities.

5.6.2.1 Network Configuration

The web server host has a static public IP address configuration and is connected to the access point on the wired interface. It is given an address on the 203.0.113 network.

5.6.2.2 Software Configuration

The Raspberry Pi is configured on Raspbian. The devkit also utilized a simple Python script to run an http server to test MUD capabilities.

5.6.2.3 Hardware Configuration

The hardware used for this devkit includes a Raspberry Pi 3 Model B.

5.6.3 Setup

The primary configuration needed for the web server device is done with the DNS mapping on the Zodiac WX access point to be discussed in the section related to setup of the Northbound Networks Zodiac WX Access Point. The Raspberry Pi is required to run a simple http server.

1. Copy the example Python script below onto the Raspberry Pi:

   Example Python script (`httpserver.py`):

```
import SimpleHTTPServer
import SocketServer
import argparse
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-H", help="Host address", default="0.0.0.0")
    parser.add_argument("-P", help="Port ", default="80")
    args = parser.parse_args()
    hostAddr = args.H
    PORT = int(args.P)
    Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
    httpd = SocketServer.TCPServer((hostAddr, PORT), Handler)
    print "serving at port", PORT
    httpd.serve_forever()
```

2. From the same directory as the script copied in the previous step, execute the command below to start the http server:

```
sudo python httpserver.py -P 443
```

```
www.nist.local ] sudo python httpserver.py -P 443
serving at port 443
```

5.7 Unapproved Server

5.7.1 Unapproved Server Overview

This section provides configuration details for the Linux-based IoT development kit used in the build, which acts as an unapproved internet host. This host will attempt to access and to be accessed by an IoT device, which, in this case, is one of the MUD-capable devices on the network.

The unapproved server is an internet host that is not explicitly authorized in the MUD file to communicate with the IoT device. When the IoT device attempts to connect to this server, the switch should not allow this traffic because it is not an approved internet service per the corresponding MUD file. Likewise, when the server attempts to connect to the IoT device, this traffic should be denied at the switch.

5.7.2 Configuration Overview

The devkit runs Raspbian 9, a Linux-based operating system, and is configured to act as an unapproved internet host. This host was used to test unapproved internet interactions related to MUD capabilities.

5.7.2.1 Network Configuration

The web host has a static public IP address configuration and is connected to the access point on the wired interface. It is given an address on the 203.0.113 network.

5.7.2.2 Software Configuration

The Raspberry Pi is configured on Raspbian. The devkit also utilized a simple Python script to run an http server to test MUD capabilities.

5.7.2.3 Hardware Configuration

The hardware used for this devkit includes a Raspberry Pi 3 Model B.

5.7.3 Setup

The primary configuration needed for the web server device is accomplished by the DNS mapping on the Zodiac WX access point to be discussed in the section related to setup of the Northbound Networks Zodiac WX Access Point. The Raspberry Pi is required to run a simple http server.

1. Copy the example Python script below onto the Raspberry Pi:

   Example Python script (`httpserver.py`):

   ```
   import SimpleHTTPServer
   import SocketServer
   import argparse
   if __name__ == "__main__":
       parser = argparse.ArgumentParser()
       parser.add_argument("-H", help="Host address", default="0.0.0.0")
       parser.add_argument("-P", help="Port ", default="80")
       args = parser.parse_args()
       hostAddr = args.H
       PORT = int(args.P)
       Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
       httpd = SocketServer.TCPServer((hostAddr, PORT), Handler)
       print "serving at port", PORT
       httpd.serve_forever()
   ```

2. From the same directory as the script copied in the previous step, execute the command below to start the http server:

   ```
   sudo python httpserver.py -P 443
   ```

   ```
   www.nist.local ] sudo python httpserver.py -P 443
   serving at port 443
   ```

Appendix A List of Acronyms

AAA	Authentication, Authorization, and Accounting
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.