POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Enhancing and testing smart home security through a MUD-enabled environment

Supervisors

Prof. Fulvio CORNO

Dott. Luca MANNELLA

Candidate

Fabio Orazio MIRTO

December 2023

To my parents, for their endless support.
To you, Deborah, for being fundamental in all the difficult moments of my life.
To the child who has always dreamed of this moment.
To me, for succeeding.

Ai miei genitori, per il loro interminabile sostegno.
A te, Deborah, per esser stata fondamentale in tutti momenti difficili della mia vita.
A quel bambino che ha sempre sognato questo momento.
A me, per esserci riuscito.

Summary

The increasing number of Internet of Things (IoT) devices in smart homes now offers a high level of efficiency and convenience that we have never seen before. However, this rapid growth also introduces some new challenges related to the security of these systems.

To address those challenges, IETF introduced Manufacturer Usage Description (MUD). MUD is a standard that offers a way to specify the network behavior and permissions of devices that are MUD-enabled. To achieve this goal, the MUD standard uses a white-listing approach based on a set of rules (also called policies). So, all traffic that is not expressly allowed by the manufacturer is blocked. These rules are read from a file, called MUD File, and instantiated for allowing connections between two end-points, for instance, using a firewall.

By offering better control over devices' interactions inside the smart home ecosystem, MUD provides a potential way to improve network security. This potential is also recognized by ENISA and NIST.

The main contribution of this thesis is to build and test a MUD-enabled smart home environment. In this scenario, there is a diverse set of IoT devices that should be managed by a Smart Home Gateway (SHG), in our case Home Assistant. By integrating the MUD standard inside the SHG, it is possible to achieve a higher level of network security and reduce the surface of attacks (e.g., avoiding unauthorized access or Denial of Service attacks).

In the context of IoT devices, the produced solution includes end-point identification for the production of the rules for the different integrations. This is done by performing a manual analysis of the traffic and source code of the integrations. After the initial setup, the proposed solution increases the security of the smart home without the need for further user action.

Acknowledgements

I have been searching for some time for the right words to write these thank-you notes. Now, having finished writing this paper, I think it is time to at least try.
Special thanks are due to Professor Fulvio Corno for his valuable advice and great support.
Another thank you undoubtedly goes to Dr. Luca Mannella for his patience and passion with which he followed this work, and for all the teachings given regarding the thesis and beyond.
Those who know me know how complicated it is for me to accept my success, but first, I would like to thank myself for never giving up all these years. It wasn't easy, but it was right to accomplish this goal.
That child, who already dreamt of a Master's Degree in Computer Engineering in primary school, can today say he succeeded. I hope he is happy about it.
My parents deserve my deepest appreciation. Their unwavering support and constant encouragement have been the rock on which I have built my path. Without them, nothing would have been possible.
A thank you also goes out to all the colleagues (many of whom have become friends over the years) I have met along the way and to those who, five years ago now, set out with me on this adventure. You have managed to dispel the loneliness that any out-of-town student feels during these experiences.
And finally, my thanks go to the girl who has put up with and supported me all these years, not only an engineering student (difficult as it is) but also a boyfriend more than 1500 km away for five long years.
Thanks Deborah, I know it wasn't easy, but watching the moon helped us.

Acknowledgements

È un po di tempo che cerco le parole giuste per scrivere questi ringraziamenti. Ora, dopo aver terminato la scrittura di questo elaborato, penso sia giunto il momento di provarci.
Un ringraziamento speciale va fatto al Professor Fulvio Corno per i suoi preziosi consigli e il grande supporto.
Un altro grazie va indubbiamente al dottor Luca Mannella per la pazienza e la passione con il quale ha seguito questo lavoro, per tutti gli insegnamenti dati riguardo la tesi e non solo.
Chi mi conosce sa quanto per me sia complicato accettare un mio successo, ma per prima cosa vorrei ringraziare me stesso per non aver mai mollato in tutti questi anni. Non è stato facile, ma era giusto portare a termine questo obiettivo.
Quel bambino, che già alle scuole elementari sognava una Laurea Magistrale in Ingegneria Informatica, oggi può dire di esserci riuscito, spero ne sia felice.
Ai miei genitori va il mio più profondo apprezzamento. Il loro sostegno incrollabile e l'incoraggiamento costante sono stati la roccia su cui ho costruito il mio percorso. Senza di loro, nulla sarebbe stato possible.
Un ringraziamento va anche a tutti i colleghi (molti dei quali diventati amici durante questi anni) che ho incontrato durante il percorso e a coloro che, ormai 5 anni fa, sono partiti con me per questa avventura. Siete riusciti ad allontanare la solitudine che qualunque studente fuorisede prova durante queste esperienze.
E infine i miei ringraziamenti vanno alla ragazza che ha sopportato e supportato in tutti questi anni, non solo uno studente di ingegneria (già difficile di suo), ma anche un fidanzato distante più di 1500 km per 5 lunghi anni.
Grazie Deborah, so che non è stato facile, ma guardare la Luna ci ha aiutato.

Table of Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

Nowadays, Internet of Things (IoT) devices are an important part of people's daily lives. According to Transforma Insight statistics [1], there will be around 17 billion IoT-connected devices worldwide by 2024, and this number could double (around 34.7 billion) by 2032.

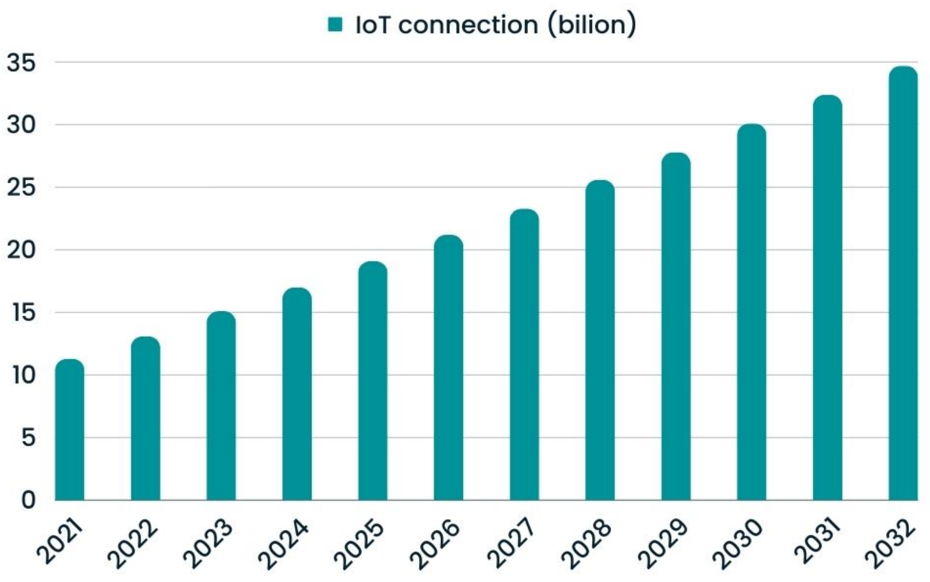


Figure 1.1: IoT Connections Forecast 2021-2032

To better understand these numbers, we need to provide some context. IoT devices refer to a vast network of interconnected physical devices and sensors to share and collect data. These devices can be used for various purposes across different domains. For example:

- Monitoring and Remote Control
- Data Collection and Analysis
- Automation and Efficiency

Based on the fact that the network is not always secure, all of these devices that are connected to it are exposed to various forms of cyber attacks. In the environment of IoT devices, it is possible to consider the fact that if some malicious code infects one device of a smart home, this code can be spread to all other devices connected to the same network. In the context of IoT devices, it is possible to have different types of attacks. Some of these attacks target the devices inside the network. Other attacks, instead, target remote endpoints. An example could be the Mirai Botnet [2], which is a notorious example of an IoT botnet responsible for one of the most significant DDoS attacks performed in 2016.

To address this challenge, IETF introduced Manufacturer Usage Description (MUD), a standard that permits specifying the network behavior and permissions of IoT devices.
Since it is possible to interconnect different IoT devices, it is easier to have something that can manage all the IoT devices in one place, like a Smart Home Gateway. A Smart Home Gateway is a central hub that connects and manages all Internet of Things devices in a smart home. It enables communication between devices and the internet, as well as with the user's smartphone or computer.

This thesis implements and deeply tests a physical proof of concept of a recently proposed architecture based on MUD and an SHG. The conducted tests demonstrate the feasibility of the approach and the benefits that this architecture can bring to smart home security.
In this thesis, we used an open-source home automation platform called Home Assistant that allows us to control, manage, and automate different IoT devices from different manufacturers and technologies through integrations.

Integrations are a fundamental part of Home Assistant, as they are used to support different devices from a wide range of manufacturers, not only devices that communicate via Wi-Fi but also devices that communicate via different IoT protocols (e.g., Zigbee, Z-Wave, MQTT).

By combining Home Assistant with the MUD standard, it was developed a MUD Aggregator [3] that is responsible for aggregating and exposing the MUD file, a file in which all the endpoints are needed for a specific (or for a group) IoT device.

With this setup, it is possible to establish a setting where a singular device can interact solely with trusted endpoints. This allows for preventing all feasible malicious connections that could potentially infiltrate the network.

After we have built the setup and deeply tested it, it is possible to notice that the experimental results obtained demonstrate the feasibility of the approach.

The chapters of this thesis are structured as follows:

- Chapter 2 - Background: in this chapter, there is a brief introduction to some background concepts, such as MUD, osMUD, SHGs, Home Assistant, and Wireless IoT Protocols.
- Chapter 3 - MUD integration: the description of a dedicated integration for the Manufacturer Usage Description (MUD) standard within the home automation platform, Home Assistant, is the main topic of discussion in this chapter. It gives a comprehensive overview of the essential ideas, approaches, and conclusions connected to this integration, emphasizing its role in enhancing network security in a smart home setting.
- Chapter 4 - Experimental setup: this chapter begins with a description of the experimental setup, followed by an exploration of the study's goals and the methodologies employed.
- Chapter 5 - Experiment results: in this chapter, it is possible to find the results of the experimental tests that were performed on the system.
- Chapter 6 - Conclusion: it highlights the thesis's contributions, the possible limitations of this setup, and chances for additional advancement in this study area.

Chapter 2

Background

To better understand the content of this thesis, it is important to introduce some background concepts. Firstly, we introduce the context of IoT devices and some protocols they use, such as Z-Wave [4], ZigBee [5], and KNX [6]. Then, we continue with the core of our research: Manufacturer Usage Description (MUD), the standard that we follow to enhance the security of our "smart home" created in the laboratory. Then, we briefly discuss osMUD, the MUD Manager (a component that retrieves the MUD specifications of a particular device) that we use to instantiate the rules starting from a file containing the rules, and OpenWRT, an open-source project that provides a Linux-based operating system and firmware for embedded devices, particularly routers and other networking equipment that work with osMUD. After this, we introduce the Smart Home Gateways, particularly Home Assistant, which can manage different IoT devices through its integrations.

2.1 Internet of Things and Smart Homes

Internet of Things, or IoT, is a fast-expanding subject that has the potential to transform a wide range of businesses completely. One of the most potential uses of IoT is in smart homes.

The IoT is a group of physical items that can collect and share data because they are integrated with sensors, software, and network connections. Smart homes, industrial automation, transportation, and healthcare are just a few industries that employ IoT devices.

The primary IoT domains are:

- **Consumer**: applications like wearables, linked cars, and smart homes fall under this category.

- **Industrial**: this domain covers supply chain management, predictive maintenance, and manufacturing automation applications.

- **Transportation**: this area covers applications including smart parking, traffic control, and driverless cars.

- **Applications**: including medical gadgets, tailored treatment, and remote patient monitoring, are under the healthcare domain.

IoT devices are used in "smart homes" to automate processes and improve comfort, efficiency, and security. Several well-known examples of smart home technology include thermostats, security cameras, smart locks, and smart bulbs.

The following are a few advantages of smart homes:

- **Enhanced comfort**: amart homes may be configured to change lighting, temperature, and other settings on their own to make a more pleasant space.

- **Enhanced efficiency**: by automatically shutting off lights and appliances when not in use, smart homes can contribute to lower energy use.

- **Enhanced security**: to assist in warding off attackers, smart homes may be outfitted with security features like motion sensors, cameras, and door locks.

2.2 Wireless IoT protocols

In the smart home context, the connection and management of smart devices rely heavily on wireless communication protocols. Three well-known protocols: Z-Wave [4], Zigbee [5], and KNX [6]. Each with its distinctive characteristics, benefits, and use cases—will be examined in this chapter.

The Figure 2.1 shows the different TCP/IP stacks for the different protocols.

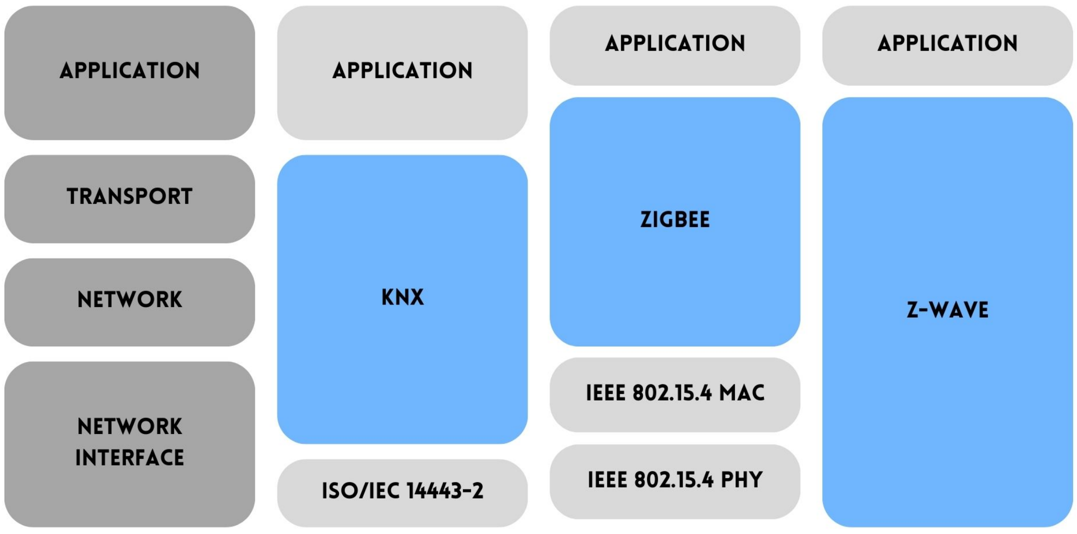


Figure 2.1: Stack of each protocol according to the TCP/IP Model

2.2.1 KNX

KNX [6], developed in 1991, is one of the main protocols in the Heating, Ventilation, and Air Conditioning (HVAC) industry, with a vast selection of compatible devices available. It was adopted as an open standard in 2006 by the ISO/IEC 14543-3 specification. KNX is based on the OSI [7] model and covers the data link, network, and transport layers.

The wireless transmission band is in the industrial, scientific, and medical (ISM) bands, specifically at 868 MHz and 2.4 GHz, with a maximum range of up to 150 meters. The maximum data rate transmission is up to 16.385 kbps. KNX nodes are connected in tree topologies: line, tree, and star (as Figure 2.2a). A KNX network is based on areas and lines. Each area can include up to 15 lines, where devices are connected as end nodes. The maximum number of addressable devices is 65,000, which can communicate with each other without a master device because KNX is a peer-to-peer system.

2.2.2 ZigBee

Zigbee [5] is a wireless communication protocol developed in 2001 and updated in 2007 with the Zigbee PRO specification, which is fully backward compatible and includes improvements such as better security. It is unique in that it implements concrete specifications for different scenarios. Two examples are Zigbee LightLink, widely used in smart lighting systems (e.g., Philips Hue), and Zigbee Green Power, which can work with battery-less devices similarly to EnOcean. Zigbee supports three different topologies: star (as Figure 2.2a), tree, and mesh (as Figure 2.3). It supports up to 65,000 nodes and operates in the "Industrial, Scientific, and Medical" (ISM) bands at 913 MHz, 868 MHz, and 2.4 GHz. The range of Zigbee devices is 10-100 meters, and the maximum data rate is 20 kbps for the 913 MHz and 868 MHz bands and 250 kbps for the 2.4 GHz band.

Zigbee uses the IEEE 802.15.4 standard [8] for the physical and data link layers. Zigbee is built on top of IEEE 802.15.4 and inherits some of its security vulnerabilities. Zigbee's security measures are implemented at the network (NKW) and application (APS) layers. Zigbee uses AES-CCM [9] for encryption and authentication, which is currently considered secure. Zigbee uses a message integrity code (MIC) and a frame counter for integrity and replay protection.

2.2.3 Z-Wave

Z-Wave [4] is a wireless communication protocol developed in 2001 for lightweight and low-latency data transmission. The latest version, called Z-Wave Plus [10], was released in 2013 and offers improved battery life and wireless range. Z-Wave is a mesh network protocol (as Figure 2.3), meaning that each device can act as a repeater to extend the range of the network. It supports up to 232 connected devices. This protocol uses the 828 MHz frequency in the European Union and the 908 MHz frequency in the United States and other markets. It has a maximum data rate of 100 kbps.

Z-Wave is not a standard, but its development is controlled by the Z-Wave Alliance [1], which includes over 600 companies, including major IoT players like Siemens and Huawei. Z-Wave provides confidentiality, authentication, and replay attack protection using a dedicated Security Layer within its Security Command Classes.

[1]`https://z-wavealliance.org/`, last visited on November 10th, 2023.

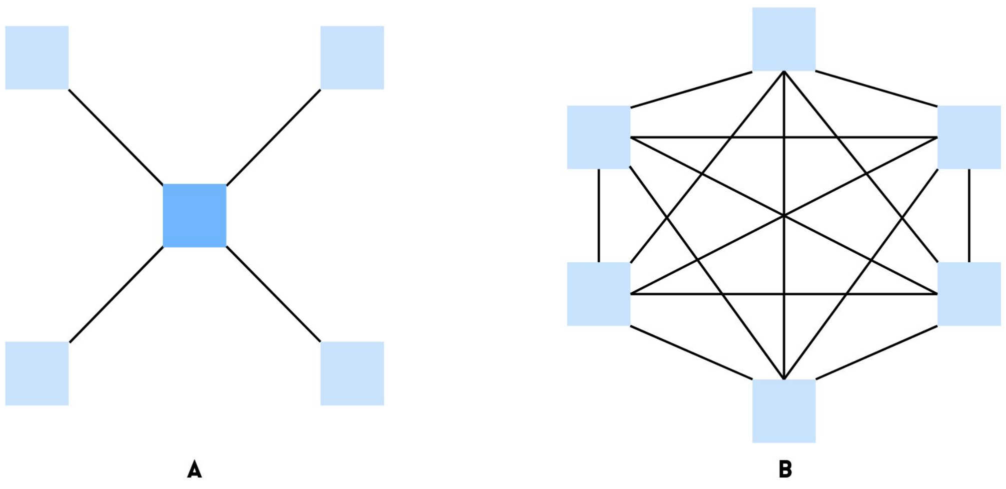


Figure 2.2: Network topologies: a) Star b) Fully connected

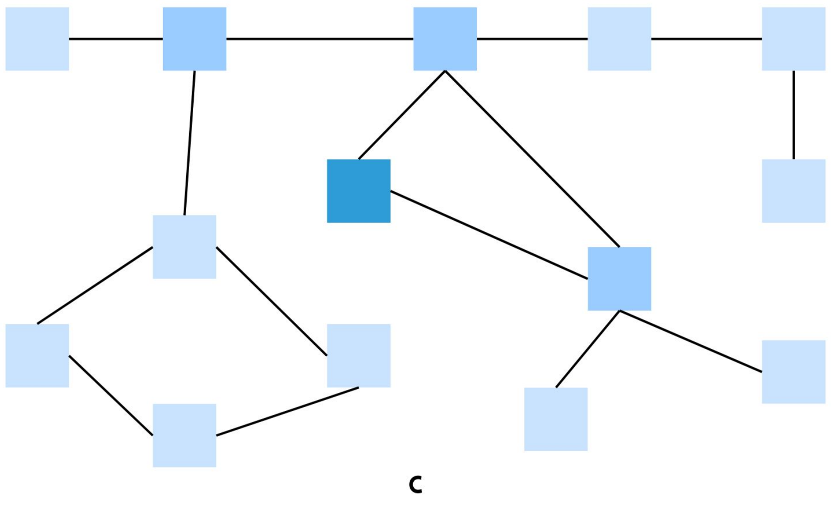


Figure 2.3: Network topologies: c) Mesh

2.3 Manufacturer Usage Description (MUD)

In the modern world, where everything is connected through networks, ensuring security has become a top priority for individuals, businesses, and organizations. The introduction of Internet of Things (IoT) devices has brought new challenges in managing and securing networks, making it more difficult to safeguard against potential threats. Manufacturer Usage Description (MUD) [11] appears to be a potentially effective approach to handling some of these difficulties. This chapter discusses the idea of MUD and how it might improve network security.

First of all, it is important to summarize the primary object of MUD:

- **Minimize Device Exposure**: By allowing only communications consistent with the manufacturer's intended usage, MUD aims to decrease the exposure of IoT devices to potential risks.

- **Scalability**: It offers a scalable approach to handle the increasing diversity of devices within a network, facilitating effective policy administration for different device kinds.

- **Cost-Efficiency**: MUD aims to keep the cost of implementing such a system as low as possible, guaranteeing its applicability to manufacturers and network administrators.

MUD is composed of three key architectural elements:

- **URL**: A Uniform Resource Locators (URL) [12] is a reference point for locating the device description.

- **Description**: The device's intended behavior inside the network is specified in this description. It provides comprehensive information about the device and how it should be interpreted.

- **Access Method**: A means that allows local network management systems to retrieve the device.

These components cooperate to create the MUD architecture, which enables the secure and controllable integration of IoT devices inside networked environments.

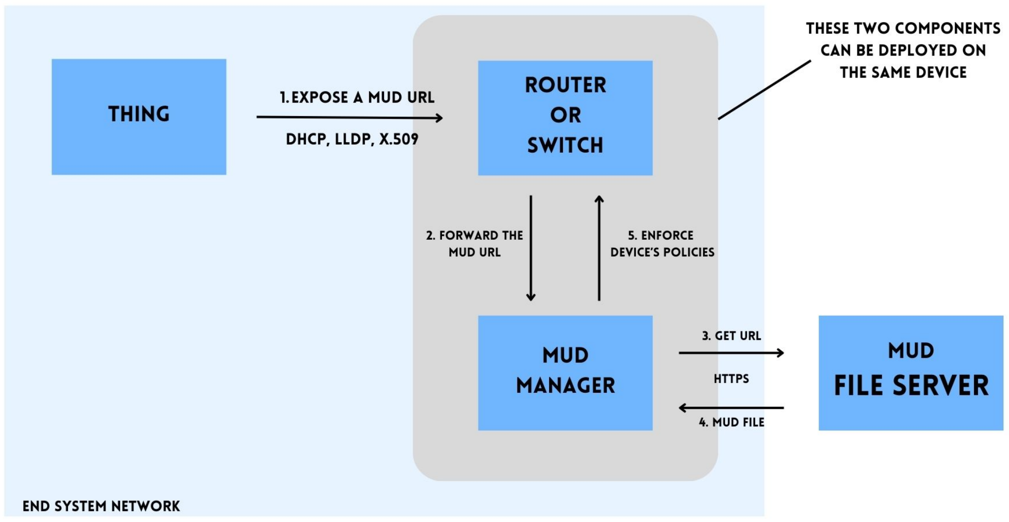


Figure 2.4: MUD architecture

2.3.1 Terminology

Understanding key terms and their definitions is essential to comprehending the Manufacturer Usage Description (MUD) structure and its components.

- **MUD File**: The file is a YANG-modeled [13] JSON [14] that describes an IoT device and provides network behavior recommendations.
- **MUD File Server**: A web server that stores and makes available MUD files
- **MUD Manager**: It is a software system that communicates with MUD servers to fetch and retrieve MUD files. Once the MUD file is processed, the MUD Manager can modify relevant network components to enforce the desired network behavior.
- **MUD URL**: A URL that the MUD manager uses as an address to get the MUD file linked to a certain Thing (IoT device).
- **Thing**: The IoT device that emits a MUD URL.
- **Manufacturer**: The organization configures the Thing to emit the MUD URL and assert advice in a MUD file. It's important to note that the company responsible for building the Thing may not always be the manufacturer. For example, it may be a supplier of components or a systems integrator.

2.3.2 MUD File Structure

The behavior and network needs of an Internet of Things (IoT) device are primarily described by a Manufacturer Usage Description (MUD) file, which is an essential part of the MUD standard. A specific IoT device, its manufacturer, and how it should operate while connected to a network are all covered in detail by the MUD file, which is a file containing YANG-based JSON. Inside the MUD file, it is possible to find some key elements:

1. **MUD Version**: The MUD specification being utilized is indicated by a version number frequently appearing at the start of a file.

2. **Device Specifications**:

    - **Device Name**: The IoT device's name or unique identification.
    - **Model Name**: The device's model name or number.
    - **Manufacturer**: The company that makes or sells the device.
    - **Documentation**: Links to the device's support and/or documentation materials
    - **System Info**: The user should be presented with a summary of this so they may decide whether to approve Thing's presence on the network

3. **Access Control Lists (ACLs)**: ACLs are user-ordered rule sets applied to networking devices to filter traffic. An Access Control Entry (ACE) represents each regulation.

4. **Access Control Entry (ACEs)**: ACEs are entries or rules that specify how a network should regulate and handle network traffic to and from a particular Internet of Things (IoT) device. These entries are an important part of the MUD file and are essential for the device's ability to enforce network security standards.

All the elements described are present inside the example of the MUD file present in the listing 2.1.

```
      {
         "ietf-mud:mud": {
           "mud-version": 1,
           "cache-validity": 24,
           "is-supported": true,
           "systeminfo": "This MUD file is generated by
   Home Assistant MUD Aggregator integration",
           "mfg-name": "e-Lite, PoliTo's Research Group",
           "documentation": "https://github.com/lucaMannella/HomeAssistant-MUD-Aggregator",
           "model-name": "home-assistant",
           "from-device-policy": {
             "access-lists": {
               "access-list": [
                 {
                   "name": "hass-mud-v4fr"
                 }
               ]
             }
           },
           "to-device-policy": {
             "access-lists": {
               "access-list": [
                 {
                   "name": "hass-mud-v4to"
                 }
               ]
             }
           }
         },
         "ietf-access-control-list:acls": {
           "acl": [
             {
               "name": "hass-mud-v4to",
               "type": "ipv4-acl-type",
               "aces": {
                 "ace": [
                   {
                     "name": "hass-cl0-todev",
                     "matches": {
                       "ipv4": {
                         "ietf-acldns:src-dnsname": "home-assistant.io"
                       }
                     },
                     "actions": {
                       "forwarding": "accept"
```

```
                      }
                    }
                  ]
                }
              },
              {
                "name": "hass-mud-v4fr",
                "type": "ipv4-acl-type",
                "aces": {
                  "ace": [
                    {
                      "name": "hass0-cl0-frdev",
                      "matches": {
                        "ipv4": {
                          "ietf-acldns:dst-dnsname": "home-assistant.io"
                        }
                      },
                      "actions": {
                        "forwarding": "accept"
                      }
                    }
                  ]
                }
              }
            ]
          }
        }

```

Listing 2.1: MUD file example

2.3.3 Exposing MUD URL

A device emitting a URL [12] is the first step in the process of using MUD URLs in our work. This URL provides access to a policy file that is related to that device.

To enable secure communication and the retrieval of policy information, MUD URLs must adhere to the Hypertext Transfer Protocol Secure (HTTPS) standard [15]. HTTPS is a secure version of HTTP [16]. It uses Transport Layer Security (TLS) [17] to guarantee encryption between a web server and a web browser. This means that your data is scrambled and cannot be read by anyone who intercepts it.

Different methods exist to expose the MUD file, such as:

- **DHCP [18] Option**: Devices can broadcast the MUD URL using a DHCP option (number 161 - name "OPTION_MUD_URL_V4"). The DHCP client sends the DHCP server this URL so it may act more, e.g., passing the MUD URL to the MUD manager to instantiate policies.
- **X.509 [19] Constraint**: It offers a certificate-based method for communicating device attributes based on IEEE 802.1AR [20].
- **Link Layer Discovery Protocol (LLDP) [21]**: It describes the usage of an LLDP frame as a method for devices to broadcast the MUD URL. As a result, devices can transmit their MUD URL information inside LLDP frames.

2.4 osMUD

The open-source Manufacturer Usage Description project [22], simply called osMUD, is a MUD Manager that aims to increase the security of networks and related devices. osMUD is presently made to work with dnsmasq and the OpenWRT firewall and operate on OpenWRT [23].

Its main role is implementing security regulations for IoT devices based on MUD files. A linked IoT device's MUD profile is first forwarded to the MUD Manager. It retrieves the MUD file where the device's network behavior is described, including which network resources and services the device should have access to and which it should not.

The MUD Manager analyzes the file after getting the device's MUD profile and knowing its identification. It examines the data included in the profile to establish what network activities the device is permitted to carry out. This comprises:

- **Authorized Services**: The MUD Manager lists the network services and assets, such as particular ports, protocols, or domains, that the device is permitted to access.
- **Limitations**: It also looks for any constraints the MUD profile may have set. To avoid security threats, it may, for instance, limit a device from accessing particular ports or services.

2.4.1 OpenWRT

OpenWrt [23] is an open-source operating system based on Linux created specifically for embedded devices, including wireless routers and networking devices. Its distinguishing quality is its adaptability, acting as a customizable and extendable platform. Utilizing the built-in stability and security of the Linux kernel [24], this router gives users improved control over their networking equipment, allowing them to customize their routers and network devices to match their specific requirements. It has a very important role in this case since osMUD is developed to work according to this firmware. Additionally, OpenWrt depends on a dedicated and engaged community of programmers and users, assuring ongoing development, support, and the accessibility of a constantly growing range of features and functions.

The documentation states that customers install OpenWrt because they think it performs better than the vendor's default firmware [2]. They discover it to be more reliable, more feature-rich, secure, and with superior support.

2.5 Smart Home Gateways

A SHG is a device or software solution that acts as a central hub for connecting and controlling smart home devices. It typically connects to the Internet and allows users to remotely control their smart home devices via a smartphone app or web interface. Smart home gateways also play an important role in automating smart home devices and managing their data.

Smart home gateways typically perform the following functions:

- **Connection**: Smart home gateways provide a central hub for connecting and controlling smart home devices. They support a variety of wireless protocols, such as Wi-Fi, Zigbee [5], and Z-Wave [4], allowing them to connect to a wide range of smart home devices from different manufacturers.

- **Control**: Through a web interface or smartphone app, SHGs enable consumers to remotely operate their smart home appliances. This allows users to turn lights on and off, adjust thermostats, lock and unlock doors, and more, even when they are not at home.

- **Automation**: Smart home gateways can be used to automate smart home devices. This allows users to create rules and schedules for their smart home devices to follow. For example, a user could create a rule to turn on the lights when they enter their home or to turn off the thermostat when they go to bed.

- **Data management**: Smart home gateways collect and manage data from smart home devices. This data can be used to track energy usage, identify patterns in device usage, and troubleshoot problems.

[2]`https://openwrt.org/#why_use_openwrt`, last visited on November 12th, 2023.

SHGs are an important component of smart home systems, but they also introduce new security and privacy risks.

To mitigate these risks, it is important to choose a smart home gateway from a reputable manufacturer and to keep the gateway's firmware up to date. Users should also be careful about what information they share with their smart home gateway and only install apps and plug-ins from trusted sources.

Smart home gateways can be used in various ways to improve home security, convenience, and energy efficiency. Here are a few examples:

- **Home security**: Smart home gateways can be used to create a comprehensive home security system. Users can connect smart door locks, security cameras, and motion sensors to their smart home gateway and receive alerts when there is suspicious activity in their home.

- **Convenience**: Smart home gateways can be used to automate smart home devices, saving users time and effort. For example, users can create a rule that turns on the lights when they get home or turns off the thermostat when they go to bed.

- **Energy efficiency**: Smart home gateways can be used to monitor and manage energy use in the home. Users can track which devices use the most energy and adjust their settings accordingly.

Smart home gateways are an essential part of modern smart home systems. As smart home technology continues to develop, smart home gateways are likely to become more powerful and sophisticated. For example, smart home gateways are expected to play an important role in the development of smart cities and the Internet of Things (IoT).

Smart home gateways are versatile devices that can be used to improve home security, convenience, and energy efficiency. However, it is important to be aware of the security and privacy risks associated with smart home gateways and to take steps to mitigate these risks.

In this chapter, we present the characteristics, strengths, and weaknesses of some of the most popular SHGs: OpenHAB and WebThings. Home Assistant will be introduced and presented in the next chapter since, after a deep analysis of the SHGs, we chose to use it for our environment.

2.5.1 OpenHAB

OpenHAB [25] is a free and open-source home automation platform that allows users to connect, control, and automate smart home devices from a variety of manufacturers. OpenHAB is a Java-based application that runs on a variety of platforms, including Raspberry Pi, Linux, and Windows.

OpenHAB is a popular choice for home automation enthusiasts because it is highly flexible and customizable. OpenHAB supports a wide range of smart home devices and protocols and can be used to create complex automation rules. In addition, OpenHAB is an open-source project, which means that there is a large community of users and developers contributing to the platform.

OpenHAB offers a wide range of home automation features, including:

- **Device support**: OpenHAB supports a wide range of smart home devices from different manufacturers. This includes devices such as lights, thermostats, locks, sensors, and more.

- **Protocol support**: OpenHAB supports a wide range of smart home protocols, including Zigbee [5], Z-Wave [4], and KNX [6]. This allows users to connect a wide range of smart home devices to OpenHAB, even if the devices use different protocols.

- **Automation**: OpenHAB allows users to create complex automation rules for their smart home devices. This allows users to automate their homes in a variety of ways, such as turning on the lights when they enter their homes or turning off the thermostat when they go to bed.

- **User Interface (UI)**: OpenHAB provides a web-based user interface for controlling and managing smart home devices. There are also some third-party mobile apps available for OpenHAB. In Figure 2.5, it is possible to see a custom page of OpenHAB of the web app. In this page, there are different IoT devices that can be managed (such as a smart light and a smart thermostat). It also shows the weather forecast.

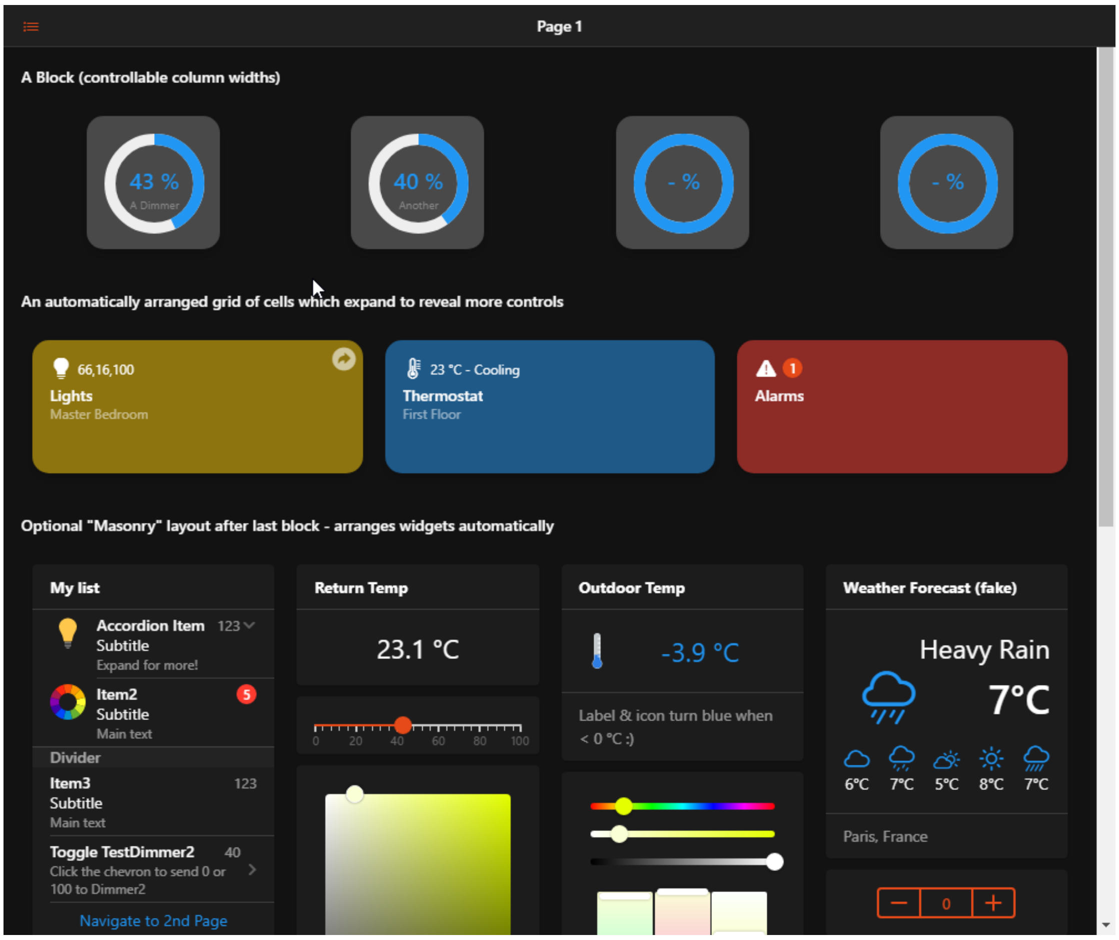


Figure 2.5: The User Interface of OpenHAB

- **Integration**: OpenHAB can be integrated with a number of other smart home platforms and services, such as Amazon Alexa, Google Assistant, and IFTTT. This allows users to control their OpenHAB devices using voice commands or other third-party apps.

OpenHAB can be used for a variety of home automation tasks, including

- **Lighting control**: OpenHAB can be used to control lights throughout the home. Users can create rules to turn lights on and off automatically or manually control lights using the OpenHAB user interface or mobile app.
- **Climate control**: OpenHAB can be used to control thermostats, fans, and other Heating, Ventilation, and Air Conditioning (HVAC) devices. Users can

create rules to adjust the temperature in their home automatically, or they can manually adjust the temperature using the OpenHAB user interface or mobile app.

- **Safety and security**: OpenHAB can be used to improve security and safety in the home. Users can connect smart locks, security cameras, and motion sensors to OpenHAB and receive alerts when there is suspicious activity in their homes.
- **Energy management**: OpenHAB can be used to monitor and manage energy consumption in the home. Users can track which devices are using the most energy and adjust their settings accordingly.

2.5.2 WebThings

WebThings [26] is a lightweight, open-source IoT platform that enables developers to build and deploy connected devices and applications. It is based on web standards and offers a number of features that make it ideal for developing IoT applications, including:

- **Security**: WebThings provides a range of security features to protect connected devices and applications, including encryption, authentication, and authorization.
- **Scalability**: WebThings is designed to scale to support large numbers of connected devices and applications.
- **Interoperability**: WebThings is interoperable with other IoT platforms and applications, making it easy to connect devices and applications from different vendors.

WebThings provides a number of features for developing IoT applications, including:

- **Device discovery and management**: WebThings provides an easy and efficient way to discover and manage connected devices.
- **Data streaming and analytics**: WebThings provides a way to stream data from connected devices and perform real-time analytics.
- **Event handling**: WebThings provides a way to handle events generated by connected devices.
- **Rules engine**: WebThings provides a rules engine that can be used to automate connected devices and applications.

- **User Interface (UI)**: WebThings provides a web-based user interface for managing connected devices and applications. In Figure 2.6 it is possible to see the dashboard of WebThings, it shows different IoT devices that can be managed by this page.

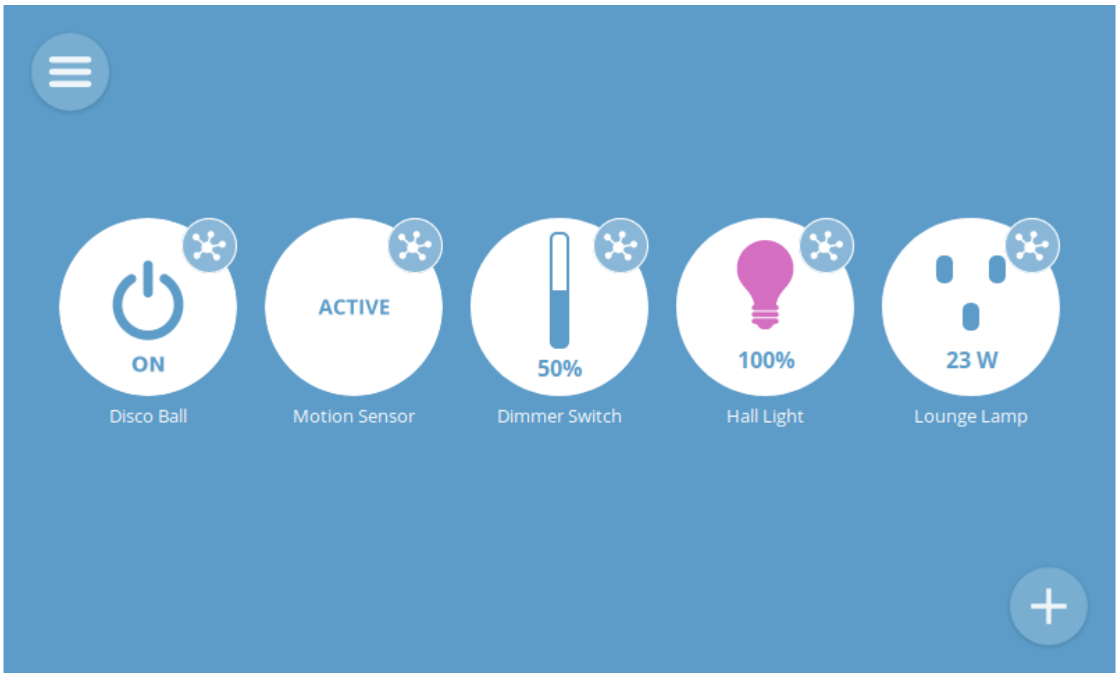


Figure 2.6: The User Interface of WebThings

- **APIs**: WebThings provides a set of APIs for developing custom IoT applications.

WebThings can be used for a variety of IoT applications, including:

- **Smart Home Automation**: WebThings can be used to develop smart home automation applications, such as turning lights on and off, controlling thermostats, and locking and unlocking doors.

- **Industrial automation**: WebThings can be used to develop industrial automation applications, such as monitoring and controlling industrial equipment.

- **Asset tracking**: WebThings can be used to develop asset-tracking applications, such as tracking the location of vehicles or shipping containers.

- **Environmental monitoring**: WebThings can be used to develop environmental monitoring applications, such as monitoring air quality or water quality.

Figure 2.7 represents the topography of a smart home. Through this section of the SHG, it is possible to see the state of the devices in each room. From this Figure, it is possible to monitor all the devices in different rooms.

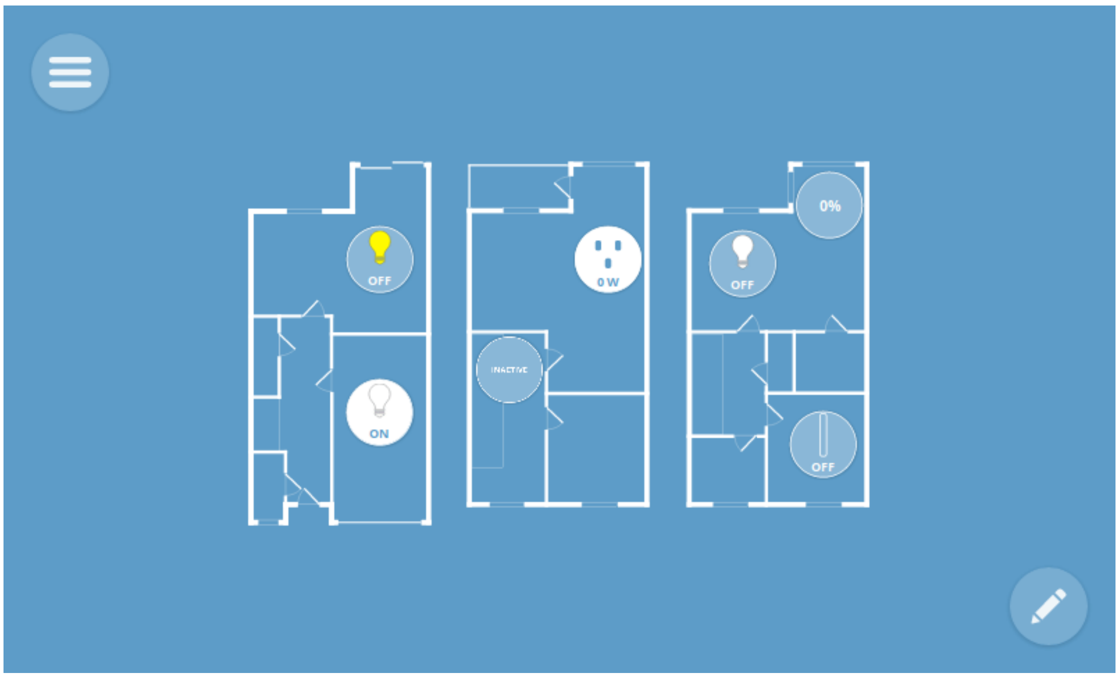


Figure 2.7: Topografy WebThings

Figure 2.8 shows all the changes in the status of different IoT devices, for example, a sensor (Hall Motion) that indicates if there is movement in a particular moment.

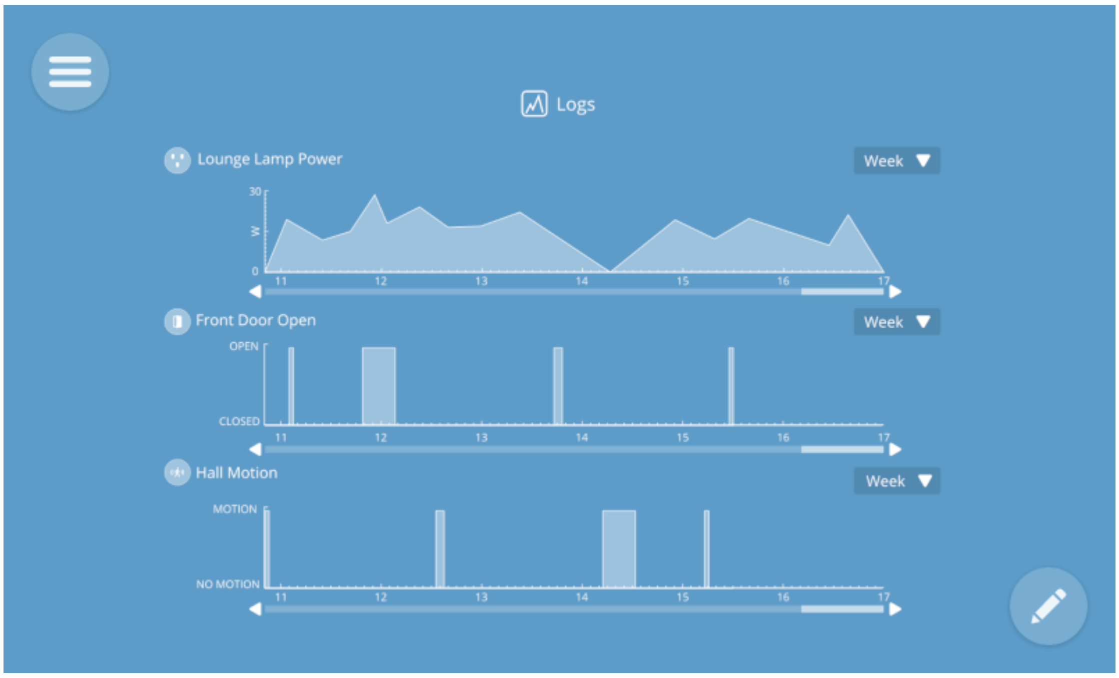


Figure 2.8: History of devices WebThings

WebThings is a powerful and flexible IoT platform that can be used to develop a wide range of IoT applications. It is a good choice for developers looking to build secure, scalable, and interoperable IoT applications.

2.6 Home Assistant

Home Assistant is an open-source home automation platform that offers a way to manage different IoT devices inside a smart home. Its strength points are:

- **Speed**: This is made possible by the fact that it works directly inside the home network and does not rely on external servers, as is common in cloud-based systems. This independence from external servers increases the system's stability while also enabling quick response times;
- **Consistency**: With this local focus, the smart home ecosystem's device interactions and automation are carried out with absolute precision, resulting in a user experience that is dependable and trustworthy;
- **Security**: Home Assistant is notable for its data integrity, which is especially valuable in light of increasing worries regarding data privacy and cyber risks.

In order to safeguard against external attackers, the policy of keeping data within the local network is stringently enforced.

The success of open-source solutions is based on these principles, each of which has a distinctive but interconnected significance.

- **Transparency**: The codebase of open-source software is transparent. This implies that anyone can check the code to make sure it lacks backdoors or hidden flaws. Users who are worried about the security of their smart home systems become more trustworthy as a result of this transparency.
- **Community Collaboration**: Open source initiatives like Home Assistant profit from a community of users and developers that work together to advance the project. This teamwork produces quick issue patches, feature upgrades, and general system stability.
- **Customization**: Open source software enables users to alter the code to suit their unique requirements. This implies that users of Home Assistant can develop customized integrations, automation, and add-ons that are suited to their particular smart home settings.

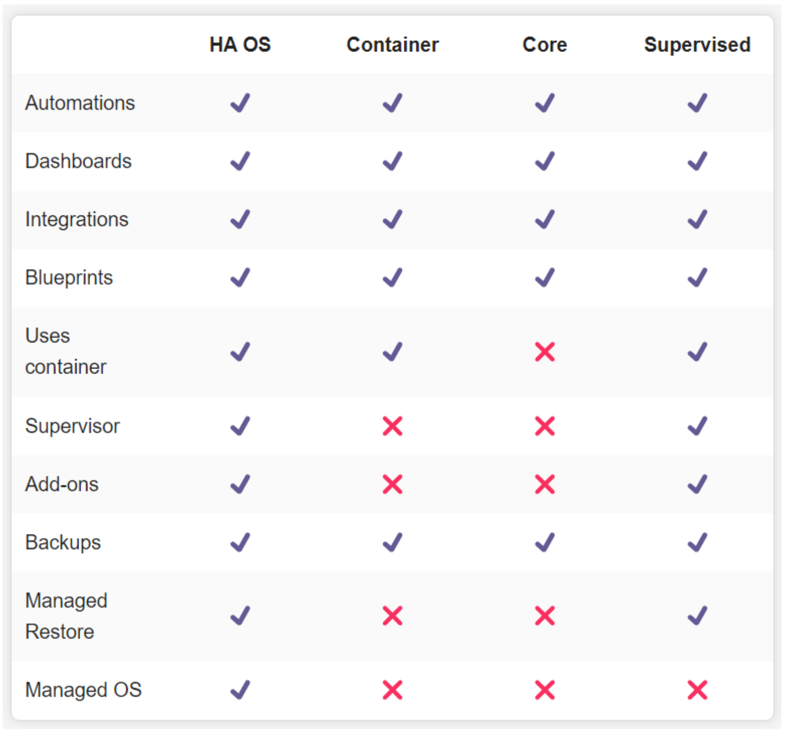
2.6.1 Home Assistant Installation

Home Assistant installation options offer users the flexibility to meet a range of needs. The Home Assistant Operating System and Home Assistant Container are the most popular and highly recommended options. The selection between these two alternatives depends on the specific use case and the required level of functionality.

- **Home Assistant Operating System**: this is the best option when a complete configuration is desired. Along with the essential Home Assistant program, this installation package also contains the useful Home Assistant Supervisor. The Home Assistant Supervisor is part of the Home Assistant ecosystem and is in charge of several tasks to guarantee the efficient management and functioning of the Home Assistant instance (e.g., Add-On and System Management, Backup and Restore, and Security).
- **Home Assistant Container**: it indicates a preference for a more simplified deployment within a container environment, which is frequently made possible by solutions like Docker. In some situations, this compact and versatile design might be especially helpful. There is a cost associated with it, though. Certain features, like the use of add-ons and the accessibility of the Home Assistant Supervisor, are either restricted or inaccessible when using this version.

It's important to remember that a Virtual Machine Image is another solution for utilizing the advantages of both Home Assistant versions. By using this method, the complete Home Assistant environment may be accessed from a single device, usually our computer. This results in the creation of a central hub where you can control and manage different IoT devices inside your smart home (through integration), and it is possible to have access to the add-ons. For users that demand both the span of the Home Assistant Operating System and the adaptability of containerization, this solution offers flexibility and scalability. This SHG can be installed on some physical devices such as Raspberry Pi or NUC. Home Assistant also provides some ready-to-go solutions like Home Assistant Yellow [27] or Green [28]. Both of them are pre-assembled Raspberry Pi with Home Assistant already installed in them.

In conclusion, Home Assistant provides users with the freedom to adapt their setup according to their requirements with various installation options, such as the comprehensive Home Assistant Operating System, the efficient Home Assistant Container, or a smart combination of both. This enables users to establish a secure and productive smart home environment.



	HA OS	Container	Core	Supervised
Automations	✓	✓	✓	✓
Dashboards	✓	✓	✓	✓
Integrations	✓	✓	✓	✓
Blueprints	✓	✓	✓	✓
Uses container	✓	✓	✗	✓
Supervisor	✓	✗	✗	✓
Add-ons	✓	✗	✗	✓
Backups	✓	✓	✓	✓
Managed Restore	✓	✗	✗	✓
Managed OS	✓	✗	✗	✗

Figure 2.9: Compare installation methods Home Assistant

2.6.2 Concepts and Terminology

Following a brief introduction to Home Assistant, it is now to understand a few fundamental concepts[3].

- **Dashboard**: It is a customizable part of the system, and it displays some pieces of information about the status of all the available devices. It is divided into two different parts: Overview and Energy, The Overview dashboard is the default landing page and gives a high-level overview of the key data and controls in your smart home. The Energy dashboard, on the other hand, is made to give customers information about their energy usage, enabling improved management and optimization of energy use in their smart homes.

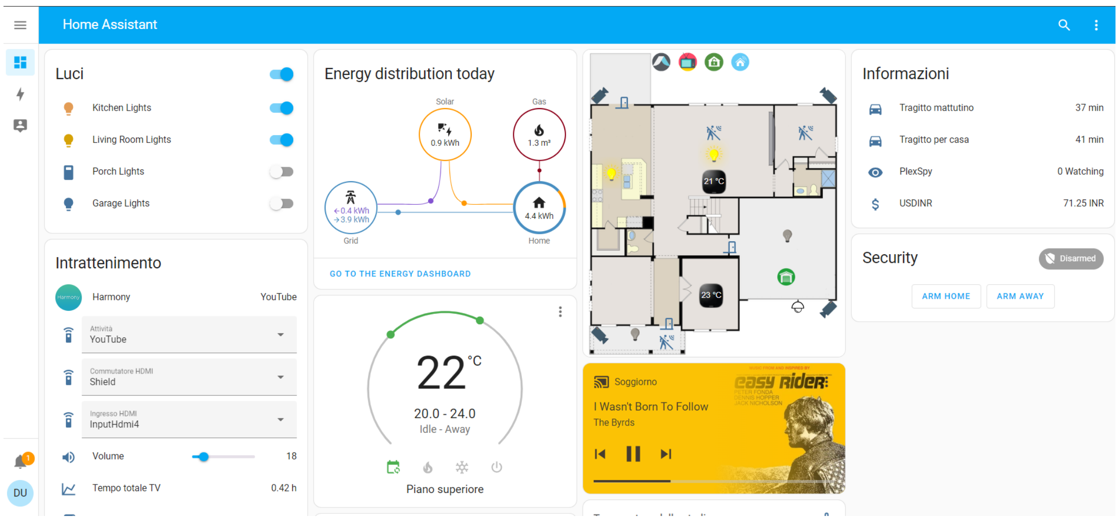


Figure 2.10: Demo Dashboard Home Assistant

- **Integrations**: As they allow the system to connect with and control a variety of smart devices, services, and platforms, Home Assistant integrations are an essential component of the Home Assistant platform. Home Assistant can interact and control a variety of IoT devices due to these integrations. Once an integration has been added, Home Assistant displays the hardware and/or data as devices and entities.

- **Devices and Entities**: In Home Assistant, a "device" commonly refers to a physical hardware unit (e.g., a smart light or a smart thermostat) that can be

[3]`https://www.homeassistant.io/getting-started/concepts-terminology/`, last visited on October 16th, 2023.

controlled and/or monitored from the dashboard. It is possible to initialize and then use these devices inside Home Assistant through the integrations. A component or feature of a device is referred to as an "entity". They are used to interact with and control the device's features. Entities can represent a device's numerous characteristics, like its temperature, humidity, power usage, on/off status, and more.

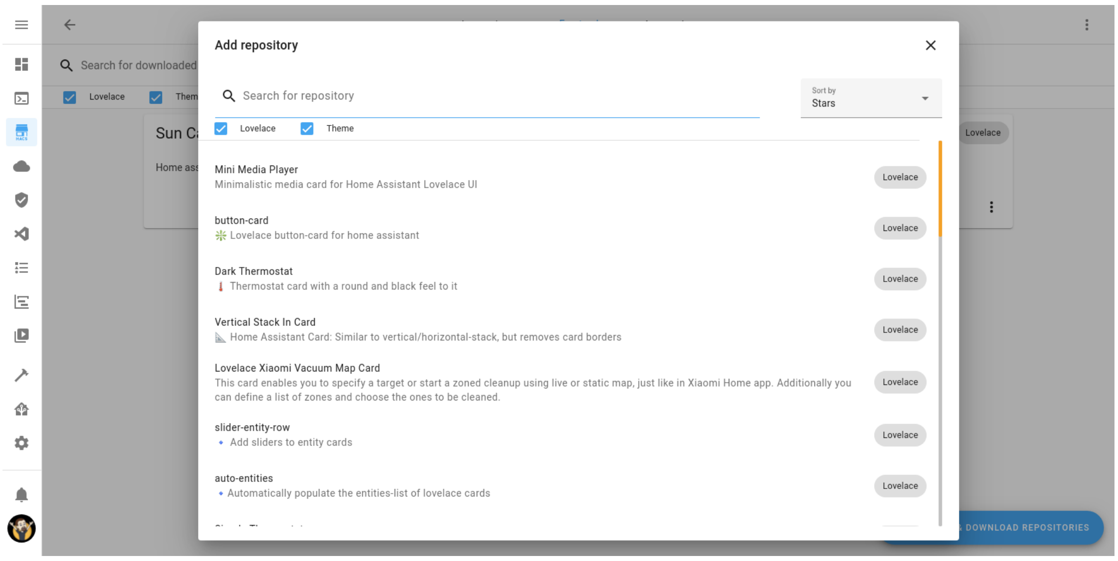
- **Automations**: With the help of Home Assistant Automations, it is possible to design unique rules and scenarios for your smart home. They enable you to automate actions and reactions based on specific triggers and conditions. In Home Assistant, an automation is a collection of rules that indicate what should happen in a certain situation. A trigger, conditions (if any), and actions normally make up an automation.

- **Scripts**: In Home Assistant, a script is similar to automation; repeating actions are a group of tasks that can all be carried out in response to a single command or trigger. It enables you to specify a series of steps that will be taken in response to different events or manually started activities. Scripts are collections of operations or service calls that communicate with your smart devices and services. Within a script, you can activate or deactivate devices, change their settings, and send notifications.

- **Scenes**: It is possible to easily create predefined configurations or "scenes" for the smart home using the feature called Home Assistant Scenes, which enables you to simultaneously capture and restore the states of several smart devices. Through the use of scenes, it is possibel to change the home's atmosphere, mood, or functionality with only one command or trigger.

- **Add-ons**: It is possible to add third-party add-ons with the complete installation of Home Assistant. These add-ons, typically, can be run inside Home Assistant and can be installed, configured, and run simply. These add-ons give Home Assistant additional features, utilities, and services, which increases its capabilities.

- **HACS**: Home Assistant Community Store (HACS) is a custom integration for Home Assistant. HACS allows users to install and manage custom integrations for Home Assistant easily, which are add-ons that extend the functionality of Home Assistant by adding support for new devices, protocols, and services. HACS is a valuable tool for Home Assistant users, as it allows them to access a wide range of custom integrations without manually installing and configuring them. It also provides some features that make it easy to manage custom integrations, such as the ability to automatically update integrations, disable integrations, and restore integrations to a previous version. The following picture represents the list of some custom integrations that can be added to Home Assistant from HACS.



Add repository

Search for repository

Sort by
Stars

Lovelace Theme

Mini Media Player
Minimalistic media card for Home Assistant Lovelace UI
Lovelace

button-card
Lovelace button-card for home assistant
Lovelace

Dark Thermostat
Thermostat card with a round and black feel to it
Lovelace

Vertical Stack In Card
Home Assistant Card: Similar to vertical/horizontal-stack, but removes card borders
Lovelace

Lovelace Xiaomi Vacuum Map Card
This card enables you to specify a target or start a zoned cleanup using live or static map, just like in Xiaomi Home app. Additionally you can define a list of zones and choose the ones to be cleaned.
Lovelace

slider-entity-row
Add sliders to entity cards
Lovelace

auto-entities
Automatically populate the entities-list of lovelace cards
Lovelace

Figure 2.11: HACS dashboard

Chapter 3

MUD Aggregator

Recently, an extended MUD architecture was proposed by the e-Lite research group [3]. This architecture is based on a SHG able to produce and expose a MUD file for all the plug-ins that the SHG manages. This architecture protects both software functionalities and physical devices with the MUD standard. This chapter describes this solution, as the SHG previously used does not support the MUD standard. With this approach, developers producing the integration for physical or virtual "devices" must also specify the necessary endpoints for enabling the correct communications. This also allows for safeguarding devices that are not MUD-enabled and all integrations on Home Assistant. In the following sections, we present two variations of the MUD integration. The first version is the one used to present this extended architecture. This has a problem since it does not check the integrity of the MUD snippet present in each folder of the integrations. For this reason, the second version is improved to perform the integrity check by the developer of the integration for the integrity of the folder.

3.1 Extended MUD architecture

The MUD standard aims to improve the security of IoT devices. This chapter presents the Extended MUD Architecture, which enables a single IoT device to expose its MUD file and allows an SHG to manage and enforce different rules for all the different plug-ins within a single MUD file. Various significant components exist within this architecture, beginning with OpenWRT, an open-source Linux system designed for routers, and osMUD, an open-source MUD manager integrated within the router. Additionally, osMUD supports the DHCP method for exposing the MUD file.

In this solution, Home Assistant played a crucial role as an SHG. Its responsibility, facilitated by the MUD integration, was to manage and assemble the MUD snippet of all integrations and expose the merged MUD file. The file is then stored within the web server of Home Assistant and sent to the router through a DHCP request within the MUD URL.

Once the router receives the DHCP request and the MUD manager verifies the signature, policy enforcement can begin. The acquisition of a fresh MUD file is triggered by a periodic check performed by the MUD Aggregator.

The employed OpenWRT firewall relies on iptables, and updates to the MUD file are performed for any currently available MUD-enabled integrations whenever users add new integrations to Home Assistant. The MUD Manager removes the previous file upon receiving the new one.

Cooperation among developers is crucial in the context of this MUD design based on a gateway architecture. For every integration they construct, developers need to establish MUD-compliant files, identify the required endpoints, and comply with the MUD standard.

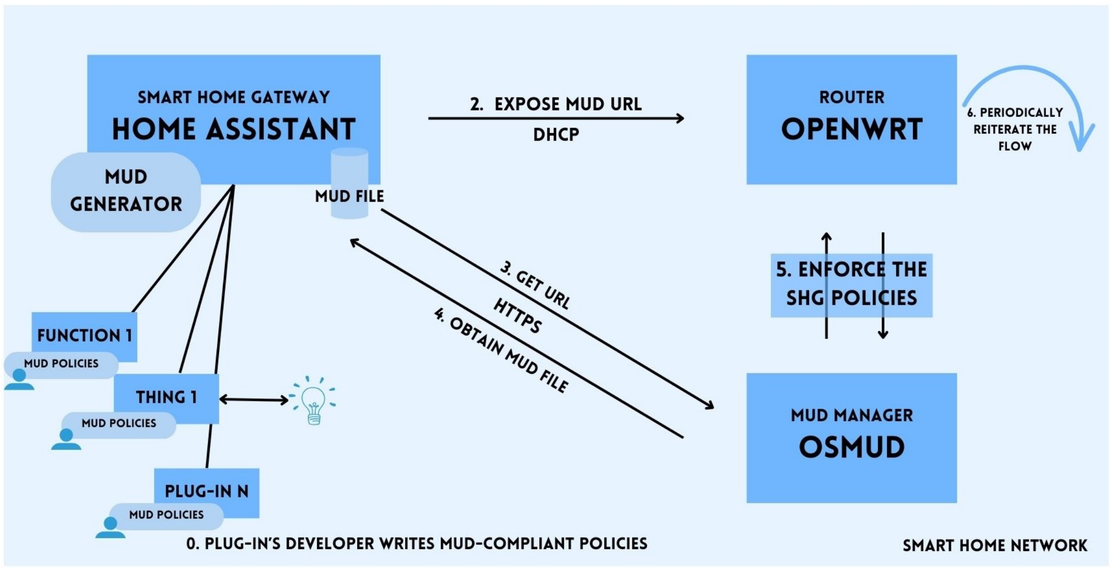


Figure 3.1: MUD architecture

It's worth noting that this design assumes that every smart home gateway plug-in, even if it only incorporates software functions, can benefit from MUD. This architecture enhances not only the security of the SHG but also the overall security of the smart home system, and the number of MUD-compliant devices may increase.

3.2 Standard MUD integration

Home Assistant does not come with built-in MUD compatibility. However, an integration called MUD Aggregator was created to overcome this shortcoming. Starting from a draft, the MUD Aggregator integration enables the creation of a MUD file. It searched through different folder integrations to find certain MUD "snippets" related to each. In each folder, it is possible to find a unique JSON file. Using a specified template, the MUD integration gathers these files and merges them into a local MUD file. The private key of the smart home gateway is then used to sign the created MUD file. When a file is signed, the MUD Aggregator alerts the MUD manager and puts it in a specified folder that is accessible via the Home Assistant web server.

3.2.1 Main software components

The MUD integration consists of several important parts. First, it scans all the integration folders available in the Home Assistant (custom and default integrations). Each folder contains the integration code and a file called “mud_gen.json”, which describes the requirements for each plug-in. This is a JSON file written using the YAML standard and is called “MUD snippet”. This file has all the properties of a simple MUD file.

From the Home Assistant point of view, this integration is a sensor that periodically scans all the folders searching for a MUD snippet. When it scans all of them and merges all the rules, it exposes the merged MUD file to our MUD manager using the DHCP method. In Listing 3.1, it is possible to see the `__init__` function of the sensor. The function `generate_mud_file` and `expose_mud_file` are contained in the MUDAggregator.py file (Section 3.2.1).

```
class MUDAggregatorSensor(SensorEntity):
    """ This sensor can recreate and expose the MUD file. """

    def __init__(self, params):
        self._mud_gen.generate_mud_file()
        self._mud_gen.expose_mud_file(self._interface)
```

Listing 3.1: Sensor’s `__init__` function

In the method `"update"`, the MUD file is periodically re-aggregated, and after the generation of the merged MUD file, it is exposed through the DHCP protocol.

```
    def update(self) -> None:
        """ Update method recreate the MUD periodically. """
        self._mud_gen.generate_mud_file()
        self._mud_gen.expose_mud_file(self._interface)
        _LOGGER.info("Automatically updating the MUD file!")
```

Listing 3.2: Sensor's `update` function

MUDAggregator.py functions

In this section, there is the analysis of some functions present inside the "MUDAggregator.py" file.

The `__init__` function initializes the MUD Aggregator object, loading the correct variable inside the `self` instance. This load is based on the environment in which the system is running. Since Home Assistant has different versions (Container or Operative System), some folder paths can be different.

In the `_generate_mud_file` function, the MUD file is generated starting from the mud_draft file that is a template. It has as input the `integration_list` (a dictionary containing the list of all integrations present inside Home Assistant). If the MUD file is changed, the timestamp inside it is changed, and the file will be signed with the certificate of the SHG using the `_write_mud_file` function.

```
def generate_mud_file(
            self, severity, security, integration_list, sign
    ) -> None:
        """This function generates a MUD file starting from a
    template"""
        self.security = security
        self._mud_draft = self._load_mud_draft()
        # (Re)loading original draft

        mud_file_path = self._LOCAL_EXTENTION_PATH + _MUD_FILENAME
        if not os.path.exists(mud_file_path):
            mud_changed = True
        else:
            with open(mud_file_path, "r") as inputfile:
                old_mud = json.load(inputfile)
                if "last-update" in old_mud["ietf-mud:mud"]:
                    # removing last-update timestamp if exists
                    del old_mud["ietf-mud:mud"]["last-update"]

                if self._mud_draft == old_mud:
                    mud_changed = False
                else:
                    mud_changed = True

        if mud_changed:
            current_time = datetime.now().isoformat(timespec="seconds")
            self._mud_draft["ietf-mud:mud"]["last-update"] = current_time
            self._write_mud_file(sign)
        else:
            _LOGGER.debug("MUD file was not changed!")
```

Listing 3.3: `generate_mud_file` function

The method `_add_mud_rules` is used to add the Access Control Lists (ACLs) to the MUD object to merge all the MUD snippets. It has also a counter that tells the user how many ACLs will be added to the MUD file.

In this version of the MUD integration, retrieving the MUD file by declaring its presence in the `manifest.json` file is possible.

```
def _add_mud_rules(self, integration_list: dict = None) -> int:
        """Adding the ACLs to the MUD object."""
        inserted_rules = 0
        if integration_list:
            inserted_rules += self._add_rules_from_manifest(integration_list)
        inserted_rules += self._add_rules_from_folders()
        _LOGGER.info("Total rules inserted in the MUD file: %d", inserted_rules)
        return inserted_rules
```

Listing 3.4: `_add_mud_rules` function

In this case, the `_add_rules_from_manifest` method adds the MUD snippet without searching it with a scan of the folder, but knowing a priori that the relative integration has it.

On the other hand, with the `_add_rules_from_folders`, all the MUD snippets present in Home Assistant are retrieved by scanning custom and default component directories. This method is designed for integrations that use the default name of the snippet. Whereas the previous method was added to allow developers to give whatever name they wanted to their snippet.

All the rules found in the MUD snippet are added to the MUD draft in the function `_add_rules_to_draft`. This method uses different functions (e.g. `_add_policies_if_not_exist` or/and `_add_acls_if_not_exist`) to check that the policies or the ACLs that want to add are not already present inside the MUD file.

After the file is signed, some other functions are responsible for exposing the MUD file. The most important one is `expose_mud_file`. This function, it is possible to select how to expose the MUD file. There are three possibilities: DHCP, LLDP, and 802.1AR. In our case, it is possible to expose it just with DHCP since our MUD manager does not support other methods.

3.3 MUD Aggregator with Notarization

The previous architecture had a limitation since the MUD snippet has no integrity checks. For this reason, a malicious user can modify it without anyone noticing. To reduce the attack surface, a solution was proposed that provided an authentication mechanism by performing the integration notarization process using the CodeNotary CAS service. This architecture ensures that the MUD integration uses all the trusted and authenticated MUD snippets to produce the gateway-level MUD file.

Before explaining the functioning of this solution, it is important to introduce CodeNotary CAS [29] and the notarization mechanism.

It is a blockchain-based [30] service used to guarantee the authenticity and integrity of software and/or digital content. It assisted in identifying any unauthorized alterations or modifications to the content by building a secure, time-stamped record of the digital content on the blockchain. The process that builds a record within the blockchain is called notarization.

Combining the MUD integration with the notarization is the main upgrade of this solution, proposed in the master thesis of Daniele Di Battista [31], since, with the previous approach, we do not have any guarantee about the integrity of the MUD snippets.

The notarization process of integration should be done by who developed the integration and the relative MUD snippet. This process has to follow different steps:

- The integration developer must first register into the CAS service to obtain the API-KEY and then log in.
- If he wants to notarize its integration, he should submit the source code of it and its MUD snippet to the CodeNotary CAS service.
- The CAS service, after calculating the cryptographic hash value of the submitted code, links this generated value with the integration to identify it.

After this process, it is comprehensible that someone modified the code if there is a mismatch between the integration's original hash value and the integration's hash value inside Home Assistant.

So, inside the MUD integration, there is this authenticity check on all the integrations in Home Assistant.

Since not all users have the same preferences, it presents a parameter that allows them to customize this security mechanism's severity level. It is possible to choose three different levels of severity.

- **High**: If the authenticity check fails, the integration is turned off inside Home Assistant, and its MUD snippet is not considered valid, so the rules are not merged inside the MUD file.
- **Medium**: If the authenticity check fails, the integration is not turned off inside Home Assistant, but its MUD snippet is not considered valid, so the rules are not merged inside the MUD file.
- **Low**: If the authenticity check fails, the integration is not disabled inside Home Assistant, and its MUD snippet is considered valid, so the rules are merged inside the MUD file.

Chapter 4

Experimental Setup

This chapter will discuss how the environment was built and the main activities performed to produce a secure testbed that can be compared with a real smart home. After the SHG and OpenWRT initialization combined with the MUD manager (osMUD), we installed different integrations. A MUD snippet was produced to define the permitted communications for each of those integrations.

4.1 Laboratory configuration and equipment

For building the experimental setup, we follow the extended MUD architecture. As described in the section 3.1, this solution allows a SHG to expose a single MUD file that contains inside all the MUD snippets for all the integrations present in Home Assistant.

First of all, we installed, on a Raspberry Pi Model 3 v1.2[1], OpenWRT [23] version 17.01.6 (described in section 2.4.1). osMUD [22] version 0.2.0, the MUD manager, was installed on it.

In the section 4.5, we will discuss how the MUD manager was modified to fix some bugs and to adapt to our use case.

On the second Raspberry Pi (same model as the previous one), there is the SHG, in our case, Home Assistant. Since we want to perform all the tests needed on the complete version of this SHG, we choose to install Home Assistant OS, described in Section 2.6.

It is composed by:

[1]`https://www.raspberrypi.com/products/raspberry-pi-3-model-b/`, last visited on October 10th, 2023.

- Home Assistant OS 10.3
- Home Assistant Core 2023.7.5
- Home Assistant Supervisor 2023.07.3

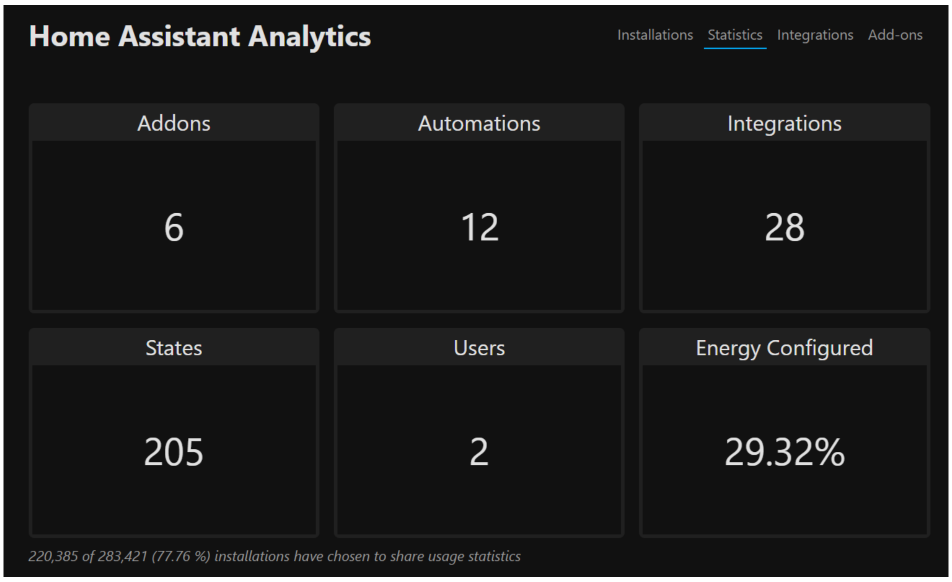
It also features a community store (HACS) in which every developer can upload its custom integration that other users can use.

This installation of Home Assistant permits to manage different standard devices (through standard integrations), install some integration from HACS (described in section 2.6.2, and also some add-ons that extend the functionalities of Home Assistant.

Following the analytics of Home Assistant [32], we aim to build a testbed that can be as similar as possible to a real smart world.

For this reason, we installed inside Home Assistant different integrations:

- 23 standard integrations: present on the core of Home Assistant
- 4 custom integrations: developed by myself in order to test some particular cases
- 5 integrations downloaded from HACS



Addons: 6
Automations: 12
Integrations: 28
States: 205
Users: 2
Energy Configured: 29.32%

220,385 of 283,421 (77.76 %) installations have chosen to share usage statistics

Figure 4.1: Home Assistant Analytics

Here is the list of some of the integrations and add-ons present in Home Assistant:

- **Bluetooth** [33]
- **CO2 Signal** [34]
- **Dipartimento Protezione Civile** [35]
- **File Editor** [36]
- **Github** [37]
- **Google Photos** [38]
- **Google Translate text-to-speech** [39]
- **HACS** [40]
- **HAM Radio Propagation** [41]
- **Home Assistant Supervisor** [42]
- **INGV Earthquakes** [43]
- **Input Boolean** [44]
- **Meteorologisk institutt (Met.no)** [45]
- **Mobile App** [46]
- **OpenAI Conversation** [47]
- **OpenWeatherMap** [48]
- **Philips Hue** [49]
- **Radio Browser** [50]
- **SSH & Web Terminal** [51]
- **Sun** [52]
- **Team Tracker** [53]
- **Timer** [54]
- **World's Air Quality Index** [55]
- **WorldTidesInfoCustom** [56]
- **Z-Wave** [57]

4.2 Methods for discovering integrations' endpoints

In order to write the MUD skeleton (the file where it is possible to find all the endpoints needed by Home Assistant for its start-up) and the MUD snippet, it was necessary to analyze the source code to find all the endpoints. After that, we performed manual network analysis, to confirm what was found inside the code and to verify that the endpoints are accessed at runtime.

In the following section, it is possible to find how this was done and the challenges found.

4.2.1 Source code

After the starting period, in which we focused on building an environment inside the laboratory, the second step was to write the MUD snippets for all the integrations that we previously installed in Home Assistant. We focused our attention on the source code present inside the Home Assistant Core repository on Github [58].

For example, for the CO2Signal integration, we found it inside the source code, and by looking at the documentation, we see that it queries the Electricity Maps API to retrieve all the data needed for the CO2 intensity in a specific place.

The API URL is inside the source code, so it was added to the MUD snippet of this integration.

This is an interesting case since the integration name is CO2signal, but the API used refers to another website. Through network analysis alone, it would have been not easy to notice this.

4.2.2 Network analysis

By doing some research, we found a tool [59], developed by NIST, that, starting from the capture file, generates the corresponding MUD file. As described by the authors, this tool is intended to assist the developers in the generation of MUD files to reduce the barrier to creating accurate files.

After some tests on this tool, we noticed that it works well with a capture file containing only a single device's traffic. Since our case is slightly different, it makes some errors during the generation. It cannot distinguish between two different integrations since it communicates with the internet through the same IP address (of the SHG).

For this reason, using automatic tools to discover the endpoints was not the best choice. The manual network analysis was first applied to the basic version of Home Assistant (without any integrations). Using tcpdump [60], a command-line tool used for packet analysis, it was performed a network traffic capture during the standard boot of Home Assistant, during its shutdown, and during the reboot. In addition, we left Home Assistant running for a long period and captured the packets sent. This allows us to analyze an important amount of data (about 500,000 packets) to discover the correct endpoints.

After this, it was necessary to write correctly the different MUD snippets. For this reason, we perform the network analysis on the SHG with all the integrations active simultaneously. In this way, collecting all the data needed to identify the correct endpoints for each integration was possible.

4.3 Manage custom endpoints in MUD files

Inside our testbed, we do not have only software integrations, but we also have some integrations that are used to manage a physical device.

A lot of IoT devices are connected to the network, so they have an IP address that can be public or local.

Since our main objective is to build a solution that is as replicable as possible, we need a way to standardize the MUD snippets with custom IP addresses.

We put a placeholder inside the MUD snippet of each of those integrations. The MUD Aggregator recognizes this placeholder and inserts all the endpoints that need to be replaced inside a JSON file.

In this case, the user must insert all the custom IP addresses of their physical devices during the initialization of the MUD integration.

The new endpoints inserted by the user will not modify the MUD snippet in the folder integration. However, the MUD integration inserts these IP addresses directly inside the gateway-level MUD file.

This is done to avoid invalidating the integrity check of the solution with Notarization described in Section 3.3. Indeed, any modification to a MUD snippet would invalidate the integrity verification process.

4.4 Reducing allowing communications

The MUD standard was proposed with the goal of blocking all communications that are not explicitly authorized. Since in our case, the MUD manager uses a standard firewall, for each device, a firewall rule is needed that “drop” all the communication between this particular device and the rest of the network.

Our use case is slightly different since the SHG exposes a single MUD file for all the possible integrations inside it.

For the integrations that manage a device with an IP address, it is necessary to add this rule to protect the device correctly. We are adding a protection mechanism that is consistent with what is specified in the MUD standard.

This is done using the JSON file that is filled, by the user, with all the custom IP addresses. In this way, we have a general solution that protects Home Assistant, its integrations, and all the physical devices.

4.5 Working on osMUD

The MUD manager used for building this environment was osMUD (described in the section 2.4). In order to adapt the existing code to our use case, we needed to improve something in the source code. For this reason, we changed the limited number of MUD files that the MUD manager can manage. Then, to fix some bugs already present in the source code, we correct the wrong translation of rules that contain the definition of a port range, and we also change an attribute to be compliant with the MUD RFC.

4.5.1 Port Range

During some tests on the instantiation of the firewall rules, we noticed that the firewall cannot translate correctly all the rules that contain a port range inside the MUD snippet.

In the following code, a part of a MUD snippet contains the key used to define a port range.

```
"aces": {
   "ace": [
     {
       {
         "name": "from-ipv4-switch-0",
         "matches": {
           "ipv4": {
             "protocol": 6,
             "ietf-acldns:dst-dnsname": "dcape-na.amazon.com"
           },
           "tcp": {
             "destination-port": {
               "operator": "range",
               "lower-port": 300,
               "upper-port": 500
             }
           }
         },
         "actions": {
           "forwarding": "accept"
         }
       }
     }
   ]
 }
```

Listing 4.1: Example of MUD file with Range of Ports

In the previous code, the range of the port is indicated in this way:

```
"lowerPort:upperPort"
```

This is a problem since, according to the documentation [2], the right definition of port ranges is the following:

```
"lowerPort-upperPort"
```

4.5.2 Limited number of manageable rules

Since there is a significant number of integrations in our SHG, the constant number of MUD files supported is not enough. In the previous version of osMUD, the maximum number of ACLs supported is 10, so, if in each file there are 2 ACLs (to and from), there are only 5 MUD files supported.

This is not enough since we have about 30 different MUD snippets for a total of 60 ACLs. Looking at the real world, some houses have hundreds of integrations, and in order for this proposed approach to work in real life, this change needs to be made to the official MUD Manager source code as well. We opened an issue to try and understand why this number was so low, which we did not get an answer to, and experimentally tried increasing it and found that it did not cause any problems for the MUD manager.

4.5.3 Compliance with RFC 8520

Another change made to the source code of osMUD was done to make the code compliant with the RFC of MUD[11].

The MUD standard should only include three different actions:

- **Accept**
- **Reject**
- **Drop**

In the original version of osMUD, the third option was "deny" and not "drop". This is not compliant with both the MUD standard [11] and the documentation of OpenWRT [61].

[2]`https://openwrt.org/docs/guide-user/firewall/firewall_configuration#rules/`, last visited on October 15th, 2023.

Chapter 5

Experiment results

This chapter presents the experiments conducted with our testbed. To build an environment as similar as possible to real installation, we consider all possible categories of integrations. Specifically, we identified the following four types of integrations:

- Software integrations
- IP-based devices' integrations
- Non-IP-based devices' integrations
- Mixed devices' integrations

In the first case, certain integrations can operate without a physical device. An example is the "MetNo integrations" [45]. It works because it obtains the weather forecast from a cloud.

The second example could be a smart light that may be turned on or off by speaking directly with the SHG via an IP address.

The ZWave integration is an example of the Non-IP-based integration. It communicates with devices via a wireless protocol, enabling easy integration without requiring an IP address. However, MUD is not a suitable technique to safeguard communications without an IP-based connection.

In the context of IoT devices, finding something that cannot be categorized in the previous categories is also possible. This is the case of the Philips Hue Bridge and Philips Hue smart light, which will be analyzed in section 5.5.

5.1 Experimental setup

In this section, it is possible to find the description of the environment produced in the Politecnico di Torino laboratory. The testbed is composed of two different

Raspberry Pi 3 Model B v1.2[1].

The first Raspberry was equipped with OpenWRT (version 17.01.6) and a customized version 0.2.0 of osMUD. Our osMUD was modified to manage a higher number of MUD snippets and to correctly manage the range ports option in the MUD file.

The second Raspberry contains a Home Assistant installation composed by:

- Home Assistant OS 10.3
- Home Assistant Core 2023.7.5
- Home Assistant Supervisor 2023.07.3

Inside Home Assistant, the following integrations were installed and configured:

- 23 standard integrations: present on the core of Home Assistant
- 4 custom integrations: developed by myself in order to test some particular cases
- 5 integrations downloaded from HACS

More details about all the integrations involved in the experimental setup were presented in Section 4.1.

5.2 Security of software integrations

Software integrations cover a range of functionalities, from the weather forecast to the "World Air Quality Index" to the "Team Tracker". As expected, various endpoints are required for each of these integrations to function.

In order to identify the endpoints required for the construction of the MUD snippets, an in-depth manual analysis of the traffic and source code of the integrations was conducted. After this network traffic analysis, it was possible to recognize different endpoints for each integration and build different MUD snippets.

5.3 Security of IP-based devices' integrations

An example of a device that is commonly IP-based is a smart light. This bulb has its IP address so the SHG can communicate directly with it to turn the light

[1] `https://www.raspberrypi.com/products/raspberry-pi-3-model-b/`, last visited on November 20th, 2023.

on/off, manage the brightness, and create some automation. When we write the MUD snippet, it is important to ensure that the light can communicate only with the SHG to reduce the attack surface and avoid unauthorized access.

Since our main objective is to produce a general MUD snippet, a limitation comes out. The IP address of a smart light is not the same in all the world's houses. So, the MUD snippet cannot be the same. In those cases, developers must insert a placeholder in the MUD snippet. During the initialization of the MUD integration, it requests the user to insert the IP address for all the physical devices.

These IP addresses are not directly inserted inside the MUD snippet since we will lose the integrity of it, but are inserted in the generated MUD file that is signed after all the custom IP addresses are communicated by the user.

With this approach, there is a possible limitation. The IP address of a smart device can change if it is rebooted due to the DHCP. In this case, the user has to update the related IP address.

To reduce the attack surface, blocking all possible connections between different devices and the external world is important. For this reason, all connections from the device to the internet (and vice versa) are blocked by a special rule.

Since the presence of placeholders would make the generated MUD file semantically incorrect, we allowed the user to decide whether to instantiate a partial MUD file or wait for the user to insert all custom IP addresses.

So, we implemented a parameter to select the level of protection that a user wants to achieve. With the highest security level, the MUD file is enforced, even if it is partial. With the lowest security level, the MUD file is generated and exposed only when the user fulfills all the placeholders.

Another limitation could be that, with this MUD Manager, all the devices inside the local network can still communicate with each other due to the fact that osMUD uses a firewall that blocks the traffic inside and outside the network and not inside the network.

5.4 Security of non-IP-based devices' integrations

In this scenario, it is possible to group all those integrations that communicate through all IoT protocols other than IP. This is the case of integrations that rely on devices based on different protocols like ZWave, Zigbee, and KNX. Since there is no connection based on IP, we cannot use the firewall as we intended. Therefore, the MUD can not be applied in this case.

5.5 Security of mixed devices' integrations

Inside our testbed, some devices are adopting a mixed approach. Specifically, they are a Philips Hue bridge and the associated smart light.

These devices have a mixed behavior since the bridge has an IP address, so it is reachable inside the network, while the smart light does not have an IP address and communicates with the bridge through the Zigbee protocol.

As described in Chapter 2, Zigbee is well-suited for IoT applications since it offers different advantages. In the particular case of the Philips Hue, the main advantage is that Zigbee supports mesh networking. A mesh network permits all the devices to communicate with each other directly, and they can also route data through other devices inside the network. This makes the Zigbee protocol very scalable and resilient to interference. From the security point of view, this protocol provides network security by protecting communications with AES-128-CCM encryption. Since it is designed for short-range communication, the mesh network is isolated from the internet and operates independently.

Since the communication between the smart light and the bridge is not IP-based, this part cannot be protected with the MUD standard. Meanwhile, with its IP address, the bridge is protected by enforcing the rules inside its MUD snippet.

5.6 Security of the SHG

The main objective of this thesis is to build an environment focused on the security of a SHG using the MUD standard. Since the architecture of our solution is different from the one presented in the RFC of the MUD, we notice that there is a time window in which there is no protection on the SHG.

During this window, from the boot of Home Assistant to the execution of the MUD integration, the protection given by the MUD is not yet enabled, so it is possible to connect without restriction to/from the internet. To reduce the window of exposure, we implement some setup rules directly on the router.

When the users want to execute the MUD manager, before turning on the SHG, they have to run a bash script that enforces all the necessary rules for the boot of Home Assistant (those rules are the same present in the MUD skeleton produced for Home Assistant, already described in Section 4.2) and starts the MUD manager.

Now, it is possible to turn on Home Assistant in a secure way. From some tests performed both with Home Assistant OS on a Virtual Machine and on a Raspberry Pi, when we enforce the rules before the boot, the start-up time is comparable with the start-up time without the rules. So, the MUD skeleton relative to Home Assistant permits the boot without slowing it down.

The only limitation of this solution is that all the integrations inside the SHG will not be started up during the first boot due to the firewall rules that we put in the router. Home Assistant will reload these integrations after enforcing the rules in the merged MUD file and after a timeout has elapsed. It is also possible to reboot the SHG after exposing the MUD file to speed up the process.

5.7 Testing the security of the SHG

In order to verify the security of the smart home gateway, we perform different tests on different areas of the environment. Table 5.1 summarizes all these tests and the expected outcome.

The tests are focused firstly on Home Assistant and the MUD integration for checking the correct management, assembly, and exposure of the gateway-level MUD file. It is possible to split two different categories of tests:

- **Generation and update of the MUD file**
- **Correct behavior of the network**

For the first category, we check the MUD integration's ability to generate a correct MUD file. It can correctly assemble a MUD file starting from different MUD snippets and expose it to the MUD manager. If a new integration's MUD snippet is added, the MUD integration will regenerate the MUD file after the configured time interval.

Discussing the tests performed about the correct behavior of the network, in our testbed, we have a smart light connected directly with its public IP address with Home Assistant through the manufacturer's integration. In order to work, in the MUD snippet of this integration, there is only the IP address of the light. During the test, we put, on purpose, a wrong IP address inside the MUD snippet, and we noticed how the SHG could not manage the smart light since the firewall blocks the connection between these two hosts. We also checked the correct functioning of the SHG and the correctness of the MUD skeleton. In the same way as the previous test, we put the wrong endpoints in the MUD skeleton. By measuring the boot time of Home Assistant, we noticed that the inability to access the correct endpoint will slow down the system boot, and some functionalities inside Home Assistant are unavailable.

Table 5.2 and 5.3 present the reboot times, the average time, and the standard deviation for our SHG under various conditions. The "Disabled" case refers to situations where the MUD integration is disabled, so no rules are present in the firewall. This is useful to compare the impact of this solution on the architecture. The "Setup" label indicates the reboot time when only the MUD skeleton rules are enforced. The “Complete” label represents the reboot time when the entire MUD file is integrated into the firewall after the user inserts its custom IP addresses.

It is possible to notice that the “Setup” average time is higher than the “Complete” average time. This is due to the fact that during the “Setup” phase, the Home Assistant (including all the integrations inside it) tries to boot itself, but in the firewall rules, there are only the endpoints needed by the SHG and not by the integrations. So, the integrations cannot access the endpoints they need and go into a timeout. The time is higher in the case of the Home Assistant OS because there are about 30 different integrations in it, whereas there are only 6 in the Home Assistant VM.

After that, another area of interest in our tests is the MUD Manager and the router. This helps us to verify the correct translation of the MUD file in firewall rules and the correct instantiation of the rules inside the firewall.

When the MUD integration exposed the MUD file, we checked the firewall inside the router and saw that the MUD Manager correctly translated and instantiated all the rules present in the generated MUD file.

In the case that the MUD file is changed, osMUD will remove the rules already present inside the firewall (without committing it), and only when it instantiates the new rules, if the process ends correctly, will it commit. Otherwise, it will roll back to the previous rules. With this behavior, we have a reduced window of exposure since the rules present in the firewall are only deleted after the correct installation of the new rules.

5.8 Common limitations

All the types of integrations have a common limitation. Since all those integrations have the same IP address (which is the address of the SHG), there is a possibility that an endpoint required by an integration also becomes accessible to others. This is because the requests for data are made directly from the IP address of the SHG, as these integrations are part of it.

Category	Test Summary	Expected Outcome
Home Assistant and the MUD integration	The MUD integration is used to generate a gateway-level MUD file starting from all the MUD snippets of the integration installed in Home Assistant	It is expected that the MUD file presents all the snippets
Home Assistant and the MUD integration	The MUD integration is also used to expose the MUD file to the MUD Manager through a DHCP request	It is expected that the router receives the DHCP request correctly with the MUD URL
Home Assistant and the MUD integration	A new integration is added to the SHG	In this case, the MUD integration has to regenerate and re-expose the MUD file after a configured time interval
Home Assistant and the MUD integration	The wrong IP address of a smart light is put inside the relative MUD snippet	The wrong MUD snippet does not permit the user to manage this device since it cannot reach the correct endpoint needed for the correct functioning
Home Assistant and the MUD integration	Some wrong endpoints are present inside the MUD skeleton	Home Assistant during its boot cannot find the correct endpoints, so some functionalities cannot be used
MUD Manager	After the generation of a gateway-level MUD file, the MUD integration exposes it to the MUD Manager	osMUD receives and instantiates correctly all the rules present inside the file
MUD Manager	If the MUD file is changed due to the addition of a new integration, for example, the MUD Manager has to update the firewall with new rules.	In case of an error during this process or if the MUD file contains some errors, the MUD Manager has to roll back to the previous state. Instead, if the MUD file is correct and there are no errors during the instantiation of the rules, those are committed to the firewall.

Table 5.1: Tests description

Firewall Rules	Fastest				Slowest	Average	Std.Dev.
Disabled	02:15	02:20	02:22	02:29	02:40	02:25	8.65 s
Setup	02:38	02:45	02:53	02:57	03:15	02:53	12.54 s
Complete	02:30	02:35	02:37	02:50	02:58	02:42	10.37 s

Table 5.2: Reboot time of Home Assistant VM in different conditions

Firewall Rules	Fastest				Slowest	Average	Std.Dev.
Disabled	04:32	04:34	04:49	04:52	05:32	04:50	21.77 s
Setup	09:15	09:45	10:27	10:58	12:53	10:33	169.26 s
Complete	04:26	04:28	04:57	05:29	06:01	05:01	37.33 s

Table 5.3: Reboot time of Home Assistant OS in different conditions

Chapter 6

Conclusion

This thesis implements and deeply tests a physical proof of concept of a recently proposed architecture based on MUD and an SHG. The testbed includes of two Raspberry Pi devices: the first running OpenWRT and osMUD (an open-source MUD manager), and the second running Home Assistant.

Within Home Assistant, I configured several integrations with some smart home devices. For each integration, we wrote a MUD snippet (containing the endpoints required for the integration) to describe the correct network behavior that the integration should have.

After processing the MUD snippets, the MUD Aggregator creates a gateway-level MUD file merging together the MUD snippets of all the integrations. In this case, there are no differences between software functionalities and physical devices since both are managed by Home Assistant through the use of integrations.

Since in the testbed there are also non-MUD-enabled physical devices, once the developer has created the snippet, we take care of protecting them with our solution. This is related to custom IP addresses, which are highly likely in a real smart home with different IP-based IoT devices. Using this architecture is the SHG that exposes the MUD file to the MUD Manager, so it was necessary to protect the physical devices (that have their IP addresses) from unauthorized communications.

In addition to the above, we investigated other aspects of the MUD, such as its performance and scalability. We conducted several experiments to measure the time required to process MUD snippets and generate the gateway-level MUD file. We also tested the enhanced version of the MUD manager to re-process the ability of the MUD manager to handle a large number of concurrent network requests.

Our results showed that MUD is a lightweight, scalable solution for securing smart home networks.

Thanks to the experimental testbed, we have demonstrated the feasibility of using MUD to secure smart home networks and smart home gateways. MUD snippets can be processed quickly, and the gateway-level MUD file can be generated efficiently.

6.1 Future work

6.1.1 Different MUD Manager architectures

The MUD manager adopted in this master thesis is osMUD. It is possible to notice that in the literature, other MUD Managers have different features and are based on different architectures:

- **Cisco MUD Manager [62]**: A RADIUS server uses the MUD Manager to convert a MUD URL into access control rules. After receiving REST APIs with the MUD URL (as well as perhaps other data), the MUD Manager provides RADIUS characteristics that may be transmitted to an Ethernet switch or other Network Access Device (NAD). By installing the policy on the access port, the NAD limits the device supplying the MUD URL to only the necessary network access.

- **NIST MUD Manager [63]**: In this case, OpenDaylight is used as the SDN controller for implementing MUD on switches that support SDN.

6.1.2 MUD Aggregator independent solution

As discussed in section 2.5, various Smart Home Gateways (SHGs) can be employed to facilitate communication between smart home devices. Within the scope of this master's thesis, we utilize Home Assistant to establish and evaluate an experimental setup, leveraging an existing plug-in known as MUD Aggregator. It facilitates the generation of MUD files. This approach provides a foundation for future advancements, potentially leading to developing an SHG-independent solution.

Home Assistant, a widely adopted open-source home automation platform, offers a versatile framework for building and testing smart home systems. Its extensive integration capabilities enable seamless interaction with various smart devices, protocols, and services. This makes it an ideal choice for implementing our experimental setup.

While the current setup relies on Home Assistant, future developments may lead to the creation of an SHG-independent solution. This would broaden the applicability of our approach beyond Home Assistant, allowing integration with other SHG platforms.

The potential benefits of an SHG-independent solution are that it would enhance the flexibility of our experimental setup, enabling compatibility with a wider range of SHGs.

The prospect of developing an SHG-independent solution can enhance the flexibility, interoperability, and adaptability of smart home systems.

Acronyms

API Application Programming Interface

DDoS Distributed Denial of Service

DHCP Dynamic Host Configuration Protocol

DoS Denial of Service

ENISA European Union Agency for Cybersecurity

HACS Home Assistant Community Store

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

JSON JavaScript Object Notation

LLDP Link Layer Discovery Protocol

MUD Manufacturer Usage Description

NIST National Institute of Standards and Technology

OS Operating System

RADIUS Remote Authentication Dial-In User Service

RFC Request for Comments

SDN Software-Defined Networking

SHG	Smart Home Gateway
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

Bibliography

[1] Transforma Insight. *IoT Connections Forecast 2021-2032*. URL: `https://transformainsights.com/research/forecast/highlights` (visited on Sept. 17, 2023) (cit. on p. 1).

[2] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. «DDoS in the IoT: Mirai and other botnets». In: *Computer* 50.7 (2017), pp. 80–84 (cit. on p. 2).

[3] Fulvio Corno and Luca Mannella. «A Gateway-based MUD Architecture to Enhance Smart Home Security». In: *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*. 2023, pp. 1–6. DOI: `10.23919/SpliTech58164.2023.10193747` (cit. on pp. 3, 28).

[4] Z-Wave Alliance. *Z-Wave*. URL: `https://www.z-wave.com/` (visited on Nov. 5, 2023) (cit. on pp. 4, 5, 7, 15, 17).

[5] Connectivity Standards Alliance. *Zigbee*. URL: `https://csa-iot.org/all-solutions/zigbee/` (visited on Nov. 5, 2023) (cit. on pp. 4–6, 15, 17).

[6] KNX.org. *KNX*. URL: `https://www.knx.org/knx-en/for-professionals/index.php` (visited on Nov. 5, 2023) (cit. on pp. 4–6, 17).

[7] *Open Systems Interconnection (OSI)*. URL: `https://www.iso.org/ics/35.100/x/` (visited on Nov. 2, 2023) (cit. on p. 6).

[8] «IEEE Standard for Low-Rate Wireless Networks». In: *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (2016), pp. 1–709. DOI: `10.1109/IEEESTD.2016.7460875` (cit. on p. 7).

[9] David McGrew and Daniel Bailey. *AES-CCM Cipher Suites for Transport Layer Security (TLS)*. RFC 6655. July 2012. DOI: `10.17487/RFC6655`. URL: `https://www.rfc-editor.org/info/rfc6655` (cit. on p. 7).

[10] Silicon Labs. *Z-Wave Specificationl*. URL: `https://docs.silabs.com/z-wave/latest/z-wave-docs/zwave-specification` (visited on Nov. 7, 2023) (cit. on p. 7).

[11] Ralph Droms Eliot Lear and Dan Romascanu. *Manufacturer Usage Description Specification*. March 2019. URL: https://www.rfc-editor.org/rfc/rfc8520 (cit. on pp. 9, 41).

[12] Larry M Masinter Tim Berners-Lee and Mark P. McCahill. *Uniform Resource Locators (URL)*. December 1994. URL: https://datatracker.ietf.org/doc/html/rfc1738 (cit. on pp. 9, 13).

[13] Martin Björklund. *The YANG 1.1 Data Modeling Language*. RFC 7950. Aug. 2016. DOI: 10.17487/RFC7950. URL: https://www.rfc-editor.org/info/rfc7950 (cit. on p. 10).

[14] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dec. 2017. DOI: 10.17487/RFC8259. URL: https://www.rfc-editor.org/info/rfc8259 (cit. on p. 10).

[15] Eric Rescorla. *HTTP Over TLS*. RFC 2818. May 2000. DOI: 10.17487/RFC2818. URL: https://www.rfc-editor.org/info/rfc2818 (cit. on p. 13).

[16] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999. DOI: 10.17487/RFC2616. URL: https://www.rfc-editor.org/info/rfc2616 (cit. on p. 13).

[17] Eric Rescorla and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/info/rfc5246 (cit. on p. 13).

[18] Droms Ralph. *Dynamic Host Configuration Protocol. RFC 2131*. March 1997. URL: https://datatracker.ietf.org/doc/html/rfc2131 (cit. on p. 14).

[19] Boeyen Sharon, Santesson Stefan, Polk Tim, Housley Russ, Farrell Stephen, and Cooper David. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. May 2008. URL: https://datatracker.ietf.org/doc/html/rfc5280 (cit. on p. 14).

[20] «IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity». In: *IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009)* (2018), pp. 1–73. DOI: 10.1109/IEEESTD.2018.8423794 (cit. on p. 14).

[21] «IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control». In: *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)* (2020), pp. 1–146. DOI: 10.1109/IEEESTD.2020.9018454 (cit. on p. 14).

[22] Yeich Kevin and Weller Daniel. *Open Source Manufacture Usage Description. [Online] Available.* 2022. URL: `https://osmud.org` (visited on Nov. 1, 2023) (cit. on pp. 14, 35).

[23] OpenWRT. *Documentation.* October 2023. URL: `https://openwrt.org/` (visited on Nov. 7, 2023) (cit. on pp. 14, 35).

[24] *The linux kernel.* URL: `https://docs.kernel.org/security/self-protection.html` (visited on Nov. 7, 2023) (cit. on p. 15).

[25] openHAB Foundation e.V. *openHAB documentation.* URL: `https://www.openhab.org/docs/` (visited on Nov. 5, 2023) (cit. on p. 17).

[26] Krellian Ltd. *WebThings documentation.* URL: `https://webthings.io/docs/` (visited on Nov. 5, 2023) (cit. on p. 19).

[27] Home Assistant. *Home assistant yellow.* URL: `https://www.home-assistant.io/yellow` (visited on Nov. 7, 2023) (cit. on p. 24).

[28] Home Assistant. *Home assistant green.* URL: `https://www.home-assistant.io/green` (visited on Nov. 7, 2023) (cit. on p. 24).

[29] CodeNotary. *CAS.* URL: `https://github.com/codenotary/cas` (visited on Sept. 7, 2023) (cit. on p. 33).

[30] Pinyaphat Tasatanattakool and Chian Techapanupreeda. «Blockchain: Challenges and applications». In: *2018 International Conference on Information Networking (ICOIN).* 2018, pp. 473–475. DOI: `10.1109/ICOIN.2018.8343163` (cit. on p. 33).

[31] Di Battista Daniele. *Ensuring integrity of MUD-enabled plug-ins for Smart Home Gateways. Master's thesis, Politecnico di Torino.* 2023. URL: `https://webthesis.biblio.polito.it/28006/` (cit. on p. 33).

[32] Home Assistant. *Analytics.* November 2023. URL: `https://analytics.home-assistant.io/statistics/` (visited on Nov. 25, 2023) (cit. on p. 36).

[33] Home Assistant. *Bluetooth integration.* URL: `https://www.home-assistant.io/integrations/bluetooth` (visited on Nov. 7, 2023) (cit. on p. 37).

[34] Electricity Maps. *Co2Signal integration.* URL: `https://www.home-assistant.io/integrations/co2signal/` (visited on Nov. 7, 2023) (cit. on p. 37).

[35] Protezione Civile. *DPC integration.* URL: `https://github.com/caiosweet/Home-Assistant-custom-components-DPC-Alert` (visited on Nov. 7, 2023) (cit. on p. 37).

[36] Home Assistant. *File Editor add-on.* URL: `https://github.com/home-assistant/addons/tree/master/configurator` (visited on Nov. 7, 2023) (cit. on p. 37).

[37] Home Assistant. *GitHub integration.* URL: https://www.home-assistant.io/integrations/github/ (visited on Nov. 7, 2023) (cit. on p. 37).

[38] Daan Sieben. *Google Photos HACS integration.* URL: https://github.com/Daanoz/ha-google-photos (visited on Nov. 7, 2023) (cit. on p. 37).

[39] Home Assistant. *Google Translate text-to-speech integration.* URL: https://www.home-assistant.io/integrations/google_translate (visited on Nov. 7, 2023) (cit. on p. 37).

[40] Home Assistant Community Store. *HACS.* URL: https://github.com/hacs/integration (visited on Nov. 7, 2023) (cit. on p. 37).

[41] Emiliano M. *HAM Radio Propagation HACS integration.* URL: https://github.com/emics/ham_radio_propagation (visited on Nov. 7, 2023) (cit. on p. 37).

[42] Home Assistant. *Home Assistant Supervisor integration.* URL: https://www.home-assistant.io/integrations/hassio (visited on Nov. 7, 2023) (cit. on p. 37).

[43] INGV. *INGV HACS integration.* URL: https://github.com/caiosweet/Home-Assistant-custom-components-INGV (visited on Nov. 7, 2023) (cit. on p. 37).

[44] Home Assistant. *Input Boolean integration.* URL: https://www.home-assistant.io/integrations/input_boolean (visited on Nov. 7, 2023) (cit. on p. 37).

[45] Home Assistant. *Meteorologisk institutt (Met.no) integration.* URL: https://www.home-assistant.io/integrations/met (visited on Nov. 7, 2023) (cit. on pp. 37, 42).

[46] Home Assistant. *Mobile App integration.* URL: https://www.home-assistant.io/integrations/mobile_app (visited on Nov. 7, 2023) (cit. on p. 37).

[47] Home Assistant. *OpenAI Conversations integration.* URL: https://www.home-assistant.io/integrations/openai_conversation/ (visited on Nov. 7, 2023) (cit. on p. 37).

[48] Home Assistant. *OpenWeatherMap integration.* URL: https://www.home-assistant.io/integrations/openweathermap/ (visited on Nov. 7, 2023) (cit. on p. 37).

[49] Home Assistant. *Philips Hue integration.* URL: https://www.home-assistant.io/integrations/hue/ (visited on Nov. 7, 2023) (cit. on p. 37).

[50] Home Assistant. *Radio Browser integration.* URL: https://www.home-assistant.io/integrations/radio_browser (visited on Nov. 7, 2023) (cit. on p. 37).

[51] Franck Nijhof. *SSH & Web Terminal*. URL: `https://github.com/hassio-addons/repository/blob/master/ssh/DOCS.md` (visited on Nov. 7, 2023) (cit. on p. 37).

[52] Home Assistant. *Sun integration*. URL: `https://www.home-assistant.io/integrations/sun` (visited on Nov. 7, 2023) (cit. on p. 37).

[53] TeamTracker. *Team Tracker HACS integration*. URL: `https://github.com/vasqued2/ha-teamtracker` (visited on Nov. 7, 2023) (cit. on p. 37).

[54] Home Assistant. *Timer integration*. URL: `https://www.home-assistant.io/integrations/timer` (visited on Nov. 7, 2023) (cit. on p. 37).

[55] Home Assistant. *World Air Quality Index (WAQI)*. URL: `https://www.home-assistant.io/integrations/waqi/` (visited on Nov. 7, 2023) (cit. on p. 37).

[56] WorldTidesInfoCustom. *WorldTidesInfoCustom HACS integration*. URL: `https://github.com/jugla/worldtidesinfocustom` (visited on Nov. 7, 2023) (cit. on p. 37).

[57] Home Assistant. *Z-Wave integration*. URL: `https://www.home-assistant.io/integrations/zwave_js` (visited on Nov. 7, 2023) (cit. on p. 37).

[58] Home Assistant. *Core*. October 2023. URL: `https://github.com/home-assistant/core` (cit. on p. 38).

[59] Watrobski P. and Klosterman J. *MUD-PD*. December 2021. URL: `https://github/ustnistgov/MUD-PD` (visited on Nov. 21, 2023) (cit. on p. 38).

[60] Jacobson Van, Leres Craig, and McCanne Steven. *tcpdump*. URL: `https://www.tcpdump.org/` (visited on Oct. 11, 2023) (cit. on p. 38).

[61] OpenWRT. *Documentation*. October 2023. URL: `https://openwrt.org/docs/guide-user/firewall/firewall_configuration#rules` (cit. on p. 41).

[62] CiscoDevNet. *MUD-Manager*. July 2021. URL: `https://developer.cisco.com/codeexchange/github/repo/CiscoDevNet/MUD-Manager/` (visited on Nov. 18, 2023) (cit. on p. 51).

[63] Nist. *MUD-Manager*. May 2020. URL: `https://github.com/usnistgov/nist-mud` (visited on Nov. 18, 2023) (cit. on p. 51).