

Kevin Nordnes

IoTective: Automated Penetration Testing for Smart Home Environments

Master's thesis in Information Security

Supervisor: Jia-Chun Lin

Co-supervisor: Ernst Gunnar Gran

June 2023

Kevin Nordnes

IoTective: Automated Penetration Testing for Smart Home Environments

Master's thesis in Information Security
Supervisor: Jia-Chun Lin
Co-supervisor: Ernst Gunnar Gran
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology



Abstract

The rapid proliferation of Internet of Things (IoT) devices has raised significant concerns regarding the security and privacy of interconnected smart environments. This thesis presents IoTective, an automated penetration testing tool designed to assess the security posture of IoT devices and systems. IoTective utilizes a range of scanning techniques, including Wi-Fi, Bluetooth, and ZigBee, to identify vulnerabilities, discover devices, and gather valuable information for analysis. The tool's intuitive user interface and automation capabilities minimize the need for manual intervention, making it accessible to both novice and experienced security analysts. Through our Proof-of-Concept (PoC), IoTective demonstrates its effectiveness in identifying vulnerabilities and enumerating a wide range of smart home devices and systems. The tool's regular updates, complementarity with manual testing, and prioritization features, contribute to its success as a comprehensive IoT security assessment tool. Future work includes expanding vendor-specific testing to incorporate support for additional manufacturers and their Application Programming Interfaces (APIs), allowing for more in-depth analysis and targeted vulnerability detection. With the continuous advancement of IoT technologies, IoTective's contributions to the field of IoT security assessment and its potential for further enhancement make it a valuable resource for securing IoT environments.

Sammendrag

Den raske utbredelsen av Internet of Things (IoT) har reist betydelige bekymringer angående sikkerheten og personvernet til sammenkoblede smarte miljøer. Denne masteroppgaven presenterer IoTective, et automatisert verktøy for penetrasjonstesting designet for å vurdere sikkerhetstilstanden til IoT-enheter og -systemer. IoTective benytter seg av ulike skanningsteknikker, inkludert Wi-Fi, Bluetooth og ZigBee, for å identifisere sårbarheter, oppdage enheter og samle verdifull informasjon for analyse. Verktøyets brukervennlige grensesnitt og automatiseringsfunksjoner minimerer behovet for manuell innblanding, noe som gjør det tilgjengelig både for nybegynnere og erfarne sikkerhetsanalytikere. Gjennom vår Proof-of-Concept (PoC) demonstrerer IoTective sin effektivitet i å identifisere sårbarheter i et bredt spekter av smarte hjemmeenheter og -systemer. Verktøyets regelmessige oppdateringer, komplementaritet med manuell testing og funksjoner for prioritering bidrar til suksessen som et omfattende verktøy for vurdering av IoT-sikkerhet. Fremtidig arbeid inkluderer utvidelse av testing for spesifikke produsenter for å inkorporere støtte for flere leverandørers APIs, slik at det blir mulig med grundigere analyser og målrettet sårbarhetsdeteksjon. Med den kontinuerlige utviklingen av IoT-teknologier, bidrar IoTectives bidrag til feltet for vurdering av IoT-sikkerhet og dets potensial for videre forbedring til å gjøre det til en verdifull ressurs for sikring av IoT-miljøer.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Jia-Chun Lin, for their invaluable guidance, support, and encouragement throughout the duration of this project. Their expertise, insights, and constructive feedback have been instrumental in shaping the direction and quality of this work. I am grateful for their patience, dedication, and mentorship, which have greatly contributed to my personal and professional growth. I would also like to extend my appreciation to my co-supervisor, Ernst Gunnar Gran, for their valuable input and assistance. I am thankful for their availability, collaboration, and willingness to share their knowledge, which have significantly enhanced the outcomes of this project.

Furthermore, I would like to acknowledge and thank NTNU, especially Department of Information Security and Communication Technology (IIK), for their support and resources that have facilitated the successful completion of this thesis. I am also grateful to my friends and fellow students for their encouragement, discussions, and assistance throughout this journey. Their support and camaraderie have made this experience more enjoyable and memorable.

I am indebted to all those who have contributed to this project in various ways. Without their support and collaboration, this work would not have been possible.

Contents

Abstract	iii
Sammendrag	v
Acknowledgments	vii
Contents	ix
Figures	xiii
Tables	xv
Code Listings	xvii
Acronyms	xix
1 Introduction	1
1.1 Project Background	1
1.2 Project Goals	2
1.3 Research Questions	2
1.4 Research Design	4
1.5 Ethical Considerations	4
1.6 Project Scope and Limitations	5
1.7 Thesis Structure	6
2 Background	7
2.1 Internet of Things (IoT)	7
2.1.1 IoT Devices	8
2.1.2 IoT Architecture	9
2.1.3 IoT Communication Technologies	10
2.1.4 Advantages of IoT	14
2.1.5 Challenges of IoT	15
2.1.6 IoT Security Requirements	16
2.2 Smart Homes	17
2.2.1 Smart Home Automation	18
2.2.2 Smart Home security issues	19
2.3 Penetration Testing	23
2.4 OWASP IoT Project	25
3 Related Work	27
3.1 IoT Vulnerabilities	27
3.1.1 ZigBee Vulnerabilities	27
3.1.2 Bluetooth Low Energy (BLE) Vulnerabilities	29
3.1.3 Vulnerabilities in IoT Products	31

- 3.2 IoT Penetration Testing Frameworks 32
 - 3.2.1 PENIOT 32
 - 3.2.2 EXPLIoT 33
 - 3.2.3 HomePwn 34
 - 3.2.4 KillerBee 34
 - 3.2.5 Comparison 34
- 4 System Design of IoTective 37**
 - 4.1 Design Considerations 37
 - 4.2 Requirements of IoTective 38
 - 4.2.1 Functional Requirements 38
 - 4.2.2 Non-Functional Requirements 39
 - 4.3 Architecture of IoTective 39
- 5 Implementation 43**
 - 5.1 Development Environment 43
 - 5.2 Python Modules 45
 - 5.3 Code Structure 46
 - 5.4 Main Functions 46
 - 5.4.1 Initialization 46
 - 5.4.2 Scanning 47
 - 5.4.3 Sniffing 49
 - 5.4.4 Reporting 51
- 6 Proof-of-Concept 53**
 - 6.1 Proof-of-Concept Environment 53
 - 6.2 Test Execution and Results 55
 - 6.2.1 Phase 1: Initialization 55
 - 6.2.2 Phase 2: Scanning 55
 - 6.2.3 Phase 3: Sniffing 58
 - 6.2.4 Phase 4: Reporting 59
 - 6.3 Analysis and Discussion 63
 - 6.4 Conclusion 64
- 7 Discussion 67**
 - 7.1 Research questions 67
 - 7.1.1 Vulnerabilities in Smart Home Environments 67
 - 7.1.2 Effectiveness of Automated Penetration Testing 68
 - 7.1.3 The Impact of Automated Penetration Testing on Security Posture 69
 - 7.1.4 Ethical Considerations in Automated Penetration Testing for Smart Homes 70
 - 7.1.5 Heterogeneity of Smart Home Environments 71
 - 7.1.6 Limitations of Automated Penetration Testing 72
 - 7.1.7 Key Success Factors of Automated Penetration Testing Tools 73
 - 7.2 IoTective 73
 - 7.2.1 Requirements 74
 - 7.2.2 Key Success Factors 76

7.2.3	Tool Limitations	77
7.2.4	Implications and Significance of Findings	78
8	Conclusion	81
9	Future Work	83
	Bibliography	85

Figures

2.1	The four layers of IoT [15].	9
2.2	ZigBee protocol stack [18].	11
2.3	Bluetooth Low Energy (BLE) Architecture [20].	13
2.4	Smart home technology display at Westfield White City, London, 2019 [32]	18
3.1	BLE threat model based on the attack domain [53].	30
4.1	Overall design of IoTective.	40
4.2	Design of the initialization phase.	40
4.3	Design of the scanning phase.	41
4.4	Design of the sniffing phase.	42
4.5	Design of the reporting phase.	42
6.1	Smart home environment used for testing.	54
6.2	Scan initialization.	55
6.3	Result from ARP scan.	56
6.4	Result from nmap scan of Raspberry Pi.	56
6.5	Result from nmap scan of Philips Lighting BV host.	57
6.6	Configuration information gathered from Philips Hue bridge API.	58
6.7	List of all reports generated by IoTective.	59
6.8	Scan information displayed in the report.	60
6.9	ZigBee device information displayed in the report.	60
6.10	Raspberry Pi device information from the report.	61
6.11	Information about port 8123 on the Raspberry Pi.	61
6.12	CVE-2016-5636 identified on the Home Assistant service.	61
6.13	Information gathered from Sony WH-1000XM4 headset.	62
6.14	Service information gathered from Sony WH-1000XM4 headset.	62
6.15	Mapping of discovered devices and devices connected to the router.	64

Tables

2.1	Comparison of home network wireless standards [24][22].	15
3.1	Vulnerabilities found in smart lighting [54].	31
3.2	Feature implementation of PENIoT for each protocol.	32
3.3	Comparison of protocol support.	35
3.4	Comparison of attack support.	35
3.5	Comparison of user experience and Python version.	35

Code Listings

5.1	Initialization Data	46
5.2	ARP Scan	47
5.3	Port Scan	48
5.4	Philips Hue mDNS discovery	48
5.5	Capture beacon frames to obtain BSSIDs	49
5.6	Packet handler identifying connected hosts	50
5.7	ZigBee device discovery	50
5.8	Bluetooth device enumeration	51
5.9	Generation of Markdown language	52

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks. 10, 11, 24

AES Advanced Encryption Standard. 12

AI Artificial Intelligence. 18

AMQP Advanced Message Queuing Protocol. 10, 32, 35

AP Access Point. 53

API Application Programming Interface. iii, v, xiii, 5, 34, 48, 57, 58, 84

ARP Address Resolution Protocol. xiii, 4, 40, 47, 55, 56, 63, 64

ASCII American Standard Code for Information Interchange. 45

AWS Amazon Web Services. 33

BLE Bluetooth Low Energy. xiii, 5, 6, 10, 12–15, 29–35, 44, 45

BSSID Basic Service Set Identifier. 41, 42, 50, 58

CAN Controller Area Network. 33

CBOR Concise Binary Object Representation. 10

CCM Counter with CBC-MAC. 12

CLI Command-Line Interface. 35, 36

CoAP Constrained Application Protocol. 10, 28, 32, 35

CPE Common Platform Enumeration. 41, 60, 61

CPU Central Processing Unit. 8

CVE Common Vulnerabilities and Exposures. 4, 31, 41, 49, 57, 61, 63, 77

CVSS Common Vulnerability Scoring System. 61, 77

- DICOM** Digital Imaging and Communications in Medicine. 33
- DoS** Denial-of-Service. 21, 24, 28, 31, 32, 35
- EEPROM** Electrically Erasable Programmable Read-Only Memory. 33
- ESSID** Extended Service Set Identifier. 41, 49, 58
- GUI** Graphical User Interface. 35, 36
- HCI** Host Controller Interface. 12
- HTTP** Hypertext Transfer Protocol. 45, 56
- I2C** Inter-Integrated Circuit. 33
- IEEE** Institute of Electrical and Electronics Engineers. 10–15, 27, 34, 50, 58
- IoT** Internet of Things. iii, v, xiii, 1, 2, 4–20, 22–25, 27, 28, 32–34, 41, 44, 67, 68, 76–79, 81, 84
- IP** Internet Protocol. 4, 5, 24, 40, 41, 46, 48, 55, 56
- IPS** Intrusion Prevention System. 2
- IPv6** Internet Protocol version 6. 10
- ISM** Industrial, Scientific, and Medical. 12
- JSON** JavaScript Object Notation. 10, 42, 48, 51, 78
- JTAG** Joint Test Action Group. 21, 33
- L2CAP** Logical Link Control and Adaptation Protocol. 12
- LDoS** Low-Rate Denial-of-Service. 28
- LoRaWAN** Long Range Wide Area Network. 10
- LPWAN** Low-Power Wide Area Network. 14, 15
- LQI** Link Quality Indicator. 59
- MAC** Media Access Control. 4, 11, 34, 40, 47, 50, 55, 56, 58
- MCU** Microcontroller Unit. 8
- mDNS** Multicast DNS. 33, 35, 36, 41, 45, 48, 57
- MIC** message Integrity Code. 12

- MitM** Man-in-the-Middle. 21, 24, 28–31
- MPU** Microprocessor Unit. 8
- MQTT** Message Queuing Telemetry Transport. 28, 32–35
- MU-MIMO** Multi-User Multiple-Input Multiple-Output. 14
- NFC** Near Field Communication. 11, 34
- NTNU** Norwegian University of Science and Technology. 5
- NVD** National Vulnerability Database. 31
- OF-DMA** Orthogonal Frequency-Division Multiple Access. 14
- OS** Operating System. 44, 55, 56
- OSI** Open Systems Interconnection. 9, 10
- OWASP** Open Web Application Security Project. 6, 25, 67, 68
- PAN ID** Personal Area Network Identifier. 27, 28, 42, 59
- PDF** Portable Document Format. 32
- PoC** Proof-of-Concept. iii, v, 6, 53, 64, 74, 77, 79, 81
- QoS** Quality of Service. 39
- RFID** Radio Frequency Identification. 9, 23
- RPL** Routing Protocol for Low-Power and Lossy Networks. 22, 33
- RSSI** Received Signal Strength Indicator. 62
- SBOM** Software Bill of Material. 33
- SPI** Serial Peripheral Interface. 33
- SWD** Serial Wire Debug. 33
- TCP** Transmission Control Protocol. 10, 31, 33
- TCP/IP** Transmission Control Protocol/Internet Protocol. 24, 33
- TI** Texas Instruments. 34, 53
- UART** Universal Asynchronous Receiver-Transmitter. 21, 33

UDP User Datagram Protocol. 10, 33

UPnP Universal Plug and Play. 34

URL Uniform Resource Locator. 41, 48

USB Universal Serial Bus. 54

UUID Universally Unique Identifier. 13, 58, 62, 63

VM Virtual Machine. 53

WPAN Wireless Personal Area Network. 14, 15

ZNP ZigBee Network Processor. 45

Chapter 1

Introduction

This chapter serves as an introduction to the master's project and provides an overview of the thesis. The aim is to introduce the reader to the topic, goals, and scope of the project while also discussing its limitations.

1.1 Project Background

The increasing prevalence of IoT devices has led to a growing concern about their security [1]. Many IoT devices are designed without security in mind and may contain vulnerabilities that can be exploited by attackers. These devices are often connected to home networks and may have access to sensitive information, making them attractive targets for attackers. Moreover, the lack of standardization in IoT protocols and hardware make them more difficult to secure. As a result, the need for effective IoT security testing tools has become increasingly important.

Penetration testing is a critical component of security testing that involves identifying vulnerabilities in a system and exploiting them to gain unauthorized access [2]. Penetration testing tools exist for many different types of systems, but there is a lack of tools that specifically target IoT devices. The purpose of this project is to develop an open-source penetration testing tool that is designed specifically for testing the security of IoT devices.

In recent years, there has been a growing interest in open-source security automation tools for the home network. Many of these tools have been developed to help users identify vulnerabilities and secure their networks against potential attacks. The market for such tools is currently dominated by a few popular options, such as Nmap¹, Wireshark², and Metasploit³, but there is a growing demand for more specialized tools that can perform automated security testing in home environments.

Some open-source tools have been developed for home network security auto-

¹Nmap (Network Mapper) <https://nmap.org/>

²Wireshark <https://www.wireshark.org/>

³Metasploit <https://metasploit.com/>

mation, such as OpenWIPS-ng [3], which is an open-source wireless Intrusion Prevention System (IPS) that can detect and prevent attacks on Wi-Fi networks. Another example is Fing [4], which is a freemium network scanner that can help users identify all devices connected to their networks and detect vulnerabilities. However, there is still room for improvement in terms of automated security testing tools for home networks, particularly in terms of integration with IoT devices and automation of the testing process. As the number of IoT devices in the home continues to grow, the need for more comprehensive and user-friendly security testing tools will also increase.

1.2 Project Goals

The primary goal of this master project is to develop an automated tool that can assist penetration testers in the early stages of reconnaissance, planning, and scanning. The tool aims to make the process of gathering information about a target network easier, faster, and more efficient. By automating this process, the tool can help reduce the amount of time and effort required for reconnaissance and scanning, allowing penetration testers to focus on the later stages of the testing process.

Furthermore, the tool aims to support a variety of different devices and protocols commonly found in IoT environments, such as Wi-Fi, Ethernet, ZigBee, and Bluetooth [5][6][7][8]. By supporting these protocols, the tool will be able to analyze and identify potential vulnerabilities in a range of IoT devices and other consumer products connected to the network.

Additionally, the tool will be designed with ease of use and flexibility in mind, with an intuitive user interface and the ability to customize the scanning capability of the tool. The tool will also be open-source and available on GitHub, allowing for community contributions and collaboration. Ultimately, the project goal is to provide penetration testers with a powerful, flexible, and efficient tool that can help identify potential security issues in IoT environments.

1.3 Research Questions

Following are the research questions this thesis will attempt to answer. Each question includes a brief explanation of why it is important and what insight it could give.

Research Question 1: What are the most common vulnerabilities found in smart home environments, and how can they be exploited by attackers?

The most common vulnerabilities will be the most interesting to identify as these are most likely to be discovered in smart homes and should be the primary target of the automation tool. Knowledge of how these vulnerabilities can be exploited can help us give relevant information on how a vulnerability can effect the user and how the user could mitigate it.

Research Question 2: How effective is automated penetration testing at identifying vulnerabilities in a smart home environment compared to manual testing?

The effectiveness of automated penetration testing will determine the usefulness of such a tool in the smart home. However, measuring effectiveness of automated penetration testing against manual penetration testing is challenging because the it will depend on many factors such as the skill of the analyst, the complexity of the environment, and the potential vulnerabilities.

Research Question 3: What impact does the use of automated pentesting have on the overall security posture of a smart home environment?

Automated penetration testing tools can be useful in finding vulnerabilities, but to the end-user, advice and recommendation on how to mitigate security risks could be just as important.

Research Question 4: What are the ethical considerations that should be taken into account when conducting automated penetration testing in smart home environments?

An open-source tool, such as the one implemented in this project, has the potential to be useful both to adversaries and legitimate users. Because of the risks in creating a tool that could potentially do more harm than good, it's important to look at what ethical consideration should be taken when implementing and running automated penetration testing tools in smart homes.

Research Question 5: How does the heterogeneity of smart home environments affect the effectiveness of automated pentesting, and what strategies can be employed to overcome these challenges?

When creating a tool to be run in various environments, it's important to look into ways to make the test that are conducted as generalizable as possible. A crucial factors is to not lose too much quality information from the analysis because of a less targeted approach.

Research Question 6: What are the limitations of automated pentesting in smart home environments, and how can they be addressed to improve the effectiveness of the testing process?

Answering this question will highlight what automated penetration testing cannot do or can only do to a limited extent. This will be interesting to an analyst because it would indicate what tests should be conducted to make the testing more complete.

Research Question 7: What are the key success factors for the implementation of automated pentesting in smart home environments?

Given the range of methods and techniques that could be utilized, identifying the key success factors is critical when it comes to prioritization of features for our tool.

1.4 Research Design

The research conducted in this project will use a mixed methodology of both a quantitative and qualitative research approach. Since the project involves the development of program/script that will perform automated penetration testing and vulnerability discovery, the quantitative approach could be used to measure the effectiveness of the script. The script will be compared to a manual approach of penetration testing as well as similar automated tools, using the same smart home environment. This way we can compare how many vulnerabilities we were able to discover using the two different methods, as well as compare the severity of the discovered vulnerabilities. We will measure the time used to perform the scanning and discuss the differences in how much relevant information was gathered. The qualitative research approach will take a look at the usability and user experience of the tool.

The study will use a case study approach which involves an in-depth analysis of a single smart home environment. This environment will be as close to a usual smart home setup as possible and use common vendors such as Philips, Samsung, and Amazon. The network will also contain non-IoT devices such as computers, phones, and media stations. The tool will attempt to gather as much information about the environment as possible and use it to identify vulnerabilities and give the analyst useful information for further analysis. The tool does not aim to be a comprehensive penetration testing tool, however, the automated aspect of it gives the user an effective method for gaining initial information.

1.5 Ethical Considerations

The aim of this project is to develop a tool that can perform security testing on smart home environments, with the goal of identifying vulnerabilities and weaknesses. The tool will utilize a variety of techniques, both passive and active, to gather the necessary data. Passive techniques include using protocols that share information between hosts, such as the Address Resolution Protocol (ARP), network scanning, and traffic monitoring. While these techniques are less risky, they may not provide a complete picture of potential vulnerabilities. Active techniques involve attempting to exploit common vulnerabilities, but come with greater risk of damage to target systems. To balance these risks, the tool will primarily rely on passive techniques.

The tool will not collect or log any private information, but will focus on identifying potential weaknesses that could lead to the leakage of such information. The information it reports will include device information such as Internet Protocol (IP) addresses, Media Access Control (MAC) addresses, ports, and service versions, as well as identified and potential Common Vulnerabilities and Exposures (CVEs) related to devices and services running on the network. All information will be stored on the host that runs the script and will not be sent over the internet or shared with third parties.

As an open-source automation tool, the project will provide an opportunity for users to test their home networks for vulnerabilities. However, to prevent misuse by potential adversaries, the tool will require users to have already obtained access to the network. It will only allow testing against private IP addresses, and will not permit any probing over the Internet. While this approach reduces the likelihood of misuse, however, it won't prevent anyone from removing this restriction themselves by changing the code. There is always a risk that an adversary has gained unauthorized access to a network and may attempt to use the tool for malicious purposes. To address this, the tool will prompt users with a disclaimer that outlines the potential risks and advises responsible use.

The proof-of-concept will be tested in a real-world environment that includes both smart home devices and common consumer devices. All users of the environment will be informed of the testing and its potential impact on their information and devices. All devices in the environment will be owned by either the users or by Norwegian University of Science and Technology (NTNU) and will be used with their consent for research purposes. The cloud services of vendors will not be tested, although they may be used to gather additional publicly available information.

1.6 Project Scope and Limitations

The aim of the development of the tool is to be as general purpose for a home environment as possible, with some features specific for IoT devices. Creating a tool with focus on IoT in general has its limitations because of the heterogeneity of the IoT domain. Because different devices and vendors use different protocols and APIs, it is difficult to create a tool that fits all. The tool is therefore limited to using aspects of security testing which are generally applicable to all home networks which includes network scanning and traffic sniffing.

IoT devices are often useful much because of their wireless aspect. Protocols such as Wi-Fi, ZigBee and Bluetooth Low Energy (BLE) are therefore common. Testing wireless communication quickly becomes difficult considering that there are usually many devices in residential areas that are communicating at the same time. When performing network sniffing in monitoring mode we are able to intercept all packets sent over the air, regardless of their intended target. Due to the automated nature of the tool, performing penetration testing on all devices in a area would not be ethical since it could both cause harm to devices and potentially expose sensitive information.

Wireless penetration testing often necessitates a focused and selective approach, with analysts handpicking specific devices to target based on prior knowledge or intelligence. This level of targeting typically involves a more manual process that falls outside the scope of the automated tool being discussed. Consequently, in order to maintain the tool's automated nature, the wireless feature is limited to passive scanning and reporting of discovered devices. This information can then complement more manual testing efforts. Additionally, the user will be

prompted to specify the Wi-Fi network to target, and it is the user's responsibility to obtain prior authorization for conducting the testing.

The case study approach allows us to showcase how the tool performs in a typical home network. However, this approach is limited in that it is difficult to replicate which in turn makes it difficult for external validation of the results. The tool will be made open-source, making it open to inspection and testing, allowing anyone to perform testing in their own environments. We will also compare the tool only with other tools that are also open-source.

1.7 Thesis Structure

This thesis is structured as follows: Chapter 1 provides an overview of the project, starting with a background discussion of the IoT concept and its significance in modern society. The chapter further outlines the project's goals and limitations and concludes by describing the thesis structure. Chapter 2, entitled "Background," discusses the three primary areas that form the foundation for the thesis. The chapter begins by defining the IoT concept and discussing the devices, architecture, communication technologies, advantages, and security requirements of IoT. The chapter then moves on to the concept of smart homes, including their automation and security issues, before covering penetration testing and the Open Web Application Security Project (OWASP) IoT Project.

Chapter 3, "Related Work," examines the existing literature on IoT vulnerabilities and penetration testing frameworks. It discusses the vulnerabilities in IoT products, such as ZigBee and BLE vulnerabilities, and reviews various IoT penetration testing frameworks such as PENIoT, EXPLIOT, HomePwn, KillerBee, and Artorias. Finally, the chapter compares these frameworks.

Chapter 4 provides details of our design and outlines the functional requirements, development environment, and architecture of the tool. Chapter 5 provides implementation details, including the Python modules used, code structure, and main functions of the automated penetration testing tool being developed.

Chapter 6 provides a Proof-of-Concept (PoC) to demonstrate the effectiveness of our tool. It describes the environment setup, test execution, and results when testing a popular smart home platform.

Chapter 7, "Discussion," addresses the research questions related to vulnerabilities in smart home environments, the effectiveness of automated penetration testing, ethical considerations, heterogeneity of smart home environments, limitations of automated penetration testing, and key success factors of automated penetration testing tools. It also provides an overview of our tool, its functional requirements, and limitations.

Chapter 8 concludes and summarizes the main findings of the thesis, while Chapter 9 proposes potential directions for further development of our tool.

Chapter 2

Background

This chapter provides an introduction to the essential concepts related to IoT, smart homes, and penetration testing. It covers relevant terminology, technologies, frameworks, standards, and tools used to perform penetration testing for IoT. The information presented in this chapter will give readers the required knowledge to understand the contributions of this master project.

2.1 Internet of Things (IoT)

The concept of IoT traces back to the late 1980s when a group of researchers at Carnegie Mellon University began experimenting with connecting everyday objects to the Internet [9]. However, it was not until the late 1990s that the term "Internet of Things" was first coined and began to gain momentum. IoT was first used by a British technology pioneer called Kevin Ashton to refer to physical objects that can connect over the Internet through sensors [9]. Although the term "IoT" is widely used, there is no broad consensus on its definition. In general, IoT refers to a network of networks consisting of uniquely identifiable endpoints that capture and share data. A review from 2019 [10] used the following definition: "... the interconnection of machines and devices through the Internet, enabling the creation of data that can yield analytical insights and support new operations."

IoT has evolved and expanded over the years, driven by technological advances such as the development of wireless technologies like Wi-Fi, Bluetooth, and ZigBee. This development opened up a wide range of possibilities for IoT as it made it possible to connect a much larger number of devices and allowed for greater flexibility in terms of device placement. In recent years, new IoT applications such as smart home automation, industrial Internet, and connected cars have entered the market. The increasing availability of cloud computing and big data analytics has further driven IoT growth, making it applicable in areas ranging from agriculture and manufacturing to healthcare and transportation [11][12][13].

2.1.1 IoT Devices

An IoT environment comprises of interconnected IoT devices that transmit signals to perform tasks and exchange data. While IoT devices have different architectures and applications, they share some fundamental concepts and vocabulary, which include [14]:

- **Actuators:** IoT devices often perform physical actions. Actuators receive signals from sensors and execute specific actions based on them. For instance, a motion sensor detecting a movement can send a signal to an actuator that turns on the lights.
- **Embedded systems:** Embedded systems refer to computer systems that are integrated into devices and products to perform specific functions. They are compact, low-power, and cost-effective. In the context of IoT, embedded systems monitor and control connected devices and collect and process data from sensors. They communicate with other devices and systems.
- **Intelligent devices:** These devices have higher computing power and intelligence than traditional connected devices. They can analyze and process data, make decisions, and take action based on that data. In the context of IoT, an intelligent device could be a hub that receives data from sensors and uses it to compute the actuator's subsequent action.
- **Microcontroller Unit (MCU):** A MCU is a compact and integrated computer system on a single chip. It typically contains a Central Processing Unit (CPU), memory, and input/output (I/O) peripherals, and is designed to perform specific tasks in embedded systems.
- **Microprocessor Unit (MPU):** A MPU is a CPU integrated on a single chip. It is designed to perform general-purpose computing tasks such as data processing and control.
- **Non-computing devices:** These devices do not have the ability to compute but can connect and transmit data. These devices are typically used for data collection and sensing and can include sensors, actuators, and other types of connected devices.
- **Transducers:** Transducers convert physical quantities into electrical signals and vice versa. They collect data from the environment and control physical devices. Some examples include sensors, actuators, microphones, speakers, and lights.
- **Sensors:** Sensors measure physical characteristics like temperature, humidity, light, pressure, or motion, and convert that information into an electrical signal. These signals can be transmitted to other devices, such as computers or smartphones, for analysis and processing.

Connectivity is a crucial aspect of an IoT environment, and devices communicate either directly or through gateways. Gateways, also called "hubs," help IoT devices connect to the corresponding cloud servers and establish device-to-device communications. Gateways provide an additional layer of security by acting as intermediaries between devices and the internet.

2.1.2 IoT Architecture

IoT creates a heterogeneous environment that poses various problems related to formalization, standardization, data, and security. One way of visualizing the architecture of IoT is by layering it into four layers: the application layer, physical layer, network layer, and perception layer, as shown in Figure 2.1 [15].

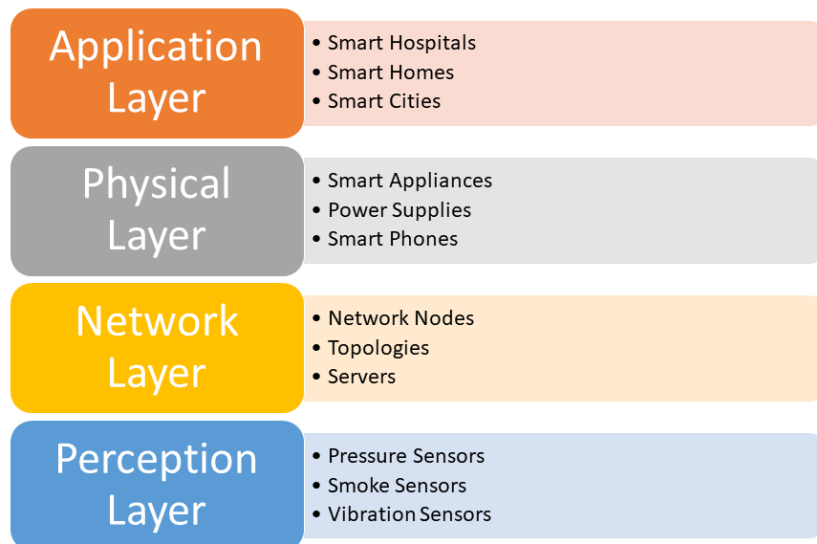


Figure 2.1: The four layers of IoT [15].

The application layer serves as an interface between the network and IoT devices, confirming which applications are gaining access and what services are delivered to different applications based on information gathered from sensors. The physical layer consists of hardware devices and physical components that sense the environment in some form, communicating with other connected devices to perform desired tasks. The network layer is responsible for communication between various devices, often over extensive distances. Its primary objective is the transportation of data from objects through sensors via wires or wireless protocols. The perception layer perceives input from different devices and technologies, such as pressure sensors, smoke sensors, vibration sensors, and Radio Frequency Identification (RFID) sensors [15].

Another way to visualize the architecture of IoT devices is by mapping the relevant protocols to the Open Systems Interconnection (OSI) model [16]. The IoT protocol stack can be organized and structured according to the seven layers of the OSI model:

1. **Physical layer:** This layer is responsible for transmitting raw bits over the physical medium, such as a wired or wireless connection. In the context of IoT, this layer includes protocols such as Zigbee, Z-Wave, Bluetooth, and Wi-Fi, which are responsible for communication between IoT devices.

2. **Data Link layer:** This layer is responsible for providing transmission of data frames between devices on a network. In the context of IoT, this layer includes protocols such as Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 and Long Range Wide Area Network (LoRaWAN), used for low-power and long-range communication between IoT devices. IEEE 802.15.4 is the groundwork for protocols such as ZigBee and IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) to build wireless embedded networks.
3. **Network layer:** This layer is responsible for routing data between devices on a network, as well as between networks. In the context of IoT, this layer includes protocols such as Internet Protocol version 6 (IPv6) and 6LoWPAN, which provide addressability and routing for IoT devices.
4. **Transport layer:** This layer is responsible for providing reliable data transfer between devices. In the context of IoT, this layer includes protocols such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), used for communication between IoT devices and other systems, such as cloud services and data centers.
5. **Session layer:** This layer provides a logical connection between applications running on different devices. In the context of IoT, this layer is not typically used, as communication between IoT devices is managed at lower levels of the stack.
6. **Presentation layer:** This layer is responsible for encoding and decoding data into a standardized format. In the context of IoT, this layer includes protocols such as Concise Binary Object Representation (CBOR) and JavaScript Object Notation (JSON), performing data encoding and decoding.
7. **Application layer:** This layer provides a high-level interface to the underlying network and communication protocols. In the context of IoT, this layer includes applications and services, such as Smart home systems, industrial control systems, and wearable devices, which use the lower layers of the protocol stack to communicate and exchange data. This layer includes protocols such as Constrained Application Protocol (CoAP) and Advanced Message Queuing Protocol (AMQP).

The composition and configuration of the protocol stack may vary depending on the specific requirements and constraints of the environment. However, this mapping provides a general understanding of how the different protocols and technologies used in IoT can be organized and structured according to the seven layers of the OSI model.

2.1.3 IoT Communication Technologies

This subsection presents some of the most popular transmission technologies used in IoT networks. Since smart homes consist of devices communicating over short distances, we will focus on low-power, short-range networks. Specifically, we will discuss ZigBee, BLE, and Wi-Fi since these are the most relevant protocols for

our tool, which will be presented later. However, it is worth noting that other short-range communication protocols such as Z-wave¹, Near Field Communication (NFC)², and 6LoWPAN³ are also commonly used in IoT environments and smart homes.

ZigBee

ZigBee is a wireless communication standard developed by the ZigBee Alliance, which has gained widespread use in various fields such as consumer electronics, home and building automation, industrial controls, medical sensor applications, toys, and games. This popularity is due to its low cost and low power consumption. The ZigBee standard builds upon the IEEE 802.15.4 standard, which specifies the physical layer and the MAC sub-layer [17].

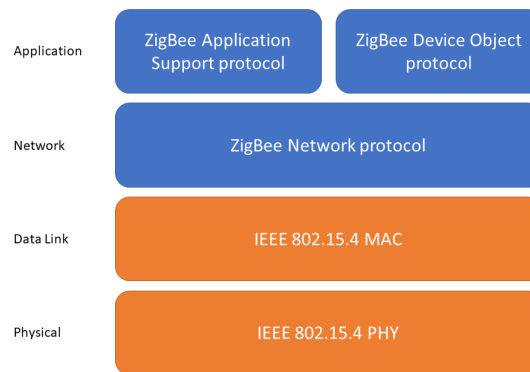


Figure 2.2: ZigBee protocol stack [18].

The ZigBee protocol stack comprises four layers: the physical layer, the data link layer, the network layer, and the application layer, as shown in Figure 2.2. The physical layer defines the physical characteristics of wireless communication, such as the frequency and modulation used. The data link layer manages access to the wireless medium and provides error detection and correction. The network layer handles the routing and addressing of packets within the network, security, and configuration of new devices. The application layer contains the actual application code that runs on the device, while the application support protocol provides compatibility for application-specific functions, such as security and power management.

To ensure interoperability between different vendors, ZigBee includes application profiles that define functional procedures and message formats. Additionally,

¹Z-wave is a wireless communication protocol used primarily for home automation. <https://www.z-wave.com/>

²NFC (Near Field Communication) is a wireless communication technology used for short-range communication between devices. <http://nearfieldcommunication.org/>

³6LoWPAN is a low-power wireless communication protocol designed for use in IoT networks. <https://www.gartner.com/en/information-technology/glossary/6lowpan>

ZigBee provides communication security using message encryption and authentication. This is accomplished through the use of Advanced Encryption Standard (AES) in the Counter with CBC-MAC (CCM) mode, with a 128-bit message Integrity Code (MIC) for integrity protection and a 4-byte counter for protection against replay attacks [19].

The ZigBee protocol stack provides a reliable wireless communication system by integrating each layer of the stack. It is noteworthy that, while the physical and data link layers are defined by the IEEE 802.15.4 standard, the network and application layers are ZigBee-specific additions. Thus, the ZigBee protocol stack builds upon and extends the IEEE 802.15.4 standard to provide a complete wireless communication solution.

Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a wireless communication technology specifically designed for low-power, low-data-rate applications. It operates in the 2.4 GHz Industrial, Scientific, and Medical (ISM) band and adopts a star topology, where a central device, such as a smartphone, connects to one or more peripheral devices. BLE utilizes a frequency hopping technique to mitigate interference with other devices operating in the same band [20].

BLE is particularly well-suited for a wide range of devices that require low power consumption, including wearables, fitness trackers, smart home devices, medical devices, and various IoT devices. Based on the Bluetooth 4.0 specification, BLE comprises three fundamental components: the controller, host, and application. Figure 2.3 illustrates the complete architecture.

The *controller* serves as the physical device responsible for transmitting and receiving radio signals. It interfaces with the external environment through an antenna and communicates with the host through the Host Controller Interface (HCI). The Physical Layer handles bit transmission and reception using the 2.4 GHz radio, while the Direct Test Mode enables testers to assess a controller's transmission and receiving capabilities.

The Link Layer, situated within the controller, is tasked with scanning, advertising, and establishing and maintaining connections. During connection setup, it utilizes one channel for advertising when connecting to new devices and another channel for transmitting data to already connected devices. Facilitating communication between the host and controller is the HCI, which provides a standardized interface.

The *host* assumes several critical functions within the Host Controller Interface (HCI) framework. The Logical Link Control and Adaptation Protocol (L2CAP) utilizes fixed channels for the signalling channel, Attribute Protocol, and Security Manager. The Security Manager Protocol governs pairing and key distribution, establishing trust between devices through authentication. The Attribute Protocol defines rules for accessing data on a peer device, with attributes denoting specific data pieces associated with permissions.

Building upon the Attribute Protocol and Security Manager, the Generic Attribute Profile defines attribute types and their usage. Additionally, the Generic Access Profile specifies device discovery, connection procedures, and the presentation of relevant information to users. It introduces mechanisms like bonding, which establishes a permanent relationship, and privacy mechanisms that enable devices to utilize constantly changing and random addresses for identification.

The Application Layer operates atop the BLE stack, defining characteristic, service, and profile specifications based on the Generic Attribute Profile. A characteristic represents a piece of data with a known format, labeled with a Universally Unique Identifier (UUID). A service, on the other hand, presents a human-readable specification of a set of characteristics and their associated behavior. Profiles, which describe two or more devices and the services they provide, further enhance interoperability [20].

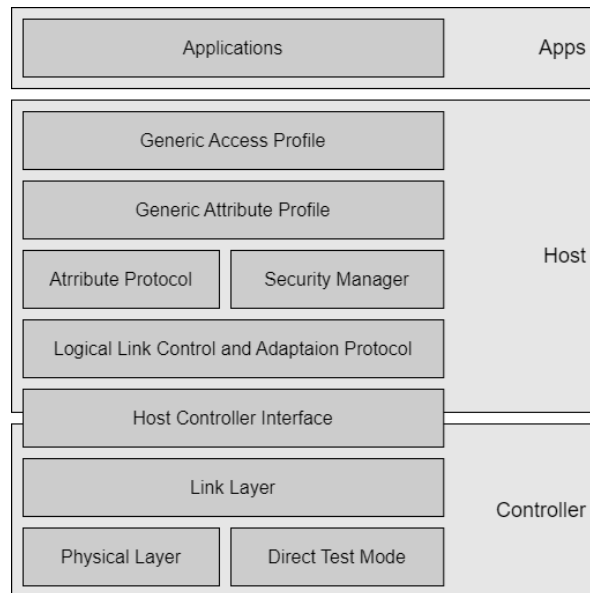


Figure 2.3: Bluetooth Low Energy (BLE) Architecture [20].

Wi-Fi

Wi-Fi, based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard [21], encompasses several sub-standards. Within the context of IoT, one particularly relevant standard is IEEE 802.11ah, also known as Wi-Fi HaLow. Introduced in 2016, Wi-Fi HaLow was specifically developed to address the requirements of IoT devices by offering higher data rates over longer distances compared to short-range technologies like BLE and ZigBee [22]. Operating in sub-gigahertz frequency bands, Wi-Fi HaLow supports a larger number of devices within a single network, exhibits low power consumption, and provides extended range in comparison to traditional Wi-Fi standards like 802.11b/g/n/ac. With data rates of up

to 347 Kbps, Wi-Fi HaLow proves suitable for various applications such as smart homes, industrial IoT, and smart cities, where low-data rate communication, extended battery life, and greater range are crucial requirements.

Another significant advancement in Wi-Fi technology is IEEE 802.11ax, also referred to as "Wi-Fi 6," which represents the latest generation of Wi-Fi standards and offers notable improvements over its predecessor, 802.11ac (Wi-Fi 5). Designed to meet the increasing demands of high-density environments like airports, stadiums, and crowded urban areas, 802.11ax incorporates features such as Multi-User Multiple-Input Multiple-Output (MU-MIMO) and Orthogonal Frequency-Division Multiple Access (OFDMA). These features enable more efficient utilization of available bandwidth, resulting in enhanced network performance. Wi-Fi 6 delivers higher data rates, improved network efficiency, better network capacity, and increased battery life for connected devices [23].

Comparison

BLE devices are known for their low power consumption, allowing them to operate for extended periods on small batteries. They also offer longer range compared to other low-power wireless technologies such as ZigBee, and are widely adopted by the industry due to their support by most smartphones and tablets. Both BLE and ZigBee are Wireless Personal Area Network (WPAN) technologies that offer medium data rates at short range. BLE operates at around 1-2 Mbps, which is slower than classic Bluetooth but faster than other low-power wireless standards like ZigBee.

On the other hand, Wi-Fi HaLow is a Low-Power Wide Area Network (LPWAN) technology designed for long-range communication and supports low or medium data rates [22]. Table 2.1 compares the different technologies. While Wi-Fi 6 and most traditional Wi-Fi technologies are designed for high throughput over small-scale networks with a few dozen devices and coverage under 100 meters, ZigBee uses mesh technology to extend its coverage, allowing devices to communicate with each other without a central access point.

For IoT in industry, long-range transmission is more critical than in smart homes. Therefore, Wi-Fi HaLow is a more suitable choice. Conversely, short-range communication protocols are better suited for smart home-specific IoT devices, as they offer benefits in terms of throughput and power consumption. ZigBee's low data rate is optimized for low power consumption, making it a suitable choice for IoT.

2.1.4 Advantages of IoT

IoT is a technology that seeks to simplify our lives by automating tasks and gathering valuable data that can be used to optimize productivity. According to [25] and [26], some of the many potential benefits of IoT include:

1. **Increased efficiency:** IoT devices can automatically collect and transmit

Table 2.1: Comparison of home network wireless standards [24][22].

	ZigBee	Wi-Fi HaLow	BLE	Wi-Fi 6
Category	WPAN	LPWAN	WPAN	WPAN
IEEE Standard	802.15.4	802.11ah	802.15.1	802.11ax
Frequency	2.4/Sub GHz	Sub GHz	2.4 GHz	2.4/5 GHz
Range	100 m	1000 m	30 m	100 m
Max Data Rate	250 Kbps	78 Mbps	2 Mbps	10 Gbps
Topology	Mesh	Star	Star	Star

data, allowing for real-time monitoring and control of systems and processes, which can lead to increased efficiency and cost savings.

2. **Improved decision making:** IoT devices can generate large amounts of data, which can be used to gain valuable insights and make more informed decisions.
3. **Enhanced connectivity:** IoT devices can be connected to other systems, such as cloud-based platforms, providing a seamless and integrated experience.
4. **Automation:** IoT devices can be programmed to perform certain tasks automatically, such as adjusting the temperature in a room or turning off lights when they are not needed.
5. **Remote control and monitoring:** IoT devices can be controlled and monitored remotely, which can be especially useful for managing and maintaining equipment, facilities, and infrastructure.
6. **Increased safety and security:** IoT devices can be used to monitor for potential safety hazards, such as gas leaks or intruders, and to improve security by enabling remote surveillance and access control.
7. **Improved customer experience:** IoT devices can be used to provide personalized experiences and services to customers, such as location-based marketing or customized product recommendations.
8. **New business models:** IoT can enable new business models such as pay-per-use, preventative maintenance, and outcome-based services.
9. **Cost savings:** IoT can help organizations to save costs by reducing human labor, inventory, and energy consumption.
10. **Sustainability:** IoT can be used for implementing sustainable practices such as smart grid, waste management, and energy efficiency.

2.1.5 Challenges of IoT

IoT has been around for quite some time now, but it is still considered a new and emerging technology. There are several challenges and obstacles that need to be addressed for IoT to reach its full potential.

1. **Interoperability:** Since the technology is so new and popular, there are many different vendors trying to make IoT products. However, there is a lack

of standardization when it comes to IoT products, meaning that the vendors often use different kinds of protocols and architectures when creating their products. This doesn't necessarily make their products any worse, but it puts a damper on the interoperability between devices from different vendors. Integration of IoT could already be difficult and time-consuming, and a lack of interoperability puts another layer on this constraint [27].

2. **Cost:** Cost is also a barrier for many who wish to adopt IoT. Even though IoT could help cut costs and power consumption, the devices themselves are still quite expensive, especially when considering that many of the advantages of IoT require multiple devices to be fully realised.
3. **Power consumption:** For IoT devices to be useful, they often need to be able to operate in remote areas where it is difficult to provide electricity through cables. The devices, therefore, need to rely on batteries to maintain their reliability and availability at all times. The lack of power storage puts a constraint on the device's processing power in order to prolong the battery life. The developers then need to make the devices as simple as possible, which in turn often leads to the omission of sufficient implementation of security features.
4. **Security:** Insufficient security mechanisms in IoT devices are a major problem. One of the main features of IoT is that the devices are connected remotely through wireless protocols such as Wi-Fi, Bluetooth, and ZigBee [18]. This also opens the door for adversaries to intercept the traffic through the air by using sniffers. In addition, to make IoT devices easily controllable and available, they are often connected to the Internet which in turn also makes them more available to potential hackers.
5. **Privacy:** Security is an important factor for keeping the system running and operational. Perhaps even more importantly, security mechanisms need to ensure the integrity and confidentiality of the data processed by the devices. This becomes an especially big concern when considering the privacy implications. More and more data is collected and transmitted by IoT devices, which could also include sensitive data about the users of such devices. The lack of standardisation makes it difficult to perform proper regulation of the use of IoT devices. As more and more data is gathered, organisations struggle to ensure compliance with data protection and privacy laws [27].

In this project, we have chosen to focus on the challenges regarding security and privacy since these are, arguably, the most important to overcome.

2.1.6 IoT Security Requirements

To address the security issues of IoT, we need to define the requirements for a secure IoT system. A survey from 2019 [28] categorises the requirements into the different operational levels: *Information*, *Access*, and *Functional* level.

The **Information Level** is concerned with the data and information that is being stored and transmitted between the devices. This level has the following

requirements:

1. *Integrity*: Received data should not have been altered during transmission.
2. *Anonymity*: The identity of the source of data should remain undisclosed to third parties.
3. *Confidentiality*: Only trusted IoT devices and users should be able to read the data. Any replication of messages should also be identified.
4. *Privacy*: Private information about the user should not be disclosed during transmission.

The **Access Level** specifies requirements associated with access to the network and devices. It tries to implement the following functionalities:

1. *Access Control*: Only legitimate users should be able to access the network and devices to perform administrative tasks.
2. *Authentication*: This mechanism is used to check whether the devices have the right to connect to the network.
3. *Authorization*: This mechanism ensures that only authorized devices and users are given the rights they need.

The **Functional Level** is closely associated with the availability of the devices and tries to ensure that they are operational and work properly. This level includes the following requirements:

1. *Resilience*: The network capacity should be sufficient in ensuring the security of the devices, also during an attack or failure.
2. *Self Organization*: The IoT system should be functional even if some part of the system is malfunctioning or being attacked.

2.2 Smart Homes

The term "smart home technologies" or "home automation systems" refers to digital devices that offer enhanced services to residents [29]. The origins of smart home technologies can be traced back to the late 19th and early 20th centuries, when the wealthy began using electricity to automate certain tasks in their homes [30]. Since the 1990s and 2000s, smart homes have become increasingly popular and are used to enhance efficiency, comfort, and entertainment.

There is often confusion regarding the definition of a smart home, as it shares similarities with IoT. Nonetheless, a smart home is a specific application of IoT technology that is focused on the residential environment. It refers to a residence that uses internet-connected devices to enable the remote control and automation of appliances and systems, such as lighting, heating, security, and entertainment. Smart homes aim to improve the quality of life of residents by providing security, convenience, comfort, energy efficiency, and entertainment [31].

According to a 2020 survey [32], all smart home technologies share three core attributes: they enable greater control or functionality through monitoring and sensor interfaces, they are connected in a network, allowing for optimized

service delivery and performance, and they empower users to change their behavior. While smart homes consist of many IoT devices, it is important to distinguish between the two. IoT devices have diverse applications in industries such as transportation, healthcare, and commercial settings, while smart homes are primarily focused on residential use.



Figure 2.4: Smart home technology display at Westfield White City, London, 2019 [32]

The popularity of smart home technology has surged, particularly among younger generations, due to the rapid development of IoT and other innovations such as Artificial Intelligence (AI). A 2020 report by Strategy Analytics projected a significant increase in spending on smart home technologies from \$120 billion in 2021 to \$175 billion by 2025 [33]. By 2025, nearly 390 million homes worldwide are expected to have at least one type of smart system installed, which corresponds to 19% of all households. An example of a smart home technology display is shown in Figure 2.4.

2.2.1 Smart Home Automation

Smart home automation, or just "home automation", refers to the use of technology to control and automate various systems and devices in a home, such as lighting, heating, cooling, security, and entertainment systems. The goal of smart home automation is to provide increased comfort, convenience, and energy efficiency for homeowners. Smart home automation systems typically use a central hub or controller that connects to various devices and systems in the home. The hub allows for remote control and monitoring of these systems using a smartphone app, tablet, or computer [34]. Some smart home automation systems also

have voice control capabilities, allowing homeowners to use voice commands to control their devices.

Smart home automation can also include the use of sensors, such as motion detectors, to automate certain functions, such as turning on lights when someone enters a room. Additionally, smart home automation systems can be integrated with other connected devices, such as smart thermostats and smart locks, to provide a more complete and seamless experience. These devices are also IoT devices and thus bring many of the same advantages and challenges as the general IoT device, but more home specific.

There are many popular smart home automation systems available in the market, each with its strengths and features. Some of the most popular include [35][36]:

- **Amazon Echo and Alexa:** Amazon's smart home platform and virtual assistant, Alexa, can control a wide range of connected devices, including lighting, thermostats, locks, and entertainment systems.
- **Google Home and Google Assistant:** Similar to Alexa, Google's smart home platform and virtual assistant, Google Assistant, can control a variety of connected devices and offer voice-activated control.
- **Apple HomeKit and Siri:** Apple's smart home platform, HomeKit, allows homeowners to control and automate their devices using their iPhones, iPads, or Apple Watches. HomeKit is compatible with a wide range of devices and can be controlled using Siri voice commands.
- **Samsung SmartThings:** Samsung's smart home platform, SmartThings, allows homeowners to control and automate a wide range of devices, including lighting, thermostats, locks, and security systems.
- **Nest:** A subsidiary of Google, Nest offers a range of smart home devices, including thermostats, security cameras, and smart locks. Nest devices can be controlled and automated using the Nest app or through Google Assistant.
- **Home Assistant:** This is an open-source system that can be hosted on many different systems such as Windows, Linux, MacOS, and Raspberry Pi. It integrates with over 1000 different APIs and can be used to control devices from vendors such as Google, Amazon, Samsung, and many more [37]. Home Assistant can control its connected devices through an app or a web interface.

2.2.2 Smart Home security issues

As shown in Figure 2.1, the IoT environment consists of four layers: the application layer, the physical layer, the network layer, and the perception layer. Security in the smart home relies on the security of each of these layers. H. Touqeer et al. [15] give an overview of the security challenges for each layer which are described in the following sections.

Application layer

- **Phishing attack:** An intruder could gain network access by sending emails that persuade the end user to expose confidential information or information that could be used to gain access to the system. This attack often involves sending the user to a seemingly legit web page, which is just an imitation or pretending to be a legitimate user.
- **Malicious code attack:** A Malicious worm could circulate the Internet while attempting to infect specific devices such as IoT devices. Infected devices could be manipulated or used as part of a botnet. Another type of malicious code is ransomware where the attacker attempts to blackmail the end user by encrypting the target devices and demanding a ransom.
- **Tampering with node-based applications:** By gaining control over the application, hackers manage to install rootkits. This could lead to the attacker managing to steal the identities of legitimate users or replacing hardware components with compromised ones.
- **Attacks on access control:** This type of attack occurs when authentic procedures for access control are violated. An access control compromise makes the whole system vulnerable to malicious hackers.
- **Failure to receive security patches:** Devices not connected to the Internet may fail to receive security updates at sufficient intervals. This could leave the devices exposed to high-risk vulnerabilities should the attacker gain access to the network.
- **Hacking into the smart meter/grid:** A compromised smart meter could allow an attacker to track the availability of the residents by monitoring the power consumption.
- **Vulnerable software:** Insufficient programming skills could lead to vulnerabilities being created in the program. Other weaknesses could originate in programs with weak default configurations or lack of authentication.
- **Manipulation of unstable configuration:** The IoT environment consists of many devices that all need to be configured properly, or else this could lead to security issues in the application layer.
- **Re-configuring remote device attack:** This type of attack exploits insecure network programming systems which could allow the IoT environment to be hijacked by intruders.
- **Social engineering attack:** Humans are targeted and provoked in an attempt to leak sensitive information or make the victim commit activities instructed by the attacker. This type of attack could be carried out face-to-face or through digital communication.

Perception layer

- **Sniffing and eavesdropping:** IoT networks consist of several different kinds of devices that communicate with each other and through the central hub. An attacker could intercept traffic through the air since the devices are often

transmitting data using wireless communication. Some devices use cloud-based servers for some of their operations, which means that traffic needs to go through the Internet where it could be intercepted by malicious actors. Third-party vendors are also a risk since they often have access to the information stored in the cloud.

- **Booting attacks:** In this attack, malicious attackers take advantage of insufficient or lack of security mechanisms during the booting sequence. This type of attack often requires physical communication with the device through protocols such as Universal Asynchronous Receiver-Transmitter (UART) or Joint Test Action Group (JTAG).
- **Node capturing:** Attackers can target sensors in the IoT network and replace the information they are sending with their false information. This was one of the attack techniques that made Stuxnet so effective [38].
- **Side-channel attacks:** This type of attack can expose information about systems even though they use strong encryption. Power usage, timing attacks, electromagnetic attacks, and laser-based attacks can be used to perform a side-channel attack. Apthorpe et al. [39] showed how smart homes can leak private information by monitoring the (encrypted) traffic rates.
- **Noise in data:** Noise in the wireless domain can lead to data becoming disrupted, rendering it irrelevant, incomplete, and false. This is especially relevant to IoT sensors where the timing of information received could be important to system functions.

Network layer

- **Denial-of-Service (DoS) attack:** This type of attack involves sending a large amount of data to the target device, making it unable to respond to each request in time, rendering it unreachable. This is also called a resource depletion attack. Another type of DoS attack, called "malicious packet DoS, sends a packet specially crafted to exploit a vulnerability in the target device. The result is the same, but it usually requires less machine power to execute.
- **Gateway attack:** A gateway attack targets the link between the smart home devices and the Internet, making them unable to communicate as intended. This type of attack could also be a DoS attack.
- **Unauthorized access:** An attacker with unauthorized access to Smart home devices could extract sensitive information or manipulate data.
- **Storage attacks:** Cloud storage and storage devices can contain sensitive information which could be extracted or manipulated should the security be compromised.
- **Man-in-the-Middle (MitM) attack:** A MitM attack is a type of cyber attack in which an attacker intercepts and manipulates communication between two parties. In a MitM attack, the attacker acts as a middleman between the two parties and can read, modify, or inject malicious content into the communication. MitM attacks can occur in a variety of settings, including

wired networks, wireless networks, and over the Internet. In some cases, the attacker may be able to intercept and modify communication without either party being aware that their communication has been compromised.

- **Data transit attack:** In a Data Transit Attack, the attacker intercepts and modifies the data being transmitted between IoT devices. This can result in the alteration or theft of sensitive information, such as personal or financial data. The attacker may also use the intercepted data to launch more sophisticated attacks, such as installing malware on the targeted IoT devices. Data Transit Attacks can be particularly dangerous in IoT networks because many IoT devices are not designed with security in mind and may not have the necessary security features to protect against this type of threat. Additionally, the sheer number of IoT devices in use today makes it difficult to detect and prevent Data Transit Attacks.
- **Black hole attack on Routing Protocol for Low-Power and Lossy Networks (RPL):** RPL is a routing protocol designed for use in IoT networks and is used to establish communication between nodes in a network. an attacker sends false routing information to the network, claiming that it has the shortest path to the destination. As a result, other nodes in the network will send their data packets to the attacker, who discards or "swallows" the packets, causing the data to be lost. The attacker creates a "black hole" in the network, effectively disrupting communication and making it difficult or impossible for data to reach its intended destination. The attack can result in significant network degradation, including reduced network throughput and increased latency [40].
- **Hello flood attack:** In a Hello Flood attack, the attacker sends a large number of "hello" messages to a target router or network, overwhelming the router's resources and causing it to crash or become unavailable. The attack can be performed in a variety of ways, including sending a high volume of messages to the target, sending messages with an incorrect format, or sending messages with a malicious payload.

Physical layer

- **Physical damage:** If the attacker can get a physical hold of the device, physical damage becomes a risk. This could render the device unusable as well as impact the rest of the network's functionality.
- **Environmental attacks:** Environmental forces such as rain, wind, snow, or storm can make the device lose its functionality.
- **Loss of power:** Should the power source of the devices be removed or damaged, the device itself would automatically enter power-saving mode. If this type of function is not present or if it is prevented, this could lead to data loss, malfunction, system interruptions, and damage to its components, and the device could become vulnerable to hacking and malware attacks.
- **Hardware failure:** If a hardware failure occurs, the device could start send-

ing erroneous information which again could impact the functioning of the whole network.

- **Jamming:** This kind of attack involves bombarding the target device or network with radio signals to disturb communication. This could make the target devices drain the power more quickly and break the circulation of the network. Jamming is one of the most dangerous security attacks since it is easy to perform and difficult to defend against.
- **Malicious code injection:** This attack involves injecting malicious code through a debugging interface. A trusted device in the network can then perform malicious activities which could expose sensitive data or bring down the network.
- **Overloading RFID:** This is similar to a jamming attack, but it targets the RFID metal surface. This method makes the RFID unable to transmit information or receive power.
- **Device duplication:** An attacker could clone a physical device in an IoT network. By making subtle changes, this "trusted" clone can perform malicious activities on the network like stealing sensitive information.
- **Tag duplication:** Cloning RFID tags is an easy task by using different hacking tools that include RFID readers. A popular tool is the Flipper Zero [41] which includes an RFID reader that can be used to read, clone and emulate RFID tags.

2.3 Penetration Testing

The practice of conducting Penetration Testing, also referred to as "security testing" or "pentesting," involves testing a computer system, network, or web application to identify vulnerabilities that could be exploited by attackers. The objective of the test is to gain unauthorized access to sensitive information or disrupt normal system operation. The penetration test is typically executed by simulating an attack from a malicious outsider or an insider who has some level of access to the system [2]. The outcomes of the test are then utilized to enhance the overall security of the system by identifying and addressing vulnerabilities before they can be exploited by real attackers.

Penetration Testing is usually used as an attacking technique to complement defensive measures. Given that attackers only need to exploit a single vulnerability to be successful, it is imperative that the penetration test is as comprehensive as possible. To achieve this objective, penetration testers employ automated tools to conduct different tests on the target system. These tests can be categorized into three types: interface testing, transportation testing, and system testing [42].

The first type, interface testing, targets the interfaces used by users or devices. Improper input validation is a common vulnerability discovered in user interfaces. It occurs when an application or system fails to validate or sanitize user input, allowing malicious users to inject malicious code or data into the application. This can cause the application to crash or behave in unexpected ways, leading to data

breaches, unauthorized access, and other malicious activities. Proper input validation and sanitization of all user input is crucial to mitigate these vulnerabilities. In the IoT domain, proper input validation is particularly important since IoT interfaces can be linked to code-oriented operations, such as controlling system programs.

The second type of testing, transportation testing, is used to discover weaknesses in communication protocols, cryptographic schemes, and network infrastructure. In the IoT domain, transportation testing targets protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP), Zigbee, Z-wave, Wi-Fi, Bluetooth, and 6LoWPAN. IoT networks are typically heterogeneous, using multiple protocols to allow communication between devices. A heterogeneous network is more vulnerable to attacks than a homogeneous network since there are more potential sources of new vulnerabilities.

MitM attacks, DoS attacks, replay attacks, eavesdropping, and message tampering are some common attacks used in transportation testing. MitM attacks involve an attacker intercepting communication between two parties and altering the messages exchanged. DoS attacks, which can be difficult to overcome, flood the victim's device with incoming traffic, making it challenging to respond in time. The other types of attacks seek to exploit weaknesses in communication protocols or cryptography, and can usually be mitigated by implementing strong cryptographic schemes.

The third type of testing, system testing, focuses on the proprietary programs of the target system. Attackers often lack sufficient knowledge of IoT devices, so they commonly use black-box techniques like fuzz testing. Black-box pentesting simulates an attacker who has no prior knowledge of the system or network being tested. The tester is only provided with the target's IP address or domain name and is expected to discover and exploit vulnerabilities through reconnaissance and scanning techniques. This type of testing is designed to simulate a real-world attacker who may only have limited information about the target.

Grey-box pentesting is a hybrid of black-box and white-box testing. In this type of testing, the tester is provided with some limited information about the system or network being tested, such as IP addresses, usernames, or network diagrams. This type of testing is used to simulate an attacker who may have some level of insider knowledge or access to the target. However, the diversity of IoT devices makes automated reverse-engineering a challenging task.

White-box penetration testing is a security testing technique that simulates an attacker who has complete knowledge of the system or network being tested. In this approach, the tester is provided with full access to the target's source code, network diagrams, and other sensitive information. White-box testing is used to simulate an attacker who has already infiltrated the target's network or system. This technique is also referred to as "clear box testing" or "glass box testing" in the literature [42].

2.4 OWASP IoT Project

The Open Web Application Security Project (OWASP) IoT Project refers to a comprehensive set of guidelines and best practices that aim to provide a comprehensive set of resources for securing IoT devices and systems [43][44]. The guidelines are designed to cover various areas such as device and network security, communication security, and incident management, and provide a comprehensive list of security controls for securing IoT devices.

The OWASP IoT Project is an open-source and collaborative effort, which is regularly updated by the community to ensure that the guidelines and best practices are up-to-date and reflect the current threat landscape. It is widely used by security professionals and penetration testers to conduct security assessments and identify vulnerabilities in IoT devices and systems.

In addition, the OWASP IoT Top 10 is an extension of the OWASP Top 10 project that focuses on web application security. The OWASP IoT Top 10 provides a comprehensive list of the most critical security risks facing IoT devices and systems, and aims to provide guidance on how to mitigate those risks. The list is ordered according to importance and is based on data collected from both public and private vulnerability sources, with an emphasis on issues that caused the most amount of impact and damage.

The guidelines for securing IoT devices cover areas such as device security, network security, cloud and application security, and incident management. The OWASP IoT Top 10 lists the most critical security risks facing IoT devices and systems, which include the use of weak passwords, insecure network services, insecure ecosystem interfaces, lack of secure update mechanisms, use of insecure or outdated components, insufficient privacy protection, insecure data transfer and storage, lack of device management, insecure default settings, and lack of physical hardening measures.

Chapter 3

Related Work

This chapter introduces work related to smart home security and automated pentesting of IoT. The work will give an overview of the most important vulnerabilities found in IoT which could be exploited by hackers. We will also introduce some contributions towards the automation of IoT pentesting.

3.1 IoT Vulnerabilities

In this section, we will look at the vulnerabilities that have been found in IoT devices and environments. We will focus on vulnerabilities associated with the protocols that are relevant to this thesis, as well as related to smart homes.

3.1.1 ZigBee Vulnerabilities

ZigBee is exposed to a suite of vulnerabilities. One such instance was discovered in Wara et al. [45] which included a replay attack on the ZigBee protocol on IoT applications. They were able to re-transmit captured packets to victim devices and demonstrated this on Phillips Hue bulbs and Xbee S1 and S2C modules. ZigBee is based on the IEEE 802.15.4 protocol specification which does not have great replay attack protection. This has been exploited many times by researchers, often using the "KillerBee" tool [46].

ZigBee maintains an association table that records all child nodes associated with the parent. A problem with the implementation is that the records are not deleted when a child node leaves the network after a power failure [46]. Attackers can exploit this by causing a frequent replacement of child nodes which consequently fills up the association table, making it impossible for more child nodes to join the network.

The Network Personal Area Network Identifier (PAN ID) Conflict Attack is a security vulnerability that affects some IoT devices. This attack targets the communication between IoT devices, which is typically done through wireless communication protocols such as Zigbee and Z-Wave. The attack works by exploiting a

vulnerability in the way that these protocols handle the assignment of unique identifiers called PAN IDs. The attack involves an attacker generating a fake PAN ID that is the same as the PAN ID used by the target IoT devices, which can cause communication between the devices to break down. In a Network PAN ID Conflict Attack, the attacker can disrupt the communication between IoT devices, cause them to stop working correctly, or gain access to sensitive data transmitted between the devices [46].

The authors in [47] looked at already identified vulnerabilities in ZigBee and CoAP to exploit an IoT network by initiating a passive attack to obtain sensitive and private information. By compromising the ZigBee installation code, the authors were able to perform DoS, masquerade, and MitM attacks on the network using the Message Queuing Telemetry Transport (MQTT) protocol. The installation code was decrypted by using a CC2531 dongle to sniff ZigBee traffic, Trust Center Key, Transport Key, and Wireshark. The DoS attack was performed by overloading the network with commands. Based on collected data from the gateway, a Raspberry Pi was able to perform a MitM attack which allowed it to change the state and colour of IKEA Trådfri lighting.

The Malicious Orphan Frame Attack works by exploiting a vulnerability in the way that the ZigBee protocol handles orphan frames. Orphan frames are packets of data that are sent by a device but do not receive a response from the intended recipient. In a Malicious Orphan Frame Attack, an attacker sends a large number of orphan frames to a target ZigBee network, causing the network to become congested and slowing down the communication between devices. This is one way of depleting the power of the target IoT device as well as performing a DoS attack [46].

Low-Rate Denial-of-Service (LDoS) attack is a type of DoS attack that degrades the quality of service using less traffic than a typical DoS. This type of attack is more difficult to detect and relevant methods aren't as accurate as that of DoS. Okada et al. [48] discovered that ZigBee coordinators and ZigBee routers can be attacked by targeting their buffer which holds messages sent to their child nodes. By sending pulses every time there is more space in the buffer, keeping the buffer full, all new messages will be discarded. The LDoS attack can be performed by connecting a sinkhole node and an attack node to the network or by performing a replay attack. In the second case, the attacker does not need to connect to the network.

In Morgner et al. [49], the researcher performs security tests on common IoT devices, namely Phillips Hue, Osram Lightify, GE Link, and IKEA Trådfri. They found many possible attacks that could be performed to do malicious activities on the devices:

1. **Active Device Scan:** An initial attack for performing any other attack. A scan request is sent on all ZigBee Channels and the responses let the attacker know of all devices that are also listening in on this channel. This attack works on each of the tested vendors' light bulbs and the Osram Lightify gateway, however, the GE Link hub didn't respond and the Phillips Hue hub

- required the physical button on the hub to be pushed.
2. **Identify Action Attack:** This attack allows the attacker to physically spot the relevant device by sending identify requests and witnessing the device act such as flashing, dimming, or beeping.
 3. **Reset to Factory-New Attack:** By sending a "reset to factory new request" command an attacker could cause the target device to discard the current configuration. The payload only contains the transaction identifier and results in the colour and brightness of the light bulb being set to its default state.
 4. **Permanent Disconnect Attack:** This attack differentiates from the reset to factory-new attack by making the recovery more challenging for the user. The attack can be performed in two ways: changing the channel of the Zig-Bee device or by joining the target device to a non-existent network. The user would need to physically reset the device to make fix this issue.
 5. **Hijack Attack:** The attacker sends a "network join end device request" command. The network key is encrypted using the leaked touchlink pre-configured link key, the transaction identifier received from a scan request of the target device. A "network join end device request" is sent, which includes the network key, to the targeted device. The device updates its internal parameters according to the received values and then confirms the transaction. This results in the attacker gaining full control of the target device since it is now connected to the network of the attacker.
 6. **Network Key Extraction:** This attack is performed by eavesdropping on the "scan response" and the "network join end device request" of an initial touchlink commissioning. This can be more easily achieved by first performing a reset to factory-new attack which would mean that the user would have to link the device again.

An attack discovered in October of 2022 [50] showed that a vulnerability in ZigBee allowed an attacker to factory reset the IKEA Trådfri bulb. The attacker needs to be in radio range and send an unauthenticated broadcast message, making every device within radio range affected. The attack causes the bulb to lose its configuration information regarding the ZigBee network and brightness level. The consequences are that the lights are turned on at maximum brightness level and the user is not able to control the lights since they are not connected to the network.

3.1.2 Bluetooth Low Energy (BLE) Vulnerabilities

The survey conducted by Casar et al. [51] extensively examines the weaknesses and vulnerabilities present in BLE technology. The authors delve into known vulnerabilities and attacks that have been discovered during the evaluation of BLE development. Some of the prominent attacks against BLE include sniffing, MitM, and jamming. The authors also identify various weaknesses, particularly those related to privacy. These weaknesses encompass issues such as device tracking, the

ability to infer user behavior through traffic analysis, and the disclosure of user location through mobile apps.

In a separate study by Garbelini et al. [52], a collection of vulnerabilities is documented, demonstrating the effectiveness of their framework named Sweyn-Tooth. The framework operates on a central device and conducts tests on the connection with a BLE device. By generating diverse inputs and sending them to the target BLE device, the researchers successfully caused the device to crash, enter a deadlock state, or bypass security measures. This highlights the potential for exploitation and disruption in BLE protocol implementations.

These findings emphasize the importance of addressing and mitigating vulnerabilities in BLE technology to enhance the security and privacy of BLE-enabled devices. Understanding the weaknesses and potential attack vectors is crucial for developing robust countermeasures and ensuring the integrity of BLE-based systems.

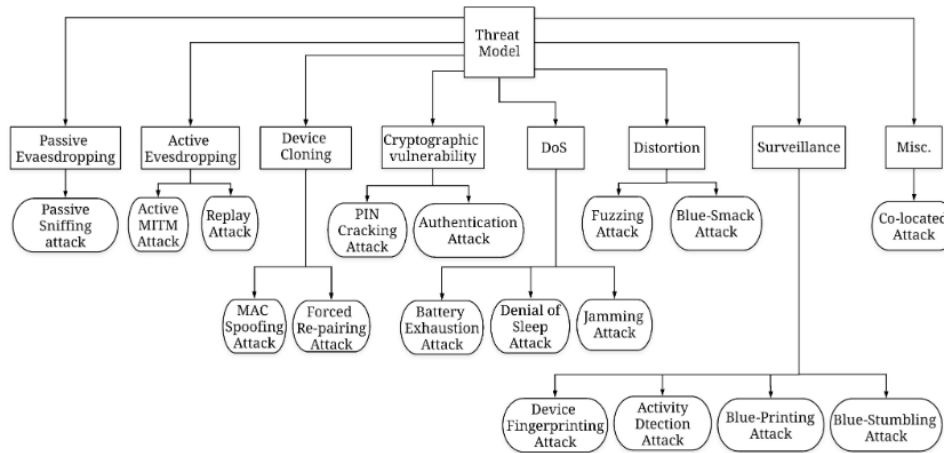


Figure 3.1: BLE threat model based on the attack domain [53].

BLE security architecture is different from that of Bluetooth classic. This is due to the trade-off between performance, security, and privacy concerns over low energy consumption. Barua et al. [53] gives an overview of the threat model against BLE devices, as shown in Figure 3.1, and discusses some of the relevant vulnerabilities found in BLE. The attacks that were classified with the highest severity were the following:

1. **Passive sniffing:** This is an attack type that is common among all wireless communication protocols. BLE's simple and predictable design of channel hopping makes it especially susceptible to passive sniffing. Once wireless communication between devices using BLE is captured, tools such as Wireshark can be used to analyse the packets. This attack can be a gateway to more malicious attacks such as MitM offline pin cracking, fuzzing, and privacy leakage.

2. **Man-in-the-Middle:** This attack is also very common in the domain of wireless communication. The attacker positions oneself between the central and end devices, and then intercepts and modifies packets without the devices being aware of it.
3. **Denial-of-Service:** There are different types of DoS attacks. The aim is to make the device or resource unavailable to the end users. A battery exhaustion attack drains the BLE device of power by keeping it active and preventing it from entering low-power idle mode. An attacker could send connection, disconnection, and service requests continuously, also called a denial of sleep attack, causing severe power drain. Another type of DoS is the use of jamming where the attacker disrupts the communication channel by sending frames either constantly, periodically, reactively or randomly.
4. **Co-Located Attack:** This threat is rooted in a vulnerability of the Android app and not BLE itself. The credentials stored in the paired Android device are potentially available to all other applications on the same device. If one of the other applications on the device is malicious, it could access unauthorized pairing-protected data of the BLE devices.

3.1.3 Vulnerabilities in IoT Products

The researchers of Davis et al. [54] did a smart home case study where they looked at vulnerabilities in four categories: physical, network, software, and encryption. They used Common Vulnerabilities and Exposures (CVE) and National Vulnerability Database (NVD) repositories to search for vulnerabilities in a handful of smart home products. Table 3.1 shows the vulnerabilities found in each of the smart lighting products.

Table 3.1: Vulnerabilities found in smart lighting [54].

Devices	Physical Vulnerability	Network Vulnerability	Software Vulnerability	Encryption Vulnerability
Philips Hue Smart Lighting	Motherboard Hack	Replay Attack DNSSEC DNS Spoofing Fake Server	"Keeps track of secret keys" Fake Server	Man-in-the-Middle attacks Unprotected communication (e.g., HTTP commands)
GE C-Life Smart Bulbs	Motherboard Hack	No Current Vulnerabilities Found	Unreliable Hub Linking	No Current Vulnerabilities Found
Feit Electric Party Bulbs	No Current Vulnerabilities Found	Not Applicable	Not Applicable	Not Applicable
HaoDeng Smart Bulbs	Motherboard Hack	Unencrypted communication	Buffer Overflow attack	Unprotected communication Information Disclosure

A vulnerability discovered in Philips Hue published in December of 2020 rendered the devices vulnerable to a DoS attack [55]. By performing a SYN flood on port TCP/80, the Phillips Hue's hub won't respond until the flooding has stopped. During the attack, the user won't be able to control the lights or use the cloud services of the vendor. Another CVE published in January of the same year showed that the Philips Hue was vulnerable to a Heap-based buffer overflow attack allowing an attacker to perform remote code execution [56].

The Sengled ZigBee Smart Bulb device was shown to be vulnerable to a DoS

attack [57]. The vulnerability allows an attacker to send malicious ZigBee messages making the device crash. The attack is performed by manipulating certain commands that are supposed to change the brightness of the lights at a certain rate. This attack works when the lights are either set to the lowest or highest brightness setting by changing the two last digits in the command string to 0x00.

3.2 IoT Penetration Testing Frameworks

This section presents frameworks for testing IoT devices and networks. Each framework has attempted to make a tool that targets some common IoT protocols, such as ZigBee and BLE, and performs both passive attacks, such as enumeration and active attacks, such as replay and DoS attacks. Many more tools are relevant for IoT, however, these are chosen because they include common protocols that allow attacks over the network as opposed to only hardware and firmware attacks.

3.2.1 PENIOT

PENIOT [58] is an open-source penetration testing tool for IoT which allows for testing against a handful of common communication protocols used by IoT devices. The tool aims to attack IoT devices with generic security attacks either semi- or fully automatic. The tools provide a suit of attacks, each specific to an IoT protocol. The tool uses a graphic interface to give the user an overview of all attacks available for the selected protocol. After an attack is selected, the user could be asked to provide the necessary information to perform the attack. After an attack is finished, a report of the test results is displayed to the user and available for download as a PDF-file. Table 3.2 shows the attacks available for each supported protocol by PENIOT. The supported protocols are Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), and Bluetooth Low Energy (BLE).

Table 3.2: Feature implementation of PENIoT for each protocol.

	AMQP	MQTT	CoAP	BLE
Sniffing	X	X	X	X
DoS attack	X	X	X	
Replay attack		X	X	X
Topic Name Fuzzing attack		X		
Random Payload Fuzzing attack	X	X	X	
Payload Size Fuzzing attack	X	X	X	
Generation Based Fuzzing attack		X		

The authors note some of the features that were planned to be implemented in the tool, but for reasons such as time constraints and lack of required hardware, were not implemented. These features include sniffing and attacks against Zigbee

and RPL. The authors also noted that the tool was written in Python2.7 which could render it legacy state and they suggested that it could be ported into Python3 - this transition has not been performed as of writing this thesis.

3.2.2 EXPLIoT

EXPLIoT [59] is an open-source framework for security testing and exploiting IoT products and IoT infrastructure. The tools consist of a set of plugins, usually referring to specific protocols, and are used to perform the assessment. The tool is written in Python3 and provides a command-line interface as well as an interactive mode. EXPLIoT aims to cover as many IoT protocols, hardware platforms, and products as possible. The supported plugins/tests are as follows:

- **BLE:** Includes scanning, fuzzing, and an exploit for unlocking Tapplock door locks.
- **BusAuditor:** Supports operations for gaining device information as well as information specific to JTAG, ARM Serial Wire Debug (SWD), UART, and Inter-Integrated Circuit (I2C).
- **Controller Area Network (CAN):** Support reading and writing on the CAN bus.
- **Crypto:** Can be used to decrypt communication between TP-Link smart devices and the Kasa home application.
- **Digital Imaging and Communications in Medicine (DICOM):** Supports scanning, fuzzing, and testing of communication and management of patient information.
- **Firmware:** Can generate a Software Bill of Material (SBOM) from the firmware file system that conforms to the CycloneDX SBOM Specification.
- **I2C:** Supports scanning of the available I2C address space and reading and writing on an Electrically Erasable Programmable Read-Only Memory (EEPROM).
- **Multicast DNS (mDNS):** Scans the local network for devices with enabled mDNS.
- **Modbus:** Supports reading and writing of coil and register values from a Modbus server running over a TCP/IP network.
- **MQTT:** Supports dictionary attacks, message publishing and subscribing both for generic devices and Amazon Web Services (AWS)-specific IoT devices.
- **nmap:** Allows for scanning of hosts, open ports and other details in a network.
- **Serial Peripheral Interface (SPI):** Supports reading and writing to SPI flash chips.
- **TCP:** Allows sending unauthorized commands to TP-Link smart devices on the same network.
- **UART:** Supports enumeration over baud rates as well as identification of undocumented or hidden console and fuzz commands and their arguments.
- **UDP:** Supports a hijack exploit for a smart plug called Kankun.

- **UPnP:** Attempts to discover devices within a network and provide details about those devices.
- **Zigbee:** Can provide device information, enumeration, sniffing, and perform a replay attack.

However, many of the plugins require hardware connectors to interact using the relevant protocol.

3.2.3 HomePwn

HomePwn [60] is a security testing tool designed to identify and exploit vulnerabilities in smart home devices. It focuses on the communication protocols used in smart homes, such as BLE, Bluetooth, MQTT, NFC, and Wi-Fi, and allows users to interact with these protocols to identify vulnerabilities. HomePwn features a user-friendly command-line interface and supports various types of attacks, including replay, injection, and eavesdropping attacks. The tool also provides comprehensive documentation and resources for users to understand the vulnerabilities and their potential impact. There are also device-specific modules such as Chromecast, Smart TVs, and cameras. HomePwn can also use the Shodan API which shows vulnerable IoT devices connected to the Internet. Some of the attacks it supports are MAC spoofing and sniffing using Bluetooth, and hijacking of Chromecast and smart TVs.

3.2.4 KillerBee

KillerBee [61] is an open-source framework and testing tool designed for exploring and exploiting the security of ZigBee and IEEE 802.15.4 networks. Developed in Python, KillerBee allows researchers and security professionals to sniff, inject, and manipulate ZigBee network traffic in real-time. The tool can also be used to conduct replay attacks, capture network keys, and perform other forms of network analysis. KillerBee supports several hardware devices and platforms, including GreatFET, HackRF, and Texas Instruments (TI) CC2531, among others. By using KillerBee, security practitioners can better understand the security risks associated with ZigBee and 802.15.4 networks, as well as develop and test effective countermeasures.

3.2.5 Comparison

This section presents a comparative analysis of the four penetration testing frameworks based on three main factors: protocol support, attack capabilities, and user experience.

To evaluate protocol support, a list of commonly used IoT device protocols in smart home environments is presented in Table 3.3, along with the frameworks that provide support for each protocol. Among the frameworks, EXPLIoT offers the most comprehensive protocol support, including ZigBee and BLE. However,

Table 3.3: Comparison of protocol support.

	PENIOT	EXPLIoT	HomePwn	KillerBee
AMQP	X			
CoAP	X	X		
MQTT	X	X	X	
mDNS		X	X	
ZigBee		X		X
Wi-Fi			X	
BLE	X	X	X	

the extent of functionality for each protocol may vary. PENIOT, on the other hand, focuses on testing commonly used protocols such as AMQP, CoAP, and MQTT, and also supports BLE. KillerBee offers exclusive support for ZigBee testing, while HomePwn supports Wi-Fi testing and BLE.

Table 3.4: Comparison of attack support.

	PENIOT	EXPLIoT	HomePwn	KillerBee
DoS	X			X
Fuzzing	X	X		
Replay	X	X		X
Sniffing	X	X	X	X
Dictionary		X		
Spoofing		X	X	
De-authentication				X

Table 3.4 presents a simplified overview of the attack capabilities of the frameworks, categorized by attack type. EXPLIoT offers the most extensive range of attacks, but each protocol only has support for a few of them. PENIOT provides its main functionality around fuzzing and DoS attacks for AMQP, CoAP, and MQTT, but it can also perform sniffing and replay attacks for BLE. HomePwn is primarily designed for discovery and device enumeration and offers support for only sniffing and MAC spoofing. KillerBee supports de-authentication attacks exclusively for ZigBee protocol.

Table 3.5: Comparison of user experience and Python version.

	PENIOT	EXPLIoT	HomePwn	KillerBee
GUI	X			
Interactive CLI		X	X	
CLI		X		X
Python Version	2.7	3.10	3.6	3 and C

The frameworks also differ in user experience, which is presented in Table 3.5.

PENIOT offers a Graphical User Interface (GUI) and is therefore more accessible for novice users. EXPLIoT and HomePwn both provide an interactive Command-Line Interface (CLI), which could provide a slightly more intuitive experience than classic command-line interfaces. EXPLIoT also offers a non-interactive CLI. KillerBee only provides a classic command-line interface.

Finally, the current status of development and support for further development also differ between the frameworks. EXPLIoT uses the newest Python version of the tools and is actively maintained. PENIOT has not been developed extensively since its launch and currently only uses Python version 2.7. KillerBee was originally written in C and is currently being ported to Python 3.5 or higher. HomePwn was written to support Python 3.6, but has not received any updates for 3-4 years.

Overall, EXPLIoT appears to be the most comprehensive framework due to its extensive protocol support and wide range of attacks. However, the choice of framework depends on the specific requirements of the user, such as the protocol and attack types to be tested, as well as the level of experience with command-line interfaces.

In comparison to the aforementioned frameworks, IoTective, the tool discussed later in this thesis, exhibits notable distinctions. IoTective employs protocols such as mDNS, ZigBee, Wi-Fi, and Bluetooth primarily for scanning and sniffing purposes. Consequently, none of the attacks enumerated in Table 3.4, with the exception of sniffing, are encompassed within IoTective's functionalities. Furthermore, IoTective integrates a GUI implemented within the terminal environment. This design choice enables users to operate the tool using cursor-based interactions, positioning it as a hybrid interface, combining features of both GUI and interactive CLI. Notably, IoTective is developed to be compatible with the latest iteration of the Python programming language, specifically Python 3.11.

Chapter 4

System Design of IoTective

This chapter will discuss the architecture of the developed tool as well as the environment used to perform the testing. This will give the user understanding what features the tool aims to accommodate and how the components of the tool works.

4.1 Design Considerations

When creating a new tool, a critical decision that needs to be made is whether to start from scratch or to build on existing software. Building a tool from scratch can offer a high degree of control over the development process and the opportunity for customization to meet specific needs. However, this approach can be time-consuming and expensive, especially if the functionality required already exists in an existing tool. Moreover, building a tool from scratch can introduce unanticipated bugs and security vulnerabilities, which may require significant time and resources to address. On the other hand, building a tool based on existing software can save time and reduce development costs while leveraging the existing functionality and features. However, it is crucial to ensure that the licensing terms and conditions of the existing software are compatible with the intended use of the new tool and that the tool is properly integrated and tested to ensure its reliability and security.

When developing a technical tool such as an automated penetration testing tool, it is crucial to consider the target audience. A tool with a lot of customization and functionality can be powerful for a proficient user, but it may be overwhelming for a novice user who wishes to test the security of their environment. Many tools attempt to be an all-in-one package, offering users the ability to perform any test they deem appropriate. Metasploit is a common example of such a tool, often used by professionals when conducting penetration testing. However, the vast functionality of Metasploit may pose a problem for novice users who do not always know where to begin. If the goal is to allow low-technical users of smart homes to test their environment, it is crucial to set the knowledge requirement bar as low as possible, providing a simple user interface for the end-user. A tool

that is user-friendly and intuitive to use is likely to be more effective in achieving the desired outcomes for the target audience.

Penetration testing is a crucial process performed by proficient analysts who possess extensive knowledge of various techniques and procedures employed to gain unauthorized access to systems or sensitive data. In a production environment, a skilled analyst must exercise caution and judiciously consider the potential impact of each test on the target system. The detection of a vulnerability such as susceptibility to reset attacks, device factory resetting, or network flooding, may potentially cause significant harm to the device. It is noteworthy that damage resulting from penetration testing may not be covered by the product's warranty, making it imperative for users to be vigilant regarding the types of tests performed on their devices. The same holds for automated penetration testing tools, particularly when the intended audience is less technically adept. It is recommended to focus on the automation of the non-intrusive parts of a penetration test to ensure safety and facilitate the gathering of information without the exploitation of devices that may result in harm.

4.2 Requirements of IoTective

This section will discuss the broad functional and non-functional requirements of the developed tool. The requirements stated describes what the goals of the tool aims to achieve without going into the technicality of how those requirements are met.

4.2.1 Functional Requirements

The functional requirements define the specific features and functions that the software should perform, which means what tasks the software should be able to perform. These requirements define what the software should do in order to fulfill its purpose. The development of IoTective uses the following functional requirements:

- **Automation:** The tool should be fully automated to minimize the need for manual intervention. This will reduce the chances of errors, save time, and increase the efficiency of the process.
- **Applicability:** The tool should be applicable in many different environments. It should be designed to be adaptable and flexible to work in various situations, such as different types of data, hardware, and software configurations. This will increase the tool's usefulness and value to users.
- **Capability:** The tools should support various smart home devices and protocols (e.g., Wi-Fi, Zigbee, Bluetooth)
- **Reporting:** The tool should be able to generate a report that is easy to understand using the information gathered from scanning and sniffing.

4.2.2 Non-Functional Requirements

Non-functional requirements are related to the characteristics and qualities of the software that are not directly related to the tasks that it performs, but are essential for its proper functioning. Non-functional requirements are often referred to as "quality attributes" or "Quality of Service (QoS) requirements". The development of IoTective uses the following non-functional requirements:

- **User-friendliness:** The tool should provide a good user experience to ensure that users can use it easily and efficiently. The interface should be intuitive, user-friendly, visually appealing, and require little technical knowledge to be operated.
- **Efficiency:** The tool should be efficient in terms of processing speed and accuracy. It should be able to perform the required tasks quickly and accurately to save time and effort.
- **Extendibility:** The tool should be developed so that it could easily be improved upon and extended with further functionality.
- **Accuracy:** The tool should provide accurate results and avoid any errors that could compromise the effectiveness of its functionality.
- **Reliability:** Ensuring that the tool is stable and provides consistent and accurate results

These requirements will be the pillars of the tool and guided the decisions made during development. Overall, this tool aims to fill a limited, but sometimes time-consuming part of penetration testing, namely planning and reconnaissance, and scanning. These phases consist of gathering the available information about the targets without exploiting them. An analyst should be able to use the tool to quickly automate this process since it is usually the later stages that require a more targeted approach.

4.3 Architecture of IoTective

In this section, we'll explore the architecture of IoTective, highlighting its phases and their contributions to modularization. We'll provide an overview of the tool's design and delve into each phase's functionality, revealing insights into IoTective's construction and design considerations.

Let's start by examining the overall design (Figure 4.1). The "Initialization" phase detects host capabilities, selects testing targets, and configures the program. If network scanning is enabled, the tool proceeds to the "Scanning" phase, focusing on device discovery without packet capture. Information is stored in the report, and if enabled, the tool moves to the "Sniffing" phase. Here, packet capture occurs via Wi-Fi, Bluetooth, and ZigBee to discover additional devices. Captured data is stored in the report. The final phase is "Reporting," responsible for generating an intuitive report for the user.

Now, let's delve deeper into each phase, discussing the operations and de-

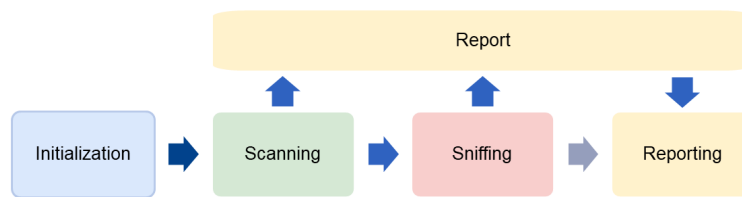


Figure 4.1: Overall design of IoTective.

cisions involved. The "Initialization" phase primarily revolves around the user selecting scan types and the corresponding adapters/interfaces to use. For example, if the user enables ZigBee sniffing, they will be prompted to choose the adapter for this operation. The same applies to Wi-Fi sniffing and network scanning. The Bluetooth adapter is configured automatically and doesn't require user specification. After the configuration, the next phase is determined based on the user's selections. If network scanning is enabled, the tool proceeds to the "Scanning" phase before entering the "Sniffing" phase, if enabled. If network scanning is disabled, the tool directly transitions to the sniffing phase if either of the sniffing options is enabled. If neither scan type option is enabled, the program exits but generates an empty report. While this empty report is primarily useful for testing the program's functionality, it doesn't provide significant usability for analysts. The design of the initialization phase is illustrated in Figure 4.2.

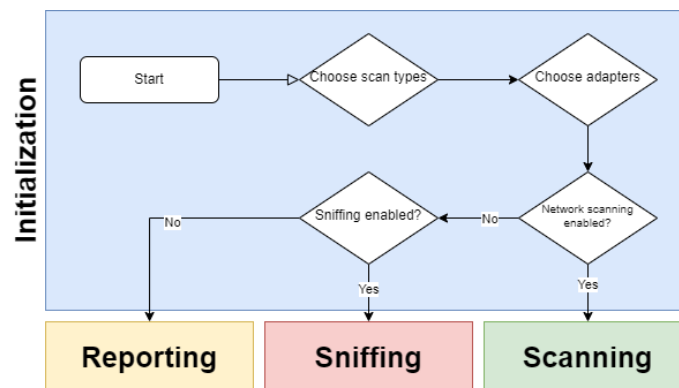


Figure 4.2: Design of the initialization phase.

Moving on to the scanning phase, depicted in Figure 4.3, it begins with an ARP scan that allows IoTective to quickly discover all live hosts by obtaining their IP and MAC addresses. Each IP address is then used as a target for an extensive nmap enumeration. This enumeration may take some time, depending on the connection speed. IoTective is configured to test the top 2000 ports of each host to identify running services. This number strikes a balance between an exhaustive list, which would be time-consuming, and allowing the discovery of more obscure ports, such as port 8123 used by Home Assistant. The enumeration also involves

guessing the host's operating system and determining the accuracy of that guess. Additionally, nmap runs a script called "vulners," which queries a remote 250 GB database of known CVEs using the Common Platform Enumeration (CPE) of the services running on the host.

Following device enumeration, IoTective performs functions specifically targeting the Philips Hue Bridge. It starts with an mDNS lookup, as this protocol is often used by IoT devices for mutual discovery and is recommended by Philips Hue¹. The subsequent step depends on whether the bridge was discovered. If it was, IoTective proceeds to fetch the configuration of the devices. If not, IoTective utilizes a GET request on the broker server URL (<https://discovery.meethue.com>), which returns the private IP address of any Philips Hue bridge on the network. The bridge configuration is then obtained by querying https://<BRIDGE_PRIVATE_IP_ADDRESS>/api/0/config using the acquired IP address.

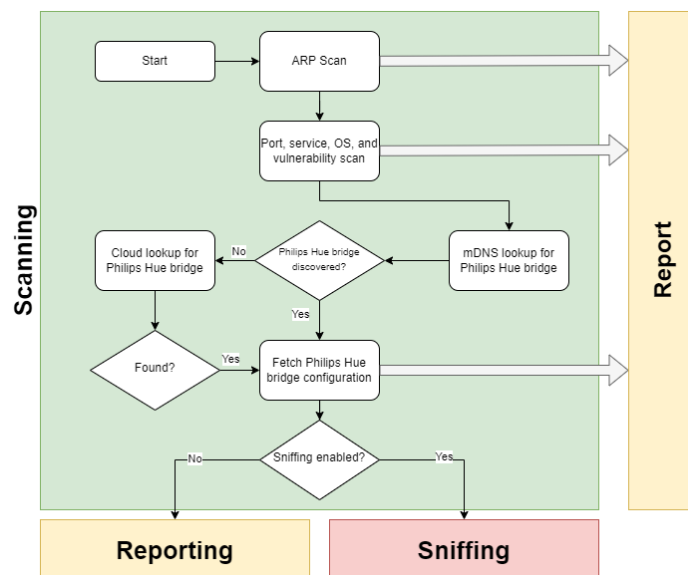


Figure 4.3: Design of the scanning phase.

If Wi-Fi, ZigBee, or Bluetooth sniffing is enabled, IoTective proceeds to the sniffing phase. Each of these parts is depicted in Figure 4.4. Let's start with Wi-Fi sniffing. IoTective captures packets based on the Extended Service Set Identifier (ESSID) of the connected access point. The ESSID is commonly known as the Wi-Fi network name, which is typically the default name set by the vendor or customized by the user. Capturing based on the ESSID, rather than just the Basic Service Set Identifier (BSSID) obtained from connections, is necessary because many access points support both 2.4 GHz and 5 GHz bands using the same ESSID. While the ESSID is the same for both bands, the BSSID is not. Since only the broadcast packets announce the ESSID, while the data packets only show the BSSID, we

¹How to develop for Philips Hue? <https://developers.meethue.com/develop/get-started-2/>

need all BSSIDs to discover all relevant packets. Once the BSSIDs are determined, IoTective captures data packets and filters them based on the BSSIDs. Any unique hosts communicating using these BSSIDs are assumed to be connected hosts.

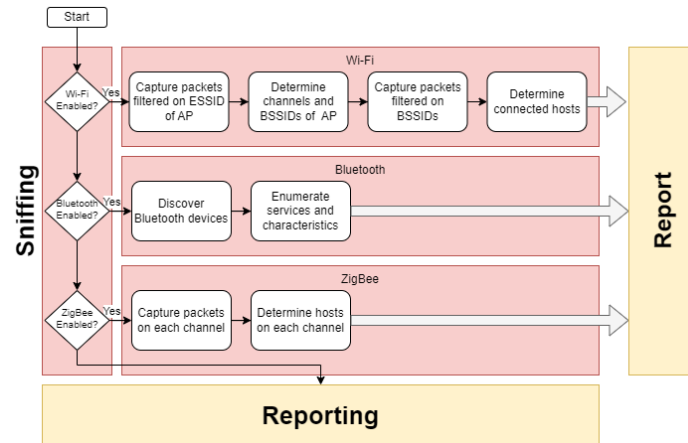


Figure 4.4: Design of the sniffing phase.

Bluetooth sniffing is a simpler task. First, a regular Bluetooth scan is performed, allowing IoTective to discover all Bluetooth devices in the vicinity. The next step is to connect to each device and attempt to gather additional information about the services and characteristics of the device.

ZigBee sniffing involves looping through each channel and logging each discovered device. This allows IoTective to determine hosts connected to each channel and their PAN ID. The information gathered during the sniffing phase is added to the report before proceeding to the final phase.

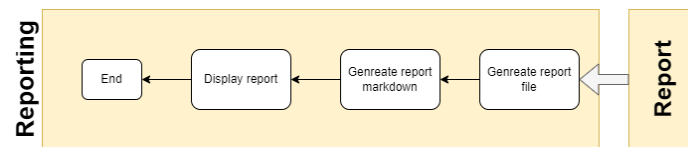


Figure 4.5: Design of the reporting phase.

Moving on to the reporting phase, depicted in Figure 4.5, it takes the report information as input and generates a JSON file. The information in the JSON report is then displayed using Markdown language. This allows the user to view the report in IoTective and review the home environment for further analysis. The reason to use Markdown language is that it allows the Python module "Texual" to automatically generate a navigation pane, making it easier for the user to move information and devices.

Chapter 5

Implementation

This chapter describes the developed tool and its functions. This includes describing the product's phases, the user experience, and the structure of the code. This chapter will give the reader insight into the code repository. This will aid any reader who wishes to help further develop the tool or review the tools underlying code.

5.1 Development Environment

The tool will be implemented using Python 3.11, which is the newest version of Python as of writing. Python is an excellent choice for developing tools for penetration testing due to its ease of use, flexibility, and abundance of libraries and frameworks available. Python's syntax is easy to read and write, making it easier to write and maintain code, especially for those who are new to programming. Its flexibility also makes it easy to integrate with other technologies, such as databases and web frameworks, allowing for the development of more complex systems. Moreover, the vast number of libraries available for Python makes it easier to develop tools quickly, without having to build everything from scratch. Additionally, Python is a popular language in the cybersecurity community, meaning there is a wealth of resources available for learning and problem-solving, such as online communities, documentation, and code examples. Overall, Python's simplicity, flexibility, and vast community make it an excellent choice for developing tools for penetration testing.

GitHub is a popular platform for hosting and sharing code, and will be used to host the code for the developed tool. The decision to make the code publicly available on GitHub is driven by the desire to encourage collaboration and feedback from the community of developers and users. The public repository will allow other developers to review, improve, and build upon the code, resulting in a more robust and feature-rich tool. Users will also be able to provide feedback and report issues, which can help to improve the tool's performance and usability. The use of GitHub also provides a transparent record of changes and updates made to the code, allowing users to track the evolution of the tool over time. Finally, by

making the code publicly available, other researchers and practitioners can benefit from the tool, potentially leading to further advancements in the field of smart home penetration testing [62].

The tool will be primarily developed for Kali Linux, however, since Python is Operating System (OS)-independent, it has the potential to be easily ported to other OSs as well. Kali Linux is a specialized Linux distribution that is designed specifically for penetration testing and security auditing. It comes pre-installed with a wide range of tools and utilities that are essential for security professionals and hackers alike. Developing a tool for Kali Linux ensures that it is compatible with the environment it is intended to be used in, and that it can leverage the full suite of tools and libraries available in Kali Linux. Additionally, Kali Linux is widely used in the industry and has a large and active community of developers and users, which means that the tool can benefit from the feedback, testing, and support of this community. By developing the tool for Kali Linux, it can also be easily packaged and distributed as a Kali Linux package, which makes it convenient for users to install and use. Developing the tool for Kali Linux can lead to a more efficient, effective, and widely adopted solution for penetration testing and security auditing.

The tool will be implemented with the assumption that the user has a wireless adapter that supports monitoring mode, a Zigbee adapter, and a Bluetooth adapter that supports monitoring mode. This allows the tool to perform wireless network analysis, Zigbee network analysis, and Bluetooth network analysis. The tool will provide the ability to capture and analyze network traffic, and to identify potential vulnerabilities in the wireless, Zigbee, and Bluetooth networks. The inclusion of Bluetooth support will enable the tool to scan for and analyze BLE devices, which are becoming increasingly popular in IoT applications. The tool will be designed to support a wide range of wireless, Zigbee, and Bluetooth adapters, and will provide guidance on adapter selection to ensure optimal performance.

The developed tool will be an open-source project, hosted on a public GitHub repository. This means that the source code will be freely available for anyone to access, use, modify, and distribute, under a permissive license. Open-source development brings numerous benefits to both developers and users, including transparency, collaboration, and innovation. By making the code open-source, other developers can review, improve, and build upon it, resulting in a more robust and feature-rich tool. The open-source model also fosters a community of users and developers who can provide feedback, suggest improvements, and contribute to the project. Furthermore, open-source tools are often free, making them accessible to a wider range of users. Finally, the open-source model ensures that the code remains available and can be modified as needed, even if the original developer is no longer actively maintaining the project.

5.2 Python Modules

As mentioned earlier, there are a lot of modules that are built to provide functionality related to performing security testing such as network scanning and sniffing. The following is a list of modules, excluding the built-in modules, used in this project to provide further functionality:

- **psutil**: A module that provides an interface for retrieving information about running processes and system utilization.
- **prettytable**: A module for creating visually appealing American Standard Code for Information Interchange (ASCII) tables from data.
- **python3-nmap**: A Python library that wraps the Nmap security scanner, allowing for network exploration and vulnerability scanning.
- **pyyaml**: A YAML parser and emitter for Python, enabling the reading and writing of YAML files.
- **bleak**: A module for interacting with BLE devices using the asyncio framework.
- **requests**: A versatile module for making Hypertext Transfer Protocol (HTTP) requests and handling responses.
- **zeroconf**: A module that provides support for mDNS/DNS-SD service discovery and registration.
- **scapy**: A powerful interactive packet manipulation module that allows for network scanning, capturing, and forging of network packets.
- **rich**: A library for rich text and beautiful formatting in the terminal, providing features like syntax highlighting, tables, and progress bars.
- **pyroute**: A Python library for network configuration and low-level interaction with network devices using the Linux kernel's netlink protocol.
- **textual**: A framework for building rich terminal applications with a focus on simplicity and extensibility.
- **zigpy_znp**: A module that extends Zigpy, a library for working with ZigBee devices, to support Texas Instruments Z-Stack ZigBee Network Processor (ZNP) interfaces.

When selecting Python modules for a development project, it is crucial to consider the popularity and recent development of the modules in question. The choice of modules should be based on their current usage within the community and their future potential for support and maintenance. This is because the selection of modules that are actively maintained and updated by a large user community provides several advantages. Firstly, these modules are more likely to remain compatible with future versions of Python and other libraries. Secondly, they are likely to have more bug fixes and feature updates available, making them more reliable and robust. Lastly, a strong community of contributors and users can provide helpful resources such as documentation, tutorials, and support forums, which can assist in the development process. Therefore, the choice of modules must be made with a long-term perspective in mind to ensure that the selected

modules remain relevant and useful for the intended purpose.

5.3 Code Structure

This section will describe how project code is structured. To use the tool, the user must run the main function called *iotective.py* as superuser. The main function runs the scripts located in the directories with names corresponding indicating the phase. The phases are as follows:

1. **Initialization:** Checks host capabilities and defines target network. If support for Wi-Fi, Bluetooth, or ZigBee sniffing is detected, the user will be asked what capabilities should be enabled.
2. **Scanning:** Performs host enumeration, port scanning, service and OS identification, and vulnerability detection.
3. **Sniffing:** Performs packet capture using Wi-Fi, Bluetooth, and ZigBee to gather a more complete picture of the network.
4. **Reporting:** Uses information gathered from phases 2 and 3 to create a report and display information to the user.

In addition, the directory "*app*" contains functions related to the graphical interface, making it easy for the user to configure the tool and to view reports after a scan has been performed.

5.4 Main Functions

This section describes some of the main functions of *IoTective* and explains how some of its features are implemented. To make it easier to explain the logic of the program, some simplified code snippets are used as reference.

5.4.1 Initialization

The initialization phase of a network security tool is a critical step that occurs prior to any data gathering from the network. During this phase, the tool conducts an assessment of the host's capabilities to determine whether it can support sniffing using various wireless communication protocols, including Wi-Fi, Bluetooth, and Zigbee. Based on this assessment, the user is presented with options to enable these functionalities. Additionally, the initialization script configures the target IP range, selects the appropriate interface for data transmission and reception, and verifies whether the script is being executed with administrative privileges or not.

Code listing 5.1: Initialization Data

```
now = datetime.now()

report = {
    "file_name": now.strftime("report_%Y-%m-%d_%H-%M-%S.json"),
    "start_time": str(now),
```

```

    "end_time": "",
    "config": config,
    "network_scan": [],
    "hue_bridge": [],
    "sniffing": {
        "wifi": {},
        "bluetooth": {},
        "zigbee": {}
    }
}

```

The "config" variable in Code Listing 5.1 refers to the configuration performed by the user, which tells IoTective which scan types to perform. The subsequent keys refers to the values where discovered hosts will be stored.

5.4.2 Scanning

The file `scanning/nmap.py` defines two functions that use the "nmap3" library to perform network scans on a specified target. The first function "arp_scan", shown in Code Listing 5.2, performs an ARP scan on the target to identify live hosts, represented as 'Host' objects. The function uses the "NmapHostDiscovery" class to perform the scan and checks if the hosts are live by checking if they have a state of 'up' and have a MAC address.

Code listing 5.2: ARP Scan

```

def arp_scan(target: str, logger) -> List[Host]:
    try:
        # Perform ARP scan using nmap
        nmp = NmapHostDiscovery()
        logger.info(f"Performing ARP scan on {target}...")
        result = nmp.nmap_arp_discovery(target=target, args="-sn")
        # Filter on live hosts
        live_hosts = [
            Host(
                ip=host,
                mac=host_info.get('macaddress', {}).get('addr', 'Unknown'),
                vendor=host_info.get('macaddress', {}).get('vendor', 'Unknown')
            )
            for host, host_info in result.items()
            if 'state' in host_info
            and host_info['state'].get('state') == 'up'
            and host_info['macaddress'] is not None
        ]

        if live_hosts:
            logger.info(f"Found {len(live_hosts)} live hosts")
        else:
            logger.info(f"Could not find any live hosts")

        return live_hosts
    # Exception handling...
    return []

```

The second function "port_scan" (Code Listing 5.3) performs a port scan, service scan, and vulnerability scan on the specified target. It uses the "Nmap" class

to perform the scan with specific arguments and returns the results as a dictionary. The logger parameter is used to log messages during the scan process. If an error occurs, the functions catch the exception and log the error message accordingly. The "typing" module is imported to specify the function parameter and return types. The "Host" class is imported from a module named "models" defined in a "models.host" package.

Code listing 5.3: Port Scan

```
def port_scan(target: str, logger) -> Dict:
    try:
        logger.info(
            f"Scanning_{target}_for_open_ports,"
            f"services_and_known_vulnerabilities..."
        )
        nmp = Nmap()
        arguments = "--open-T4-0--top-ports_10000--script_vulners"
        return nmp.nmap_version_detection(target=target, args=arguments)
    # Exception handling...
    return {}
```

After the initial scanning is performed, the next step uses vendor-specific techniques to look for a Philips Hue bridge. The script *scanning/hue.py* includes the function *discover_philips_hue_bridge()*. To search for Philips Hue bridge, it first attempts to discover it on the local network using a mDNS lookup on the address "*hue._tcp.local*". This allows us to discover the bridge without the need for it to be connected to the cloud.

Code listing 5.4: Philips Hue mDNS discovery

```
hue_scan = MdnsScan(service_type="hue")
hue_scan.scan()
discovered_bridges = hue_scan.get_devices()
```

The function named *MdnsScan()* (Code Listing 5.4) utilizes a dictionary of mDNS service addresses to query the relevant service. Upon receiving a response from the bridge, the IP address is obtained and subsequently used to extract the bridge configuration from the API. However, if no response is received from the mDNS query, the tool resorts to using Philips Hue's broker service to discover the IP. This process involves issuing a GET request to "https://discovery.meethue.com", which, if the bridge is present in the network, will return a JSON file containing information about the bridge.

After obtaining the private IP address of the Philips Hue bridge using the methods described earlier, the API can be queried for more information. The API is primarily intended for developers to create applications that can control the connected devices. However, to obtain an access token, the user must physically push the button on the bridge, which can be an obstacle in certain scenarios. Despite this, some information can still be obtained from the API without authentication. For instance, a configuration file containing details about the data store, API, and software version can be queried using a GET request to the URL "*https://<IP ADDRESS>/api/0/config*". In this URL, the private IP address of the bridge obtained

in the previous step is used. By analyzing the different versions, it is possible to look for related CVEs and determine whether the bridge has been patched or not.

5.4.3 Sniffing

The code showed in Code Listing 5.5 defines a function called "discover_bssids_on_ssid" that takes three parameters: "interface" is a string representing the name of the Wi-Fi interface to use, "ssid" is a string representing the ESSID of the Wi-Fi network to search for, and "logger" is an instance of the logging class that is used for outputting log messages. The function uses the scapy library to capture Wi-Fi packets on each channel of both the 2.4GHz and 5GHz bands, looking for beacons or probe responses that contain the target ESSID.

Code listing 5.5: Capture beacon frames to obtain BSSIDs

```
def discover_bssids_on_ssid(ssid: str, interface: str, logger) ->
Dict[str, List[str]]:
    bssids = {}
    channels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] # 2.4 GHz channels
    channels += [36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124,
128, 132, 136, 140, 149, 153, 157, 161, 165] # 5 GHz channels

    with Progress() as scanner:
        scan_task = scanner.add_task(
            f"[cyan]Discovering BSSIDs using '{ssid}' as SSID...",
            total=len(channels)
        )

        for channel in channels:
            # Change the channel of the wireless interface
            os.system(f"iwconfig_{interface}_channel_{channel}")

            # Define a packet handler function to extract MAC addresses
            def packet_handler(pkt: Packet):
                # Extract the source and destination MAC addresses from the packet
                if pkt.haslayer(Dot11)
                and pkt.haslayer(Dot11Elt)
                and pkt.info.decode() == ssid:
                    # Check if the packet is a Beacon frame or a Data frame
                    if pkt.type == 0 and pkt.subtype == 8:
                        # Extract the BSSID from the Beacon frame
                        bssid = pkt[Dot11].addr3
                        bssids.setdefault(bssid, [])

            scanner.update(scan_task, advance=1)

            # Sniff Wi-Fi packets for 2 seconds on the current channel
            try:
                sniff(prn=packet_handler, iface=interface, timeout=2)
            except Scapy_Exception as e:
                logger.error(e)

    return bssids
```

The "channels" variable contains a list of channels for both the 2.4 GHz and 5 GHz frequency bands. The function utilizes the "rich.Progress()" module, enabling the display of a progress bar to provide visual feedback to the user. Within the

function, there is a loop that iterates over each channel, allowing the wireless interface to listen on each channel sequentially. The packet handler specifically filters the packets based on IEEE 802.11 beacon frames. Each discovered BSSID is appended to the "bssids" variable, which is ultimately returned once the process is completed.

Following the BSSID discovery, *IoTective* employs the "discover_hosts_on_bssids()" function to iterate through each channel, identifying hosts based on the previously identified BSSIDs. Code Listing 5.6 closely resembles Code Listing 5.5, with the exception of the packet handler, which filters on data frames. It looks for new MAC addresses where the identified BSSIDs are either the source or destination address of the packet, indicating potential connected hosts.

Code listing 5.6: Packet handler identifying connected hosts

```
def packet_handler(pkt: Packet):
    # Extract the source and destination MAC addresses from the packet
    if pkt.haslayer(Dot11) and pkt.type == 2:
        # Extract the BSSID and source MAC address from the Data frame
        bssid = pkt[Dot11].addr3
        src_mac = pkt[Dot11].addr2
        dst_mac = pkt[Dot11].addr1
        if bssid in bssids:
            if src_mac == bssid and dst_mac not in bssids[bssid]:
                bssids[bssid].append(dst_mac)
            elif dst_mac == bssid and src_mac not in bssids[bssid]:
                bssids[bssid].append(src_mac)
```

ZigBee sniffing operates in a similar manner to Wi-Fi sniffing. *IoTective* utilizes the "discover_zigbee_devices()" function (Code Listing 5.7), which accepts parameters such as the logger, number of scans, and the path to the ZigBee adapter. Within this function, the "znp.connect()" method establishes a connection with the ZigBee adapter, while the "network_scan()" function conducts the sniffing process on each specified channel. The output of the function is a dictionary that maps channels to their respective hosts discovered during the sniffing process.

Code listing 5.7: ZigBee device discovery

```
async def discover_zigbee_devices(logger, radio_path: str, num_scans=6) ->
dict[str, list]:
    znp = ZNP(CONFIG_SCHEMA({"device": {"path": radio_path}}))
    await znp.connect()
    channels = t.Channels.from_channel_list(map(int, [11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26]))

    devices = await network_scan(
        duplicates=False,
        duration_exp=4,
        num_scans=num_scans,
        channels=channels,
        znp=znp,
        logger=logger
    )
    znp.close()
    return devices
```

Bluetooth enumeration involves two main steps: device discovery and service identification. In the "bluetooth_enumeration()" function shown in Code Listing 5.8, the first step is to identify all devices in the vicinity by executing the "scan_devices()" function. This function returns a dictionary where the device addresses serve as keys, and the corresponding values contain a dictionary with device information.

Once the devices are discovered, IoTective proceeds to gather additional information by attempting to establish connections with each device and identify the services available on them. This process involves querying the devices for their supported services and characteristics.

Code listing 5.8: Bluetooth device enumeration

```

async def bluetooth_enumeration(logger) -> dict[str, dict]:
    devices = await scan_devices(logger)

    if len(devices) > 0:
        with Progress() as scanner:
            scan_task = scanner.add_task(
                description="Fetching_device_services...",
                total=len(devices)
            )
            for address in devices:
                try:
                    scanner.update(
                        scan_task,
                        description=f"Fetching_services_for_device_{address}..."
                    )
                    devices[address]["services"] = await get_device_services(
                        address=address,
                        logger=logger
                    )
                    scanner.advance(scan_task, advance=1)
                except BleakError as e:
                    logger.error(e)
                    scanner.advance(scan_task, advance=1)
                    devices[address]["services"] = []
            logger.info(
                f"Gathered_information_about{str(len(devices))}_Bluetooth_devices."
            )
        return devices
    else:
        return {}

```

5.4.4 Reporting

The reporting phase consists of two primary components: generating the report file and displaying the report to the user. The report file generation process is straightforward. Python parses the dictionary initialized at the beginning into a JSON file, which is then stored in the "reports" directory within IoTective. The "reports" directory serves as a repository for all generated reports, which can be accessed by the user. By selecting a specific report, the user can view the information extracted from the JSON file, which is displayed in Markdown format.

Code Listing 5.9 presents the "create_report_information_markdown()" function, which is responsible for generating the report presented to the user. The "markdown" variable contains all the information that is displayed. Depending on the specified configuration, additional device information is included. IoTective employs the "textual" module to present the Markdown content in a user-friendly manner, enabling easy navigation between different sections of the report.

Code listing 5.9: Generation of Markdown language

```
def create_report_information_markdown(data: dict[str, any]) -> str:
    markdown = f"""
# Scan Information #

**Start Time:** {datetime.strptime(data['start_time'],
'%Y-%m-%d %H:%M:%S.%f').strftime('%A, %B %d, %Y %I:%M:%S %p')}

**End Time:** {datetime.strptime(data['end_time'],
'%Y-%m-%d %H:%M:%S.%f').strftime('%A, %B %d, %Y %I:%M:%S %p')}

## Configuration ##

**Interface:** {data['config']['interface']}
**IP Address:** {data['config']['ip_address']}
**Netmask:** {data['config']['netmask']}
**Network Scanning:** {'Yes' if data['config']['network_scanning'] else 'No'}
**WiFi Sniffing:** {'Yes' if data['config']['wifi_sniffing'] else 'No'}
**BLE Scanning:** {'Yes' if data['config']['ble_scanning'] else 'No'}
**Zigbee Sniffing:** {'Yes' if data['config']['zigbee_sniffing'] else 'No'}
**Zigbee Device Path:** {data['config']['zigbee_device_path']}

## Scan Summary

Network devices: {len(data["network_scan"])}
Wi-Fi devices: {len(data["sniffing"]["wifi"])}
Bluetooth devices: {len(data["sniffing"]["bluetooth"])}
ZigBee devices: {len(data["sniffing"]["zigbee"])}

# Devices #
"""
    if data['config']['zigbee_sniffing']:
        markdown += create_zigbee_markdown(data=data["sniffing"]["zigbee"])
    if data['config']['network_scanning']:
        markdown += create_network_scanning_markdown(data=data["network_scan"])
        if len(data["hue_bridge"]) > 0:
            markdown += create_hue_bridge_markdown(data=data["hue_bridge"])
    if data["config"]["ble_scanning"]:
        markdown += create_bluetooth_markdown(data=data["sniffing"]["bluetooth"])

    return markdown
```

Chapter 6

Proof-of-Concept

The Proof-of-Concept chapter plays a crucial role in this thesis as it demonstrates the practical implementation of the automated penetration testing tool in a simulated smart home environment. This chapter aims to showcase the tool's effectiveness in identifying vulnerabilities and weaknesses in the network, while also providing insights into the potential impact of these vulnerabilities if exploited by an attacker. The PoC was conducted in the environment described in Section 6.1, which provided a realistic and controlled testing environment. This chapter provides a detailed account of each step of the testing process, including the setup of the testing environment, execution of the tool, and analysis of the results. The insights gained from this PoC inform the conclusions and recommendations for future work presented in the subsequent chapters of this thesis.

6.1 Proof-of-Concept Environment

This section provides an overview of the home network environment utilized for testing the IoTective tool. Figure 6.1 illustrates the network diagram, demonstrating the central hub of the test environment, a router/Access Point (AP), which connects Wi-Fi and Ethernet devices.

The testing employed a Lenovo Yoga Slim 7 Pro 14IHU5¹ laptop running Windows 11 as the attacker machine. The laptop hosted a Kali Linux Virtual Machine (VM) on VMWare Workstation 17 Player². The Lenovo laptop was equipped with an Intel Core i5-11300H³ CPU and 16 GB of RAM. The Kali Linux VM connected to the AP via Wi-Fi using a wireless adapter supporting monitoring mode. ZigBee sniffing was performed using a TI CC2531⁴ device, which was flashed with custom

¹Lenovo Yoga Slim 7 Pro 14IHU5. https://psref.lenovo.com/syspool/Sys/PDF/Yoga/Yoga_Slim_7_Pro_14IHU5/Yoga_Slim_7_Pro_14IHU5_Spec.pdf

²VMWare Workstation 17 Player. <https://www.vmware.com/products/workstation-pro.html>

³Intel Core i5-11300H. <https://ark.intel.com/content/www/us/en/ark/products/196656/intel-core-i511300h-processor-8m-cache-up-to-4-40-ghz-with-ipu.html>

⁴Texas Instruments CC2531. <https://www.ti.com/product/CC2531>

firmware using a CC Debugger⁵. This enabled the utilization of the "zipbgy_znp" Python module. For Wi-Fi monitoring, a Realtek RTL8812AU⁶ adapter with monitor mode support was employed. Both Universal Serial Bus (USB) adapters were connected to the virtual Kali Linux machine.

The network environment consisted of various devices, including an Ethernet-connected Philips Hue Bridge, a Philips Hue Bloom light, a Philips Hue motion sensor, a Philips Hue smart plug, a Samsung TV, phones, and computers. Additionally, a Raspberry Pi was incorporated, running Home Assistant and Pi-Hole. Home Assistant controlled the Philips Hue Bridge and connected IoT devices, while Pi-Hole served as a network-wide ad and malware blocker. The Raspberry Pi operated on the Raspbian OS, and both Home Assistant and Pi-Hole were set up as Docker instances. This combination of devices and services created a more realistic smart home environment, enhancing the validity and relevance of the experimental results.

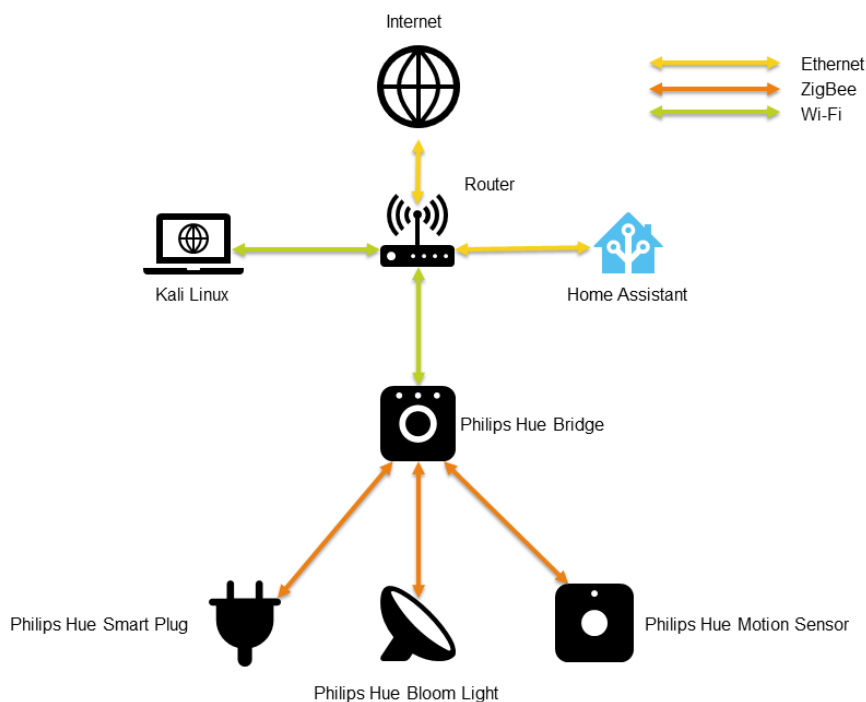


Figure 6.1: Smart home environment used for testing.

⁵CC Debugger. <https://www.ti.com/tool/CC-DEBUGGER>

⁶Realtek RTL8812AU. <https://zsecurity.org/product/realtek-rtl8812au-2-4-5-ghz-usb-wireless-adapter/>

6.2 Test Execution and Results

In this section, we run the tool in the previously described environment and discuss the results obtained from different stages, as well as how the user can interpret them.

6.2.1 Phase 1: Initialization

To test all the capabilities of IoTective, we enable all scan types, including network scanning (using nmap), Wi-Fi sniffing, Bluetooth scanning, and ZigBee sniffing. This allows us to gain a comprehensive view of the home environment. Figure 6.2 shows the configuration for the scan initialization.

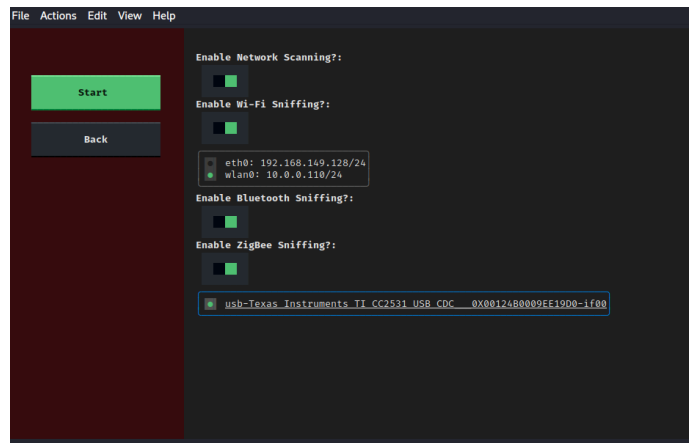


Figure 6.2: Scan initialization.

Network scanning means that the script will perform an ARP scan before using nmap to gather information about services, ports, OS, and known vulnerabilities associated with the services. Wi-Fi sniffing assumes that the chosen interface is associated with a wireless adapter that supports monitor mode and is connected to the target Wi-Fi network. It uses packet capture to determine which hosts are connected to the access point through Wi-Fi. Bluetooth scanning performs a regular Bluetooth scan, however it also performs the initial phase of a Bluetooth connection which allows us to gather information about services and characteristics. ZigBee sniffing uses packet capture to create an overview of which ZigBee devices are communicating on each channel.

6.2.2 Phase 2: Scanning

The scanning phase begins with an ARP scan, which is a network scanning technique used to identify and associate IP addresses with their corresponding MAC addresses in a local network. This allows for the rapid detection of all connected devices, and the MAC addresses can provide vendor information. The results of

the ARP scan are shown in Figure 6.3, which displays seven active hosts in the network along with their MAC and IP addresses. The initial six characters of the MAC addresses often indicate the vendor, providing additional information.

MAC	IPv4	Vendor
00:31:92:AC:04:C4	10.0.0.1	TP-Link Limited
F0:57:A6:67:C6:D1	10.0.0.141	Intel Corporate
E4:5F:01:BE:2D:A4	10.0.0.171	Raspberry Pi Trading
FA:EF:61:05:66:38	10.0.0.213	Unknown
A0:80:69:C4:EF:3C	10.0.0.218	Intel Corporate
BC:7E:8B:2E:B5:14	10.0.0.223	Samsung Electronics
EC:B5:FA:8E:9D:37	10.0.0.239	Philips Lighting BV

Figure 6.3: Result from ARP scan.

After identifying the connected hosts, the tool proceeds to perform a comprehensive nmap scan for each device. This scan gathers information about the operating system, open ports, running services on each port, and any known vulnerabilities associated with these services. The real-time scanning process provides continuous updates to the user. Figure 6.4 shows the scan results for the Raspberry Pi, a device running Home Assistant software for smart home device management. The scan reveals four open ports, with port 8123 being particularly noteworthy as it is commonly used by Home Assistant for web services employing HTTP. Additionally, the scan identifies 77 known vulnerabilities associated with the services running on this port. The subsequent sections will explore the report in detail to gain further insights into these vulnerabilities.

Host Information		Port Information			
Port	Service	Product	Version	CVEs	
Host Information Host: 10.0.0.171 MAC Address: E4:5F:01:BE:2D:A4 Vendor: Raspberry Pi Trading OS: Linux 4.15 - 5.6 Accuracy: 100 Type: general purpose					
22	ssh	OpenSSH	8.4p1 Debian 5+deb11u1	5	
53	domain	dnsmasq	pi-hole-v2.8...	0	
80	http	lighttpd	1.4.59	3	
8123	http	aihttpd	3.8.4	77	

Figure 6.4: Result from nmap scan of Raspberry Pi.

To provide a comparison, let's examine the findings from the scan conducted on the Philips Lighting BV host in the network shown in Figure 6.5. As expected, the host is identified as a Philips Hue bridge, as indicated by the OS identification labeling it as Philips Hue Bridge 2.0. The confidence level of this identification is 97%, reinforcing its accuracy. The host has open ports 80 and 443, indicating the

presence of a web server. However, it is important to note that web security testing falls beyond the scope of this tool. Encouragingly, no known vulnerabilities were detected in any of the services operating on the host.

```

[05/16/23 09:43:43] INFO Scanning 10.0.0.239 for open ports, services, and known vulnerabilities ... nmap.py:58
Host 10.0.0.239 Tue May 16 09:44:37 2023

Host Information
-----
MAC Address EC:B5:FA:8E:9D:37
Vendor Philips Lighting BV
OS Philips Hue Bridge 2.0 (Linux)
Accuracy 97
Type specialized

Port Information
-----
Port      Service Product      Version      CVEs
-----
80        http    nginx         Unknown      0
443       http    nginx         Unknown      0
8080      http    Web-based    Unknown      0
          Enterprise
          Management
          CIM
          access
          OpenPages
          WSEM
          httpd
  
```

Figure 6.5: Result from nmap scan of Philips Lighting BV host.

Once all the devices are enumerated, IoTective searches for a Philips Hue bridge on the network. Without prior knowledge of whether a bridge exists, the tool initiates an mDNS lookup using the address "_hue._tcp.local.," commonly associated with Philips Hue devices. In this case, the mDNS query did not yield a response. The tool then resorts to a cloud lookup by sending a GET request to <https://discovery.meethue.com>, which, if the bridge is connected to the cloud, returns the device's private network address. The bridge periodically queries the cloud to announce its private network address, facilitating cloud services in establishing a mapping between the household's public address and the bridge. This mapping ensures control over the device when connected via the public network.

Having confirmed the presence of a Philips Hue bridge on the network and obtained its associated private network address, the tool gains crucial information. Each Philips Hue bridge features an API that allows other applications, such as Home Assistant or custom-developed software, to control connected smart devices. Most API commands require authentication and authorization by the application. However, the "config" query serves as an exception, providing significant information about the bridge. This information is acquired by querying https://<BRIDGE_PRIVATE_IP_ADDRESS>/api/0/config, and the outcome is presented in Figure 6.6. The configuration information includes properties such as model ID, bridge ID, and name. Of particular significance for both the analyst and IoTective are the API version and software version, as certain CVEs are associated with specific versions of the API and software used by the Philips Hue bridge. Verifying the up-to-date and patched status of the API and software versions ensures protection against these CVEs, such as CVE-2020-6007 (buffer overflow vulnerability) and CVE-2017-14797 (deficiency in transport layer encryption).

```
INFO Successfully fetched bridge configuration.
```

Parameter	Value
api_version	1.57.0
bridge_id	ECB5FAFFFE8E9D37
cves	{'CVE-2020-6007': False, 'CVE-2017-14797': False}
datastore_version	155
internet	True
ip	10.0.0.239
mac	ec:b5:fa:8e:9d:37
model_id	BSB002
name	Philips hue
port	443
server	None
software_version	1957200040
type	None
weight	None

Figure 6.6: Configuration information gathered from Philips Hue bridge API.

6.2.3 Phase 3: Sniffing

After completing the scanning phase, IoTective proceeds to phase 3, which involves packet capture and scanning of Bluetooth, Wi-Fi, and ZigBee devices. The tool starts by performing Wi-Fi packet capture. Initially, it determines the BSSIDs of the access points to which the wireless adapter is connected. This is achieved by capturing IEEE 802.11 broadcast packets on each channel and filtering them based on the access point's ESSID. This approach is used to avoid relying solely on the BSSID provided during Wi-Fi connection, as many routers support both 2.4 GHz and 5 GHz bands with the same ESSID. Each band has its distinct BSSID, and filtering based on only one of them could potentially result in missing relevant packets. Once the BSSIDs are identified, the tool captures packets on each channel again, applying filters based on the BSSIDs to determine the band to which each connected device is connected. While the encrypted nature of the packets limits the acquisition of new information, the tool is able to establish a mapping between the MAC addresses and the addresses discovered during the scanning phase. It's important to note that in this run, the tool was only able to discover two out of four connected hosts, most likely due to the limitation of capturing packets for only 10 seconds on each channel, and not all devices may send packets within that timeframe.

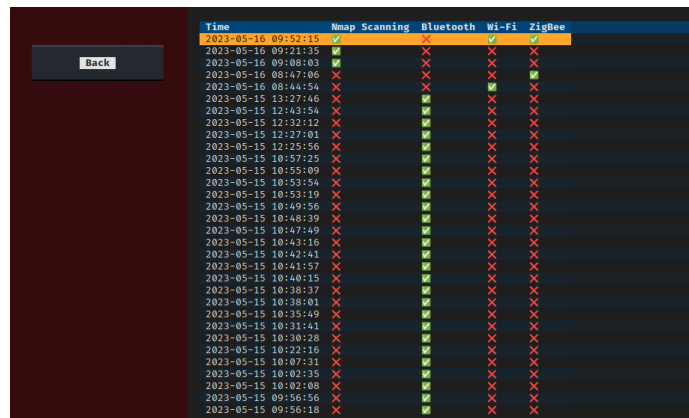
After identifying Wi-Fi hosts, the tool proceeds to perform Bluetooth scanning. This phase involves standard Bluetooth scanning procedures similar to those carried out by phones or computers. It gathers information such as device name, company name, local name, signal strength, and service UUIDs. Subsequently, IoTective attempts to establish a connection with each device to acquire additional information, including services and characteristics. However, certain devices may have security mechanisms that restrict connection establishment, such as devices requiring explicit user acceptance or activation of pairing mode. The scan discovered a total of 44 Bluetooth devices, with only three of them allowing access to information regarding services and characteristics.

Finally, the tool proceeds with ZigBee packet capture on each ZigBee channel to identify the devices associated with each channel. This phase requires a ZigBee

USB dongle capable of packet capture, such as the CC2531⁷. The dongle was flashed with custom firmware⁸ for this purpose. Although the encrypted nature of the packets limits the available information, it is still possible to determine the channel, protocol version, PAN ID, and whether the device allows joining. The scan discovered hosts on channels 15, 20, and 25, with a total of five devices.

6.2.4 Phase 4: Reporting

Once the scanning and sniffing phases are completed, IoTective provides users with a convenient way to access the scan results through the "View Reports" option. The reports are listed chronologically, with the most recent report displayed at the top, as shown in Figure 6.7. Each report includes the date, time, and enabled scan types for reference.



Time	Nmap Scanning	Bluetooth	Wi-Fi	ZigBee
2023-05-16 09:52:15	✓	✗	✓	✓
2023-05-16 09:21:35	✓	✗	✗	✗
2023-05-16 09:08:03	✓	✗	✗	✗
2023-05-16 08:47:06	✗	✗	✗	✗
2023-05-16 08:44:54	✗	✗	✓	✗
2023-05-15 13:27:46	✗	✓	✗	✗
2023-05-15 12:43:54	✗	✓	✗	✗
2023-05-15 12:32:12	✗	✓	✗	✗
2023-05-15 12:27:01	✗	✓	✗	✗
2023-05-15 12:25:56	✗	✓	✗	✗
2023-05-15 10:57:25	✗	✓	✗	✗
2023-05-15 10:55:09	✗	✓	✗	✗
2023-05-15 10:53:54	✗	✓	✗	✗
2023-05-15 10:53:19	✗	✓	✗	✗
2023-05-15 10:49:56	✗	✓	✗	✗
2023-05-15 10:48:39	✗	✓	✗	✗
2023-05-15 10:47:49	✗	✓	✗	✗
2023-05-15 10:43:16	✗	✓	✗	✗
2023-05-15 10:42:41	✗	✓	✗	✗
2023-05-15 10:41:57	✗	✓	✗	✗
2023-05-15 10:40:15	✗	✓	✗	✗
2023-05-15 10:38:37	✗	✓	✗	✗
2023-05-15 10:38:01	✗	✓	✗	✗
2023-05-15 10:35:49	✗	✓	✗	✗
2023-05-15 10:31:41	✗	✓	✗	✗
2023-05-15 10:30:28	✗	✓	✗	✗
2023-05-15 10:22:16	✗	✓	✗	✗
2023-05-15 10:07:31	✗	✓	✗	✗
2023-05-15 10:02:35	✗	✓	✗	✗
2023-05-15 10:02:08	✗	✓	✗	✗
2023-05-15 09:56:56	✗	✓	✗	✗
2023-05-15 09:56:18	✗	✓	✗	✗

Figure 6.7: List of all reports generated by IoTective.

By opening a specific report, users can view the information obtained during the scan. The report is generated using Markdown language and presents the data in a structured format. In the middle column, a table allows for easy navigation between devices. In this particular run, Bluetooth enumeration was performed separately, so the report states that Bluetooth scanning was not conducted. The scanning and sniffing processes took a total of 29 minutes and 55 seconds, with Wi-Fi and ZigBee accounting for 24 minutes and 41 seconds, while Bluetooth scanning took 4 minutes and 14 seconds.

Examining the ZigBee device section of the report reveals detailed information about each device, as presented in the table shown in Figure 6.9. This table includes relevant details such as the PAN ID, router and device capacity, Link Quality Indicator (LQI), and protocol version. By analyzing the channel number and PAN ID, analysts can gain insights into nearby ZigBee networks. If the analyst has a

⁷Texas Instruments CC2531. <https://www.ti.com/product/CC2531>

⁸Flashing the CC2531 USB stick. https://www.zigbee2mqtt.io/guide/adapters/flashing/flashing_the_cc2531.html

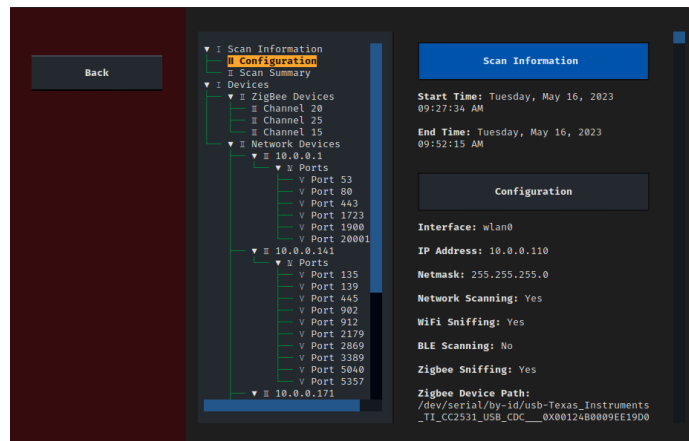


Figure 6.8: Scan information displayed in the report.

ZigBee device capable of packet injection, further active tests can be conducted using tools like KillerBee.

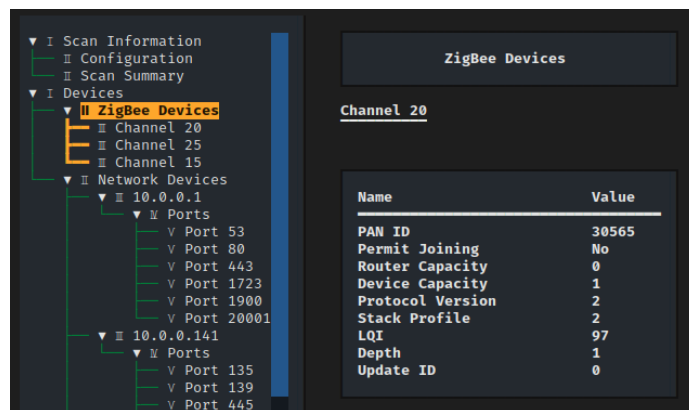


Figure 6.9: ZigBee device information displayed in the report.

Focusing on the network devices discovered by the network scanner, we can specifically examine the Raspberry Pi device. The Raspberry Pi provides us with the same information observed during the scanning process, as depicted in Figure 6.10. If the presence of a Raspberry Pi in the network was previously unknown, IoTective helps the analyst by providing valuable information for making such determinations.

During runtime, IoTective gains visibility into open ports on the host and the corresponding services running on those ports. For each open port, the report presents a list of potential vulnerabilities. Additionally, the report includes information on the CPE, which standardizes the naming of software applications, operating systems, and hardware platforms. In this context, the CPE refers to the service operating on the respective port. Analysts can utilize this information for further analysis, such as examining the source code of the software or identifying

Name	Value
IP Address	10.0.0.171
MAC Address	E4:5F:01:BE:2D:A4
Vendor	Raspberry Pi Trading
OS	Linux 4.15 - 5.6
OS Guess Accuracy	100
OS Type	general purpose

Figure 6.10: Raspberry Pi device information from the report.

relevant CVEs. However, IoTective already automates CVE lookup.

Name	Value
Port	8123
Protocol	tcp
Service Name	http
Product	aihttp
Version	3.8.4
CPE	cpe:/a:python:python:3.10

Figure 6.11: Information about port 8123 on the Raspberry Pi.

For each open port identified on the host, IoTective conducts a search for CVEs associated with the CPE of the corresponding services. The generated report includes a comprehensive list of these identified CVEs, accompanied by their respective Common Vulnerability Scoring System (CVSS) scores, which provide insights into the severity level of each vulnerability. In the case of the Home Assistant service running on the Raspberry Pi, IoTective detected 77 CVEs. Each CVE entry in the report, as depicted in Figure 6.12, showcases relevant information such as the CVE ID, CVSS score, vulnerability type, and an indication of whether an exploit for the known vulnerability exists.

Name	Value
ID	CVE-2016-5636
Is exploit?	Yes
CVSS	10.0
Type	cve

Figure 6.12: CVE-2016-5636 identified on the Home Assistant service.

One example is CVE-2016-5636, which relates to a vulnerability found in certain earlier versions of Python. This vulnerability involves an integer overflow issue that could potentially trigger a heap-based buffer overflow, enabling a remote attacker to exploit the vulnerability. It is important to note that although this CVE is associated with Python 3.10 in the identified CPE, it does not exclusively apply to the Home Assistant software implementation. Instead, it is relevant to any software utilizing Python as a programming language. Further investigation of the CVE ID reveals that this vulnerability only affects Python versions below 3.5.2, indicating that it does not pose a direct risk to the current Python installation used

by Home Assistant.

Figure 6.13 showcases the information gathered from a Sony WH-1000XM4 wireless Bluetooth headset. This data provides valuable insights for the analyst to understand the device's capabilities and functionalities.

Name	Value
Address	CB:BC:E6:3B:CD:93
Local Name	CB:BC:E6:3B:CD:93
Name	LE_WH-1000XM4
Company Name	Sony Corporation
RSSI	-50
Service UUID	0000fe03-0000-1000-8000-00805f9b34fb
Transmit Power	-21

Figure 6.13: Information gathered from Sony WH-1000XM4 headset.

The gathered information reveals that the device is a product made by the Sony Corporation with the name "LE_WH-1000XM4," which represents the simplified name of the product. The local name, which is usually a shorter or alternative name that can be changed by the user, remains unchanged in this case and is identical to the Bluetooth MAC address of the device.

In Bluetooth technology, Received Signal Strength Indicator (RSSI) indicates the signal strength on the receiving unit's antenna, providing an estimation of the signal's quality and proximity of the Bluetooth device. On the other hand, TX power refers to the amount of energy transmitted by the sending unit's antenna.

Service UUIDs, represented as 128-bit values, are used to uniquely identify different Bluetooth services or profiles offered by a device. Each Bluetooth device may support one or more services, and each service is identified by a unique UUID. When scanning a Bluetooth device, the list of service UUIDs provides information about the available services that the device supports. This information is useful for determining the capabilities and functionalities of the Bluetooth device.

```

● ** Google Inc. **
  Name: 0000fe2c-0000-1000-8000-00805f9b34fb (Handle:
  128): Google Inc.
  Characteristics:
    Vendor specific
      UUID: 00001235-0000-1000-8000-00805f9b34fb
      Handle: 132
      Properties: ['write', 'notify']
      Descriptors:
        ['00002902-0000-1000-8000-00805f9b34fb (Handle:
        134): Client Characteristic Configuration']
    Vendor specific
      UUID: 00001234-0000-1000-8000-00805f9b34fb
      Handle: 129
      Properties: ['write', 'notify']
      Descriptors:
        ['00002902-0000-1000-8000-00805f9b34fb (Handle:
        131): Client Characteristic Configuration']
    Vendor specific
      UUID: 00001236-0000-1000-8000-00805f9b34fb
      Handle: 135
      Properties: ['write']
      Descriptors:
        ['00002902-0000-1000-8000-00805f9b34fb (Handle:
        137): Client Characteristic Configuration']

```

Figure 6.14: Service information gathered from Sony WH-1000XM4 headset.

Figure 6.14 illustrates an example of one of the services identified on the Sony headset. Each service is accompanied by a "description," which in this case is "Google Inc." This could indicate that the headset has some sort of integration with Google, which aligns with the presence of Google Assistant as a feature on the Sony WH-1000XM4 headset.

Services further consist of "characteristics," which are individual data elements within a Bluetooth service that contain specific information or attributes. Both the service itself and each characteristic have unique UUIDs. An interesting attribute for the analyst to consider is the "properties" associated with each characteristic. These properties indicate the capabilities of each characteristic, specifying what actions can be performed, such as write, read, or notify. This information can be valuable for further testing, enabling the analyst to explore possible device modifications or gather additional information.

By examining the gathered Bluetooth device information, the analyst gains insights into the manufacturer, names, signal strength, transmission power, supported services, and their associated characteristics. This knowledge allows for a better understanding of the device's capabilities, potential integrations, and possible actions that can be performed with the Bluetooth device.

6.3 Analysis and Discussion

The report presents a comprehensive range of information gathered from various sources, protocols, and devices, showcasing the capabilities of IoTective. Network scanning tools like nmap provide valuable insights into the hosts connected to the local network, enabling initial vulnerability detection without disruptive techniques. Identifying the services running on each device empowers analysts with knowledge of potential weaknesses that can be further explored and exploited. For example, our analysis of the Raspberry Pi running the Home Assistant software allowed for additional research on specific CVEs and the exploration of potential exploits.

Wi-Fi sniffing proved effective in discovering devices that do not respond to ARP scans. In a specific test scenario, we encountered the Samsung TV, initially undetectable during ARP scanning, likely due to being turned off. However, through packet capture, we successfully captured its MAC address and determined its connectivity to the access point. While Wi-Fi sniffing can be time-consuming and may not capture packets from all devices within a limited time period, it serves as a valuable supplementary method for device discovery.

Similarly, ZigBee sniffing yielded positive results by allowing us to discover hosts in the area. However, determining the specific devices connected to our Philips Hue bridge required consultation of the settings in the Hue app. ZigBee sniffing provides insights into the presence of ZigBee devices but necessitates additional investigation to establish precise device associations.

Bluetooth scanning provides information about Bluetooth devices in proximity. However, the ability to gather information about the services and characteristics of the devices is limited to a subset of devices. In our analysis, we focused on a discovered Sony headset, acquiring insights into its services and characteristics. This information equips analysts with knowledge of available operations for further vulnerability assessment and analysis. Bluetooth scanning is most effective when analysts have specific indications of the devices to investigate, considering

the potentially extensive list of discovered devices.

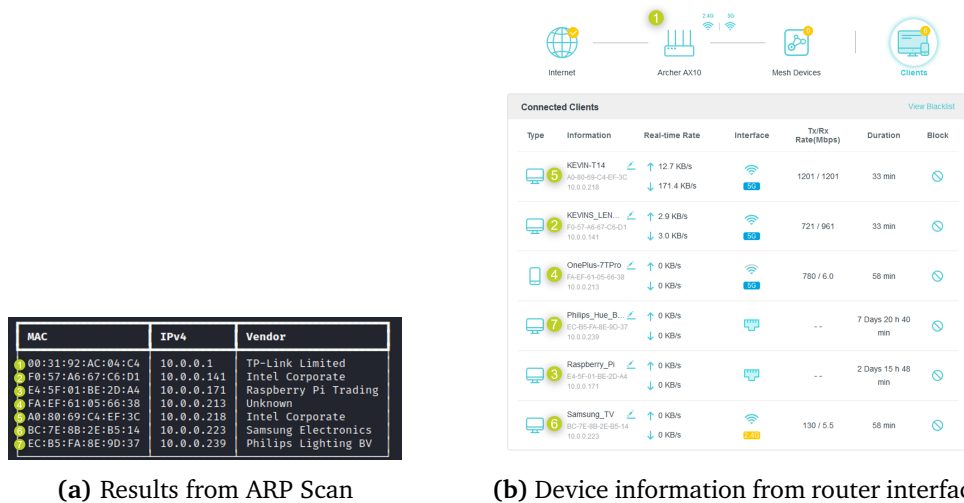


Figure 6.15: Mapping of discovered devices and devices connected to the router.

Figure 6.15 shows the mapping between the devices discovered by the ARP scan (Figure 6.15a) and the devices listed on routers interface (Figure 6.15b). IoTective managed to discover all devices on the network, while also giving information about what vendors the devices are associated to (with exception of the OnePlus phone).

Although the half-hour runtime may not be particularly fast, real-time information display allows for parallel manual analysis. The automated nature of IoTective reduces the need for human intervention, except for initial configuration. The generated report features an intuitive navigation system, enabling analysts to concentrate on advanced tasks based on device types.

Overall, the combination of network scanning, Wi-Fi sniffing, ZigBee sniffing, and Bluetooth scanning provides a comprehensive view of connected devices and their services. IoTective empowers analysts to conduct vulnerability assessments, discover hidden devices, and gain insights into potential security risks. Its automated features and intuitive report layout enhance efficiency and enable more in-depth analysis.

6.4 Conclusion

Through our PoC evaluation, IoTective distinguishes itself from other similar tools discussed in Section 3.2. While many tools focus on providing a wide range of features, relying on analysts to select and perform specific tasks, IoTective takes a different approach by emphasizing automation and user experience. It does not strive to be as powerful or thorough as some of these tools, such as KillerBee, but rather automates parts of the penetration testing process that requires minimal knowledge and significant manual labor. As a result, IoTective can be seen as a

valuable supplement to these other tools, which excel in more specialized and aggressive security testing. By automating repetitive and time-consuming tasks, IoTective streamlines the initial stages of IoT device assessment, enabling analysts to focus on more advanced security analysis and exploitation techniques.

Chapter 7

Discussion

This chapter aims to answer the research questions presented in the methodology chapter as well as discuss the result and limitations of the developed tool.

7.1 Research questions

This section discusses the research questions stated in section 1.3. The discussions will focus on automated penetration testing of smart home environments in general while the next section will discuss how the developed tool performed.

7.1.1 Vulnerabilities in Smart Home Environments

Research Question 1: *What are the most common vulnerabilities found in smart home environments, and how can they be exploited by attackers?*

The study aims to identify the most prevalent vulnerabilities in smart home environments and explore how attackers can exploit them. With the increasing popularity of smart home systems, homeowners are faced with new vulnerabilities and risks. The Open Web Application Security Project (OWASP) Internet of Things Top 10 provides a valuable framework for identifying common vulnerabilities in smart homes. Among the most prevalent vulnerabilities are insecure web interfaces, weak authentication and authorization, inadequate encryption and privacy, insecure network services, cloud interface security, and lack of physical hardening.

Insecure web interfaces pose significant risks in smart homes as they provide attackers with internet-accessible entry points to exploit vulnerabilities in web applications. These vulnerabilities can lead to unauthorized access to sensitive information or control over smart home devices. Weak authentication and authorization mechanisms are also common vulnerabilities resulting from devices or systems configured with easily guessable credentials or inadequate authorization protocols. Inadequate encryption and privacy measures can further compromise security when devices or systems transmit sensitive information without proper encryption or fail to protect user privacy.

Attackers exploit these common vulnerabilities in various ways, including exploiting vulnerable firmware or software to gain control over devices or networks, utilizing default or weak credentials to gain unauthorized access, intercepting or modifying data in transit by exploiting vulnerabilities in smart home protocols, or physically accessing the smart home network to install malware or tamper with devices.

Therefore, understanding and addressing these common vulnerabilities are crucial for homeowners to ensure the security and protection of their smart homes. Implementing strong authentication mechanisms, employing encryption and privacy techniques, and regularly updating firmware and software are effective measures to mitigate the risk of attacks and prevent unauthorized access to smart homes.

It is important to note that the OWASP IoT Top 10 was last released in 2018. Given the dynamic nature of the IoT domain, it is possible that the list has evolved in recent years. However, OWASP is a globally recognized project that extensively gathers insights from numerous sources to provide high-quality information about threats and vulnerabilities in various domains, including web, IoT, firmware, and APIs. Therefore, considering OWASP as one of the best sources for information about threats and vulnerabilities in a user-friendly format is reasonable.

7.1.2 Effectiveness of Automated Penetration Testing

Research Question 2: *How effective is automated penetration testing at identifying vulnerabilities in a smart home environment compared to manual testing?*

The effectiveness of automated penetration testing in identifying vulnerabilities in a smart home environment compared to manual testing is an important research question. In recent years, automated penetration testing has gained popularity due to its ability to efficiently and rapidly detect vulnerabilities in complex systems like smart homes. Automated testing tools simulate attacks and scan the system to identify potential vulnerabilities, allowing security analysts to prioritize and remediate them effectively.

One notable advantage of automated penetration testing is its ability to quickly detect a wide range of vulnerabilities. These tools can scan large amounts of code and network configurations within a relatively short timeframe, which is particularly advantageous in the context of smart home environments with numerous devices and configurations. Automated testing tools can also identify vulnerabilities that may go unnoticed in manual testing, including those that are not easily visible to the human eye or require specific testing scenarios. *IoTective*, for example, utilizes *nmap* for vulnerability detection, enabling analysts to identify potential weaknesses based on the software utilized by running services.

However, the effectiveness of automated testing largely relies on the quality of the test cases and the analyst's expertise. Automated testing tools can only identify vulnerabilities within the scope of the defined test cases, and if these cases are not comprehensive or fail to cover all possible attack scenarios, certain vulnerabilities

may be overlooked. Furthermore, the skill of the analyst using the tool plays a crucial role in determining the effectiveness of the testing process. Skilled analysts can customize the tool's settings to optimize its performance and interpret the results accurately. IoTective, while not exhaustive on its own, provides analysts with a valuable foundation to work with in conjunction with more powerful tools.

In complex smart home environments, manual testing is also essential to identify vulnerabilities that are too intricate for automated testing. Some vulnerabilities require a human touch to uncover, such as those resulting from a combination of factors or necessitating a deep understanding of the system's architecture. Manual testing aids in identifying business logic flaws that enable unauthorized access to sensitive information or allow attackers to manipulate the system in unintended ways. To address this limitation in automated penetration testing tools, it is crucial for analysts to employ diverse methods and tools to achieve comprehensive coverage.

In conclusion, automated penetration testing is effective in identifying a broad range of vulnerabilities in smart home environments, but its effectiveness is contingent upon the quality of the test cases and the skill of the analyst. Manual testing is indispensable for uncovering complex vulnerabilities that cannot be identified through automated means alone. IoTective solves the initial phases of information gathering and scanning, but further analysis is needed for the assessment to be comprehensive. Therefore, a combination of automated and manual testing is recommended for conducting thorough security assessments of smart home environments.

7.1.3 The Impact of Automated Penetration Testing on Security Posture

Research Question 3: *What impact does the use of automated pentesting have on the overall security posture of a smart home environment?*

The impact of automated penetration testing on the security posture of a smart home environment is a crucial research question. Our analysis reveals that automated penetration testing can significantly enhance the security posture by identifying vulnerabilities that could be exploited by attackers. Through vulnerability identification and the provision of remediation recommendations, automated penetration testing empowers homeowners and security professionals to take proactive steps towards securing their smart home environment.

One key advantage of automated penetration testing is its ability to decrease the likelihood of successful attacks. By pinpointing and addressing vulnerabilities, homeowners can minimize the attack surface of their smart home environment, making it more challenging for attackers to exploit weaknesses. Additionally, automated penetration testing can uncover security gaps such as misconfigurations or unsecured network connections that may go unnoticed in manual testing. While IoTective does not currently fulfill the complete potential of automated penetration testing, further development could bring it closer to becoming a comprehens-

ive tool.

Furthermore, automated penetration testing offers valuable insights into the security of the smart home environment over time. Regular automated tests enable homeowners and security professionals to monitor changes in the security posture and identify areas that may require additional security measures. This proactive approach helps prevent the introduction of new vulnerabilities as new devices and services are integrated into the smart home environment.

Overall, automated penetration testing has a positive impact on the security posture of a smart home environment. It identifies vulnerabilities, provides remediation recommendations, reduces the attack surface, reveals security gaps, and offers ongoing insights into the environment's security. However, it is important to acknowledge that automated penetration testing should not be the sole security measure in a smart home environment. Implementing other measures such as regular software updates, strong passwords, and network segmentation is essential to ensure comprehensive security.

7.1.4 Ethical Considerations in Automated Penetration Testing for Smart Homes

Research Question 4: *What are the ethical considerations that should be taken into account when conducting automated penetration testing in smart home environments?*

The fourth research question delves into the ethical considerations that must be taken into account when conducting automated penetration testing in smart home environments. The development and use of automated penetration testing tools have the potential to be beneficial for both legitimate users and malicious actors. Therefore, ethical considerations are paramount to ensure that these tools are utilized exclusively for ethical purposes. This entails obtaining user consent, preventing potential damage, and restricting access to the tool to authorized individuals only.

Automated penetration testing is a valuable means of identifying and mitigating vulnerabilities in smart homes. However, it is crucial to approach such testing with ethical considerations in mind, ensuring that it is conducted responsibly and ethically.

One fundamental ethical consideration is the risk of collateral damage. Automated penetration testing tools can inadvertently cause disruptions to network services or disable critical functions. It is essential to conduct testing within a controlled environment, minimizing the potential for unintended damage. This was a challenge during the development of *IoTective*, particularly in scoping the tool to specifically scan for relevant Bluetooth and ZigBee devices. Striving for automation, minimal user input was desired. However, filtering out devices outside the scope proved difficult due to limited information. As a compromise, *IoTective* collects information from all devices in the vicinity but does so with minimal disruption and focuses on authorized information from publicly available sources.

Protecting end-users' privacy is another critical ethical consideration. Automated penetration testing tools may collect sensitive information, such as passwords or personal data, during the testing process. It is imperative to ensure that any data collected remains confidential and is not used for unauthorized purposes. In the case of IoTective, no sensitive data is collected as it does not attempt to exploit target devices. Only authorized information is gathered and analyzed using publicly available sources.

Furthermore, automated penetration testing should only be conducted with explicit consent from end-users. Providing clear information about the purpose of the testing, potential risks involved, and the steps taken to minimize those risks is vital. End-users should have the option to opt-out of testing if they are uncomfortable with it. Additionally, testing should comply with relevant laws and regulations, such as data protection laws and regulations governing the use of automated tools for security testing.

Lastly, it is essential to ensure that automated penetration testing is conducted transparently and accountably. This includes documenting the testing process, the obtained results, and the steps taken to address identified vulnerabilities in a clear and accessible manner.

7.1.5 Heterogeneity of Smart Home Environments

Research Question 5: *How does the heterogeneity of smart home environments affect the effectiveness of automated pentesting, and what strategies can be employed to overcome these challenges?*

The fifth research question focuses on how the heterogeneity of smart home environments affects the effectiveness of automated penetration testing, and what strategies can be employed to overcome these challenges.

The heterogeneity of smart home environments can pose significant challenges when conducting automated penetration testing. Smart home environments may consist of different devices, communication protocols, and network architectures, making it difficult to ensure comprehensive coverage of all potential vulnerabilities. Additionally, these environments may have varying levels of security, with some devices or systems being more secure than others.

To overcome these challenges, penetration testing tools and techniques must be adaptable and flexible. Automated penetration testing tools should be able to identify and scan different types of devices and protocols, such as Zigbee, Z-Wave, or Wi-Fi. These tools should also be capable of testing different layers of the network stack, including the application layer, transport layer, and network layer.

Another strategy for overcoming the challenges posed by heterogeneity is to use a combination of automated and manual testing. While automated testing can help identify common vulnerabilities, manual testing can help identify more complex or unique vulnerabilities that may be missed by automated tools. Manual testing can also help provide more in-depth analysis of the results and identify

potential false positives or negatives.

Additionally, collaboration between security researchers, vendors, and end-users can help address the challenges of heterogeneity in smart home environments. Vendors can provide documentation and support for their devices and systems, while end-users can provide feedback on their experiences and identify potential vulnerabilities. Security researchers can help identify vulnerabilities and develop testing tools and methodologies that are tailored to the specific characteristics of smart home environments.

In conclusion, the heterogeneity of smart home environments presents significant challenges for automated penetration testing. However, with adaptable testing tools and a combination of automated and manual testing, these challenges can be overcome. Collaboration between different stakeholders can also help ensure that smart home environments are tested comprehensively and effectively.

7.1.6 Limitations of Automated Penetration Testing

Research Question 6: *What are the limitations of automated pentesting in smart home environments, and how can they be addressed to improve the effectiveness of the testing process?*

The final research question explores the limitations of automated penetration testing in smart home environments, and how they can be addressed to improve the effectiveness of the testing process.

Automated penetration testing is a useful tool for identifying vulnerabilities in smart home environments. However, it is not without its limitations. One of the main limitations is that automated tools may not be able to identify all types of vulnerabilities. For example, some vulnerabilities may require a human understanding of the system to identify, or may require physical access to the device. In addition, some vulnerabilities may be difficult to detect because they are not apparent from the device's software or firmware, but rather from the hardware design or configuration.

Another limitation of automated penetration testing is that it may not be able to assess the full impact of a vulnerability. For example, a vulnerability may be identified in one device or system, but it may also affect other devices or systems in the network. Additionally, automated tools may not be able to assess the potential impact of a vulnerability on the privacy or safety of the end-user.

To address these limitations, it is important to supplement automated penetration testing with other testing methods, such as manual testing and physical testing. Manual testing involves human experts who can assess the system from a different perspective and identify vulnerabilities that may be missed by automated tools. Physical testing involves testing the system in a real-world environment to assess vulnerabilities that may be difficult to identify in a simulated environment.

Another approach to addressing the limitations of automated penetration testing is to use a variety of automated tools. Different tools may have different strengths and weaknesses, and using multiple tools can increase the likelihood

of identifying a broader range of vulnerabilities. It is also important to keep tools and testing methodologies up to date, as vulnerabilities and attack techniques are constantly evolving.

In addition to using multiple testing methods and tools, it is also important to engage with the vendor community and to stay informed about new vulnerabilities and updates. Smart home devices and systems are constantly evolving, and vendors may release updates or patches that address known vulnerabilities. It is important to stay informed about these updates and to ensure that devices and systems are regularly updated and patched to ensure the security of the smart home environment.

7.1.7 Key Success Factors of Automated Penetration Testing Tools

Research Question 7: *What are the key success factors for the implementation of automated pentesting in smart home environments?*

The successful implementation of automated penetration testing in smart home environments requires consideration of several key factors. Firstly, it is important to select an appropriate testing tool that is capable of identifying vulnerabilities in a wide range of smart home devices and systems. The tool should also be regularly updated to ensure it is able to detect new vulnerabilities as they are discovered.

Secondly, the automated testing process should be complemented with manual testing by experienced security professionals. Manual testing can help identify vulnerabilities that automated tools may not detect, such as logical flaws or configuration errors. Additionally, manual testing can help validate the results of automated testing and ensure that vulnerabilities are accurately identified.

Thirdly, it is important to ensure that the automated testing process is conducted in a controlled environment that mimics real-world scenarios. This can help ensure that the testing accurately reflects the security risks faced by smart home devices and systems. In addition, it is important to obtain the necessary permissions and approvals before conducting any automated testing, to avoid legal or ethical concerns.

Finally, it is important to prioritize the vulnerabilities identified during testing based on their potential impact and likelihood of exploitation. This can help ensure that resources are allocated to address the most critical vulnerabilities first.

7.2 IoTective

In this section, we will explore IoTective and examine its functionality and performance. We will delve into how IoTective fulfills both functional and non-functional requirements, as well as the critical success factors that contribute to its effectiveness. Additionally, we will address the limitations of the tool to provide a comprehensive understanding of its capabilities and potential constraints.

7.2.1 Requirements

We will now review what functional requirements the developed tool managed to meet and how it compares to the key success factors of automated penetration testing. We will also discuss what shortcomings the tool has compared to those factors.

Functional Requirements

When examining the functional requirements, the primary objective is to ensure that IoTective operates in a fully automated manner, minimizing the level of human intervention required. The PoC clearly demonstrates how IoTective effectively fulfills this requirement by automating the discovery of network interfaces and ZigBee devices. This eliminates the need for users to manually enter device paths or specify target IP ranges, streamlining the scanning process.

IoTective provides users with the flexibility to choose the desired scan types, allowing analysts to have control over the specific scans they deem necessary for their assessment. This feature enables users to save time by skipping unnecessary processes and focusing on the most relevant aspects of their evaluation. By offering this level of customization, IoTective empowers analysts to tailor their scans according to their specific requirements and preferences.

Following the initialization phase, IoTective performs the configured scan types autonomously, executing the necessary operations to gather the required information. The tool then generates a comprehensive report, which can be conveniently accessed and viewed through the graphical user interface. This reporting functionality enhances the usability of IoTective, allowing analysts to easily review and analyze the results of their scans. This also adheres to the functional requirement regarding reporting, providing users with an intuitive way of navigating the collected information.

In addition to meeting the functional requirements, IoTective is designed to be applicable in various environments. Although it was developed using Python, which technically allows it to run on any operating system, IoTective has primarily been tested and optimized for Linux, specifically Kali Linux. Kali Linux is a popular choice among security analysts due to its comprehensive set of pre-installed security tools. IoTective leverages the capabilities of tools such as nmap, which is used for device enumeration. While IoTective currently relies on a few Linux-specific operations, it can be extended to work with other operating systems like Windows in the future, broadening its applicability.

Furthermore, IoTective exhibits vendor independence, except for the Philips Hue configuration fetching. This allows the tool to be deployed in a wide range of home environments, providing valuable information regardless of the type of hosts being analyzed. As long as the devices are connected to the same router as the analyzing host and utilize Wi-Fi, ZigBee, or Bluetooth protocols, IoTective is capable of performing enumeration and reporting on device information.

By supporting these protocols, IoTective can effectively function with various

smart home devices commonly found on the market. Bluetooth, Wi-Fi, and Zig-Bee cover a wide range of smart home devices, enabling IoTective to perform enumeration and reporting tasks effectively. However, it is important to note that IoTective currently lacks support for other protocols such as Z-wave, which is commonly used in smart home environments. Additionally, there are other less common radio communication protocols utilized in home environments that may not be supported by IoTective at present.

In summary, IoTective successfully meets the functional requirements by offering automation, customization, and comprehensive reporting capabilities. While it currently primarily supports Linux-based systems and certain protocols, efforts can be made to expand its compatibility to other operating systems and additional protocols in the future, further enhancing its versatility and applicability in diverse environments.

Non-Functional Requirements

IoTective strives to provide a good user experience by offering an intuitive and user-friendly interface. The graphical user interface allows users to interact with the tool easily and efficiently. It is designed to be visually appealing and requires minimal technical knowledge to operate. The tool's automation and customization features also contribute to its user-friendliness, as users can specify scan types and interfaces without delving into complex technical details. By presenting the results in a comprehensive and easy-to-navigate report, IoTective ensures that users can quickly understand and analyze the collected information, further enhancing the user experience.

Efficiency is a key consideration for IoTective. The tool is designed to perform tasks quickly and accurately, saving time and effort for the users. By automating the scanning process and utilizing efficient algorithms, IoTective optimizes the execution of scan types, minimizing unnecessary operations and focusing on relevant aspects. Additionally, the tool's integration with existing frameworks and libraries, such as nmap and scapy, enhances its efficiency by leveraging their optimized functionalities. The use of progress indicators and status updates during the scanning process also contributes to the perception of efficiency, keeping users informed about the progress and status of the tool's operations.

IoTective is developed with extendibility in mind, allowing for future improvements and the addition of further functionality. The tool is built using Python, a versatile and extensible programming language. Python's rich ecosystem and vast collection of libraries enable developers to easily integrate additional features and expand IoTective's capabilities. The modular design and well-organized codebase of IoTective facilitate the incorporation of new scan types, protocols, or reporting formats, making it relatively straightforward for developers to extend the tool's functionality to adapt to evolving security requirements and emerging technologies.

Accuracy is crucial for IoTective in order to provide reliable results. The tool

employs established scanning techniques and protocols to gather information from the target environment. By utilizing well-maintained and widely-used frameworks like nmap and scapy, IoTective leverages their robust functionality and algorithms, enhancing the accuracy of the scanning process. Additionally, IoTective employs packet filtering mechanisms and precise identification methods to ensure that only relevant packets and devices are captured and analyzed. This attention to accuracy helps to prevent false positives or misleading information, enabling users to make informed decisions based on reliable data.

Reliability is a fundamental aspect of IoTective. The tool is designed to be stable, ensuring consistent and accurate results across different scanning scenarios. Extensive testing and validation are performed to detect and address any potential issues or instabilities. Additionally, by leveraging established frameworks and libraries, IoTective benefits from their reliability and proven track record in the security community. The tool's error handling mechanisms and informative error messages also contribute to its reliability by providing users with clear indications of any issues encountered during scanning or reporting. This allows users to rely on IoTective's results and trust the tool's effectiveness in identifying potential security risks in IoT environments.

7.2.2 Key Success Factors

IoTective effectively addresses several key success factors, ensuring its capability as an automated penetration testing tool for identifying vulnerabilities in smart home devices and systems. These success factors were discussed in section 7.1.7 and are outlined below:

1. **Identify vulnerabilities in a wide range of smart home devices and systems:** IoTective is designed to identify vulnerabilities across various smart home devices and systems. By supporting protocols such as Wi-Fi, Bluetooth, and ZigBee, it covers the most commonly used communication technologies in smart home environments. This breadth of coverage enables IoTective to perform comprehensive scanning and analysis, effectively identifying potential vulnerabilities and security weaknesses across a diverse range of IoT devices. Furthermore, IoTective's extendibility allows for the incorporation of new protocols and scan types, ensuring its adaptability to emerging technologies and evolving security threats within the smart home ecosystem.
2. **Regular updates:** IoTective places great emphasis on remaining up-to-date with the latest security vulnerabilities and attack techniques. To address this, the tool is regularly updated to accommodate new vulnerabilities and maintain compatibility with changing device firmware and communication protocols. The development team actively monitors security advisories, research findings, and updates from relevant vendors and security communities. By incorporating these updates into the tool's codebase, libraries, and dependencies, IoTective ensures its continued effectiveness in identifying vulnerabilities and protecting against emerging threats. Additionally, the

integration of nmap's "vulners" script, which utilizes a remote database to search for CVEs, further enhances IoTective's ability to stay updated independently of the tool's development.

3. **Complementary manual testing:** While IoTective provides automated scanning and analysis capabilities, it acknowledges the importance of manual testing and expertise in security assessments. Recognizing that no automated tool can fully replace the value of human analysis, IoTective encourages the use of complementary manual testing techniques. Analysts can utilize manual testing to validate and further investigate the findings discovered by the automated tool. Manual testing allows for in-depth analysis, targeted attacks, and evaluation of complex scenarios that may not be covered by automated scanning alone. By combining automated and manual testing, IoTective enables security analysts to conduct a thorough assessment, gaining a comprehensive understanding of the security posture of smart home devices and systems.
4. **Prioritization:** Prioritizing vulnerabilities is crucial for efficiently allocating resources and addressing the most critical risks in a security assessment. IoTective incorporates prioritization features in its comprehensive scan results and reports. The tool identifies and categorizes vulnerabilities based on severity levels, impact, and exploitability. By highlighting high-priority vulnerabilities, IoTective empowers security analysts to focus their attention on the most critical issues and prioritize their mitigation efforts accordingly. This prioritization feature ensures that limited resources are effectively utilized, enabling timely and efficient resolution of potential risks. The tool's prioritization mechanism is implemented in the generated report, where CVSS scores are used to sort vulnerabilities based on severity.

By effectively addressing these key success factors, IoTective aims to provide a robust and effective automated penetration testing tool for identifying vulnerabilities in smart home devices and systems. Its adaptability, regular updates, integration with manual testing, and prioritization capabilities collectively empower security analysts to enhance their assessments and mitigate risks in IoT environments efficiently and effectively.

7.2.3 Tool Limitations

During the PoC, IoTective revealed certain limitations and challenges that provide valuable insights into the tool's capabilities and areas for improvement.

One significant limitation relates to the gathering of information about Bluetooth and ZigBee networks, as compared to Wi-Fi networks. Unlike Wi-Fi networks, which the user would typically connect to before performing the scan, Bluetooth and ZigBee networks operate differently. Consequently, IoTective resorts to capturing excessive data in order to gather as much information as possible. This approach leads to a substantial amount of captured data, which can be inefficient and time-consuming to analyze. Exploring more efficient solutions for capturing

Bluetooth and ZigBee network information would enhance the tool's performance and usability.

Another limitation concerns Bluetooth scanning, which can be problematic due to the scanner's attempts at establishing a connection with devices. This prompts users to accept the connection, potentially disrupting the testing process and raising suspicions. One possible solution could involve conducting a preliminary Bluetooth scan to gather device information, followed by allowing the analyst to selectively choose devices for connection attempts. This approach would provide greater control and minimize unnecessary disruptions during the scanning process.

Additionally, the current version of IoTective lacks certain features that could enhance its usability and user experience. For instance, although the tool provides an intuitive way to access the report, there is no feature for exporting the information in a readable format. This limitation restricts users from easily sharing their findings with other analysts or storing the information in alternative locations. The only option currently available is to manually copy the JSON file generated in the "reports" directory, but this file is not intended for use outside the tool itself.

Moreover, while IoTective aims to automate the enumeration phase of penetration testing and maintain a non-intrusive approach, it does not encompass all phases of the testing process. Notably, the tool lacks capabilities in the "exploitation" phase, which involves actively attacking devices to discover unknown vulnerabilities. Implementing attacks using custom code or leveraging frameworks like Metasploit, which can be utilized as Python modules, could expand IoTective's capabilities and enable the identification of previously unknown vulnerabilities.

Addressing these limitations and incorporating the suggested improvements would broaden the tool's functionality, usability, and effectiveness. By streamlining data capture, providing export features for reports, and enhancing capabilities in different phases of penetration testing, IoTective would offer a more comprehensive and robust solution for IoT security assessments.

7.2.4 Implications and Significance of Findings

The findings obtained through IoTective's scanning and analysis have significant implications for IoT security. The identified vulnerabilities and weaknesses underscore the potential risks associated with insecure smart home devices and systems. These vulnerabilities can expose sensitive user data, enable unauthorized access to devices or networks, and facilitate various types of attacks.

Understanding the significance of these findings in the context of IoT security is crucial. The vulnerabilities discovered by IoTective highlight the need for proactive security measures in the design, development, and deployment of smart home devices and systems. They emphasize the importance of proper device configuration, strong authentication mechanisms, encryption protocols, and secure communication channels. Addressing these vulnerabilities is vital to protect user privacy, prevent unauthorized access, and mitigate the potential consequences of

security breaches in the IoT ecosystem.

The research conducted using IoTective contributes to the broader field of IoT security by addressing existing gaps and challenges. By providing an automated penetration testing tool specifically tailored for smart home devices and systems, IoTective addresses the need for comprehensive security assessments in this rapidly expanding domain. The findings obtained through the tool's scanning and analysis contribute to the existing body of knowledge surrounding IoT vulnerabilities, weaknesses, and potential attack vectors.

Moreover, IoTective's ability to identify vulnerabilities, reveal critical security weaknesses, and prioritize remediation efforts offers valuable insights for security professionals, device manufacturers, and policymakers. It highlights the urgency of implementing robust security measures throughout the lifecycle of IoT devices, from design and development to deployment and maintenance. The research underscores the significance of securing smart home environments to safeguard user privacy, protect against potential attacks, and foster trust in IoT technologies.

Moving forward, future enhancements to IoTective could focus on addressing the limitations identified during the PoC, such as improving the efficiency of Bluetooth and ZigBee network data capture. Additionally, incorporating advanced manual testing features and techniques would further enhance the tool's effectiveness and usability, providing security analysts with comprehensive assessment capabilities.

Chapter 8

Conclusion

In conclusion, IoTective is a promising automated penetration testing tool for identifying vulnerabilities in smart home devices. Its comprehensive scanning and analysis capabilities address key success factors, including vulnerability detection, support for updates, and integration with manual testing. During the PoC, IoTective demonstrated its capacity to automate the discovery of network interfaces and ZigBee devices, enhancing usability through flexible scan options and user-friendly reports.

However, limitations and challenges were identified. Gathering information about Bluetooth and ZigBee networks posed difficulties due to the lack of a direct connection. The tool's current absence of export functionality for reports and limited coverage of all penetration testing phases, especially exploitation, suggest areas for improvement in efficiency and coverage.

Overall, IoTective provides valuable insights into IoT security by identifying vulnerabilities and highlighting critical weaknesses in smart home networks. Addressing the identified limitations and incorporating user feedback will enhance its effectiveness, contributing to ongoing efforts in securing IoT devices and networks.

As the field of IoT security evolves, IoTective's development and refinement contribute to the existing knowledge. Its focus on automation, adaptability, and integration with manual testing aligns with industry needs. Further research and development of IoT security assessment tools, considering the discussed limitations, will play a vital role in safeguarding the interconnected smart home ecosystem.

Chapter 9

Future Work

Based on the discussions and insights gained from this conversation, several areas of future work can be identified to further enhance and extend IoTective as an automated penetration testing tool for IoT security assessment:

1. **Optimizing Data Capture:** Investigate more efficient solutions for capturing Bluetooth and ZigBee network information. This could involve refining the packet capture process to reduce the amount of excessive data collected, improving data filtering techniques, and exploring ways to streamline the analysis of captured data.
2. **Export Functionality:** Develop an export functionality for the tool's reports, allowing users to save and share findings in a readable format outside of IoTective's directory. This would enhance collaboration among security analysts and enable integration with other security analysis tools or reporting systems.
3. **Enhancing User Experience:** Continuously improve the tool's user interface to enhance user-friendliness, intuitiveness, and visual appeal. Simplify the Bluetooth scanning process by providing options for selective device connection attempts, minimizing disruptions during the scanning phase.
4. **Expanding Exploitation Capabilities:** Consider implementing active attack capabilities within IoTective, using custom code or integration with frameworks like Metasploit. This would enable the tool to discover unknown vulnerabilities by performing targeted exploitation attempts on smart home devices, thereby providing more comprehensive insights into their security posture.
5. **Support for Additional Protocols:** Investigate the feasibility of incorporating support for additional communication protocols commonly used in smart home environments, such as Z-wave or other proprietary protocols. This expansion would broaden the tool's coverage and increase its effectiveness in identifying vulnerabilities across a wider range of devices and systems.
6. **Continuous Updates and Maintenance:** Establish a process for regular updates and maintenance to ensure IoTective remains up to date with the

latest security vulnerabilities, attack techniques, and changes in device firmware and communication protocols. Actively monitor security advisories, research findings, and updates from vendors and security communities to promptly address emerging threats.

7. **Performance Optimization:** Continuously optimize the tool's processing speed and accuracy to improve efficiency and reduce scanning and analysis time. This could involve optimizing algorithms, refining scanning techniques, and leveraging parallel processing capabilities to handle large-scale smart home environments more effectively.
8. **Integration with Reporting and Workflow Tools:** Explore integration possibilities with existing security reporting and workflow management tools. This would enable seamless integration of IoTective into larger security assessment frameworks and enhance the overall efficiency and productivity of security analysts.
9. **Expand Vendor-specific Testing:** By incorporating support for other vendors and their developer APIs, the tool can gather a broader range of valuable information for security analysis. This would involve developing testing methodologies tailored to each vendor's devices and protocols, allowing IoTective to target vulnerabilities specific to different IoT devices on the market.

By focusing on these areas of future work, IoTective can evolve into a more robust, efficient, and comprehensive automated penetration testing tool for IoT security assessments. These enhancements would further strengthen its ability to identify vulnerabilities, contribute to the advancement of IoT security practices, and assist in securing smart home devices and networks against emerging threats.

Bibliography

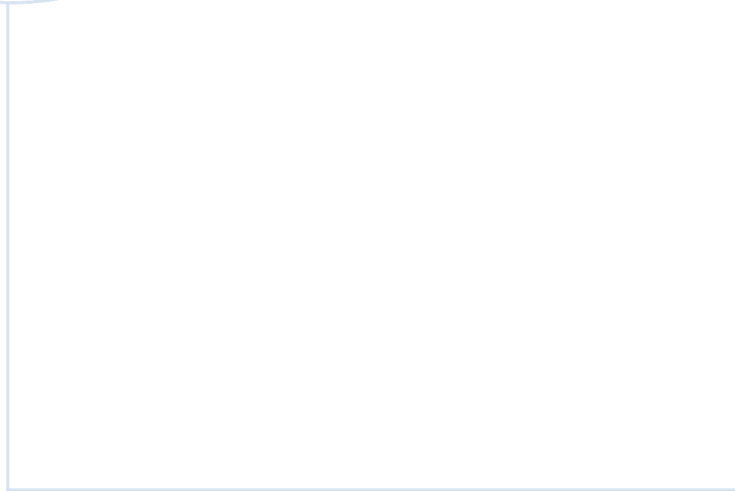
- [1] A. Alrawais, A. Alhothaily, C. Hu and X. Cheng, 'Fog computing for the internet of things: Security and privacy issues,' *IEEE Internet Computing*, vol. 21, no. 2, pp. 34–42, 2017.
- [2] S. Shah and B. M. Mehtre, 'An overview of vulnerability assessment and penetration testing techniques,' *Journal of Computer Virology and Hacking Techniques*, vol. 11, pp. 27–49, 2015.
- [3] *Openwips-ng*. [Online]. Available: <https://www.openwips-ng.org/>.
- [4] *Fing, Fing*. [Online]. Available: <https://www.fing.com/>.
- [5] *Wi-fi alliance*. [Online]. Available: <https://www.wi-fi.org/>.
- [6] *Ethernet alliance*, May 2023. [Online]. Available: <https://www.ethernetalliance.org/>.
- [7] CSA-IOT, *Csa-iot*. [Online]. Available: <https://csa-iot.org/all-solutions/zigbee/>.
- [8] B. T. Website, *Bluetooth technology website*. [Online]. Available: <https://www.bluetooth.com/>.
- [9] K. Rose, S. Eldridge and L. Chapin, 'The internet of things: An overview,' *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.
- [10] J. H. Nord, A. Koochang and J. Paliszkiwicz, 'The internet of things: Review and theoretical framework,' *Expert Systems with Applications*, vol. 133, pp. 97–108, 2019.
- [11] A. Khanna and S. Kaur, 'Evolution of internet of things (iot) and its significant impact in the field of precision agriculture,' *Computers and electronics in agriculture*, vol. 157, pp. 218–231, 2019.
- [12] T. Tekeste Habte, H. Saleh, B. Mohammad, M. Ismail, T. Tekeste Habte, H. Saleh, B. Mohammad and M. Ismail, 'Iot for healthcare,' *Ultra Low Power ECG Processing System for IoT Devices*, pp. 7–12, 2019.
- [13] F. Zantalis, G. Koulouras, S. Karabetsos and D. Kandris, 'A review of machine learning and iot in smart transportation,' *Future Internet*, vol. 11, no. 4, p. 94, 2019.
- [14] Microsoft, *Iot technologies and protocols*. [Online]. Available: <https://azure.microsoft.com/en-us/solutions/iot/iot-technology-protocols/>.

- [15] H. Touqeer, S. Zaman, R. Amin, M. Hussain, F. Al-Turjman and M. Bilal, 'Smart home security: Challenges, issues and solutions at different iot layers,' *The Journal of Supercomputing*, vol. 77, no. 12, pp. 14053–14089, 2021.
- [16] I. O. for Standardization, *Iso/iec 7498-1:1994*, Nov. 1994. [Online]. Available: <https://www.iso.org/standard/20269.html>.
- [17] S. C. Ergen, 'Zigbee/ieee 802.15. 4 summary,' *UC Berkeley, September*, vol. 10, no. 17, p. 11, 2004.
- [18] N. A. Somani and Y. Patel, 'Zigbee: A low power wireless technology for industrial applications,' *International Journal of Control Theory and Computer Modelling (IJCTCM)*, vol. 2, no. 3, pp. 27–33, 2012.
- [19] H. Zemrane, Y. Baddi and A. Hasbi, 'Comparison between iot protocols: Zigbee and wifi using the opnet simulator,' in *Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications*, 2018, pp. 1–6.
- [20] R. Heydon and N. Hunn, 'Bluetooth low energy,' *CSR Presentation, Bluetooth SIG*, 2012.
- [21] 'Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications,' *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021. DOI: 10.1109/IEEESTD.2021.9363693.
- [22] L. Tian, S. Santi, A. Seferagić, J. Lan and J. Famaey, 'Wi-fi halow for the internet of things: An up-to-date survey on ieee 802.11 ah research,' *Journal of Network and Computer Applications*, vol. 182, p. 103036, 2021.
- [23] E. Khorov, A. Kiryanov, A. Lyakhov and G. Bianchi, 'A tutorial on ieee 802.11 ax high efficiency wlans,' *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 197–216, 2018.
- [24] S. S. I. Samuel, 'A review of connectivity challenges in iot-smart home,' in *2016 3rd MEC International conference on big data and smart city (ICBDSC)*, IEEE, 2016, pp. 1–4.
- [25] G. Lobaccaro, S. Carlucci and E. Löfström, 'A review of systems and technologies for smart homes and smart grids,' *Energies*, vol. 9, no. 5, p. 348, 2016.
- [26] R. J. Robles and T.-h. Kim, 'Applications, systems and methods in smart home technology: A,' *Int. Journal of Advanced Science And Technology*, vol. 15, pp. 37–48, 2010.
- [27] S. Kumar, P. Tiwari and M. Zymbler, 'Internet of things is a revolutionary approach for future technology enhancement: A review,' *Journal of Big data*, vol. 6, no. 1, pp. 1–21, 2019.

- [28] F. Meneghello, M. Calore, D. Zucchetto, M. Polese and A. Zanella, 'Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices,' *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.
- [29] Y. Strengers and L. Nicholls, 'Convenience and energy consumption in the smart home of the future: Industry visions from australia and beyond,' *Energy Research & Social Science*, vol. 32, pp. 86–93, 2017.
- [30] K. Gram-Hanssen and S. J. Darby, "'home is where the smart is"? evaluating smart home research and approaches against the concept of home,' *Energy Research & Social Science*, vol. 37, pp. 94–101, 2018.
- [31] W. Li, T. Yigitcanlar, I. Erol and A. Liu, 'Motivations, barriers and risks of smart home adoption: From systematic literature review to conceptual framework,' *Energy Research & Social Science*, vol. 80, p. 102211, 2021.
- [32] B. K. Sovacool and D. D. F. Del Rio, 'Smart home technologies in europe: A critical review of concepts, benefits, risks and policies,' *Renewable and sustainable energy reviews*, vol. 120, p. 109663, 2020.
- [33] W. Ablondi and J. Narcotta, *2020 global smart home forecast - june 2020*, Jun. 2020. [Online]. Available: <https://www.strategyanalytics.com/access-services/devices/connected-home/smart-home/market-data/report-detail/2020-global-smart-home-forecast---june-2020>.
- [34] U. Singh, *What is Home Automation?* May 2021. [Online]. Available: <https://www.geeksforgeeks.org/what-is-home-automation/>.
- [35] M. Diaz, *The 6 best home automation systems of 2022*, Aug. 2022. [Online]. Available: <https://www.zdnet.com/home-and-office/smart-home/best-home-automation-system/>.
- [36] C. -. L. S. Committee, *Standard for information technology – telecommunications and information exchange between systems local and metropolitan area networks – specific requirements - part 11: Wireless local area network (lan) medium access control (mac) and physical layer (phy) specifications*, Feb. 2021. [Online]. Available: <https://ieee802.org/11/>.
- [37] *Home Assistant*. [Online]. Available: <https://www.home-assistant.io/>.
- [38] R. Langner, 'Stuxnet: Dissecting a cyberwarfare weapon,' *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.
- [39] N. Apthorpe, D. Reisman and N. Feamster, 'A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic,' *arXiv preprint arXiv:1705.06805*, 2017.
- [40] K. Chugh, L. Aboubaker and J. Loo, 'Case study of a black hole attack on lowpan-rpl,' in *Proc. of the Sixth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), Rome, Italy (August 2012)*, vol. 7, 2012, pp. 157–162.
- [41] *Flipper Zero*. [Online]. Available: <https://flipperzero.one/>.

- [42] C.-K. Chen, Z.-K. Zhang, S.-H. Lee and S. Shieh, 'Penetration testing in the iot age,' *computer*, vol. 51, no. 4, pp. 82–85, 2018.
- [43] T. O. W. A. S. P (OWASP), *Owasp iot top 10*, 2018. [Online]. Available: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project.
- [44] T. O. W. A. S. P (OWASP), *Owasp foundation, the open source foundation for application security | owasp foundation*. [Online]. Available: <https://owasp.org/>.
- [45] M. S. Wara and Q. Yu, 'New replay attacks on zigbee devices for internet-of-things (iot) applications,' in *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, IEEE, 2020, pp. 1–6.
- [46] T. Pan, 'Zigbee wireless network attack and detection,' in *Advances in Artificial Intelligence and Security: 7th International Conference, ICAIS 2021, Dublin, Ireland, July 19-23, 2021, Proceedings, Part III 7*, Springer, 2021, pp. 391–403.
- [47] N. Hussein and A. Nhlabatsi, 'Living in the dark: Mqtt-based exploitation of iot security vulnerabilities in zigbee networks for smart lighting control,' *IoT*, vol. 3, no. 4, pp. 450–472, 2022.
- [48] S. Okada, D. Miyamoto, Y. Sekiya and H. Nakamura, 'New ldos attack in zigbee network and its possible countermeasures,' in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, 2021, pp. 246–251.
- [49] P. Morgner, S. Matthejat, Z. Benenson, C. Müller and F. Armknecht, 'Insecure to the touch: Attacking zigbee 3.0 via touchlink commissioning,' in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017, pp. 230–240.
- [50] N. I. of Standards and Technology, *Cve-2022-39064 detail*, Oct. 2022. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2022-39064>.
- [51] M. Cäsar, T. Pawelke, J. Steffan and G. Terhorst, 'A survey on bluetooth low energy security and privacy,' *Computer Networks*, vol. 205, p. 108 712, 2022.
- [52] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sun and E. Kurniawan, 'Sweyntooth: Unleashing mayhem over bluetooth low energy,' in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, 2020, pp. 911–925.
- [53] A. Barua, M. A. Al Alamin, M. S. Hossain and E. Hossain, 'Security and privacy threats for bluetooth low energy in iot and wearable devices: A comprehensive survey,' *IEEE Open Journal of the Communications Society*, 2022.
- [54] B. D. Davis, J. C. Mason and M. Anwar, 'Vulnerability studies and security postures of iot devices: A smart home case study,' *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 102–10 110, 2020.

- [55] N. I. of Standards and Technology, *Cve-2018-7580 detail*, Dec. 2020. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2018-7580>.
- [56] N. I. of Technology and Standards, *Cve-2020-6007 detail*, Jan. 2020. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2020-6007>.
- [57] N. I. of Technology and Standards, *Cve-2022-47100 detail*, Jan. 2023. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2022-47100>.
- [58] B. Cankar, B. Bingöl, D. Çavdaroğlu and E. Çelebi, *Github - yakuza8/peniot: Peniot: Penetration testing tool for iot*, Jun. 2019. [Online]. Available: <https://github.com/yakuza8/peniot>.
- [59] A. Jakhar, A. Magesh, F. Affolter, S. Rajguru, A. Thusoo and D. Hinge, *Explotiot*. [Online]. Available: <https://explotiot.io/>.
- [60] P. González, J. E. García and L. Fernández, *Homepwn - swiss army knife for pentesting of iot devices*. [Online]. Available: <https://github.com/Telefonica/HomePWN>.
- [61] J. Wright, R. Speers and R. Melgares, *Killerbee*. [Online]. Available: <https://github.com/riverloopsec/killerbee>.
- [62] K. Nordnes, *Iotective: Smart home penetration testing tool*, GitHub repository, 2023. [Online]. Available: <https://github.com/kevnor/IoTective>.



 **NTNU**

Norwegian University of
Science and Technology