Bachelor Degree Project

Testing IoT Security

- A comparison of existing penetration testing frameworks and proposing a generic framework

Authors: William Mattsson, Alva W. Palmfeldt
External Supervisors: Amer Hodzic, Jawdat Kour
Supervisor: Arslan Musaddiq
Semester: VT 2023
Subject: Computer Science

Abstract

The Internet of Things (IoT) refers to the billions of physical devices linked to the Internet worldwide, integrating into various systems like healthcare, finance, and transportation. However, the rapid market expansion has led to software and hardware security shortcomings, leaving IoT devices vulnerable to cybercriminals. The security can be maintained and evaluated in different ways, nonetheless, this thesis focuses on investigating the process of a penetration test to identify vulnerabilities present in IoT devices. This paper investigates and compares existing penetration testing frameworks and proposes a generic testing framework for IoT. The results show that there is no standardized penetration testing framework to target IoT devices, as there are for networks and the web. By defining IoT-specific testing methodologies, our research shows that common IoT vulnerabilities could be identified and exploited.

Keywords: Internet of Things, penetration testing, standardization, framework, vulnerability exploitation

Preface

We would like to thank TietoEvry in Kalmar, Sweden for giving us access to some IoT devices to penetration test and for their enthusiastic communication. We especially thank Amer Hodzic and Jawdat Kour for welcoming and helping us with the project. We would also like to thank our supervisor from Linnaeus university, Arslan Musaddiq for helping and contributing to the project. At last, we would like to thank all the other teachers that we have met during our study period at Linnaeus University, especially our program coordinator Ola Flygt.

Contents

1 Introduction

The phrase "Internet of Things" (IoT) denotes the billions of physical devices that are now linked to the Internet from all over the world. These devices are becoming ubiquitous and are being integrated into critical systems such as healthcare, finance, and transportation. However, they are not limited to enterprise settings as they can be found in an everyday home with applications like smart lamps, energy-meters and other forms of home automation. These interconnected devices are forecast to grow to a number of about 30 billion in the next few years [1].

The revolution of IoT comes with numerous benefits for all different sectors as well as for personal use. However, due to the rapid expansion of the market it has led to IoT development shortcomings, more specifically in the software and hardware security features. Developers may focus more on implementing as many end-user features as possible for a better selling point incentive, rather than focusing on securing existing features. IoT devices collect, process, and transmit vast amounts of sensitive data which make them prime targets for cybercriminals. As they are such a big part of our existing IT infrastructure, there is a great need to secure them. Not doing so causes a looming financial and privacy risk for both home users along with the industries that are dependent on IoT [2].

By simulating an attack on an IoT device, penetration testing helps identify vulnerabilities and weaknesses in the device's security before malicious actors can exploit them. This initiative-taking approach to security helps to prevent data breaches, protect end-users, and secure critical infrastructure. Maintaining and conducting regular penetration testing on IoT devices is a crucial step in ensuring the security of these systems.

This thesis investigates and compare existing penetration testing frameworks to see if they can be applied to IoT devices.

To continue, the proposed research advocates for the development of a comprehensive penetration testing framework tailored to IoT , grounded in the comparative analysis results of pre-existing frameworks. Upon its completion, the efficacy of this novel framework will be assessed by applying it to a diverse array of IoT devices to ascertain their susceptibility to prevalent IoT vulnerabilities.

While it is important to clarify that this framework does not seek to provide an exhaustive solution for penetration testing, it aims to fill notable gaps observed in the currently available frameworks. The key areas of improvement have been identified and will be incorporated into the new framework to enhance its effectiveness and efficiency in conducting rigorous penetration testing on IoT devices.

1.1 Background

The growth of IoT devices has led to an exponential increase in the number of interconnected devices. While these devices have changed the way we live and work by being used in a wide range of public facilities, a challenge is to maintain security and privacy. To establish communication between trusted parties, it is necessary to maintain authorization and authentication over a secure network, which IoT devices do not always conform to [3].

As mentioned in the introduction, one way to evaluate the security of these devices is to perform penetration testing. The purpose of a penetration test is to identify potential risks that an attacker may exploit when gaining access to an organization's computing system and networks. By conducting a penetration test, organizations can close security gaps before a real attack occurs. This proactive approach helps organizations minimize the risk of financial and information loss, which in turn protects their customers as well as the organization itself [4].

To ensure that the penetration testing is conducted in a standardized and effective manner, a penetration testing framework specific for IoT devices is needed. Shanley and Johnstone [5] define a "framework" in their study as: “encapsulates methodology and methodology encapsulates tools, techniques, and resources”.

We will undertake a comparative analysis of existing methodologies and standards as they pertain to the conceptualization of the term ’framework’. Further, we aim to determine whether these standards incorporate a dedicated methodology and the requisite security instruments explicitly designed for conducting penetration testing on IoT devices.

1.2 Related work

There are quite a few reports that focus on creating frameworks for penetration testing devices in general. However, most papers are for network connected devices that are using IEEE 802.3 & IEEE 802.11 (I.e., the Ethernet and Wi-Fi standard) and do not focus on IoT devices or networking specifically. IoT uses a broad set of different interfaces and networking standards, hence the need for a specific research papers on the matter.

There are three reports that focus specifically on the subject of creating penetration testing frameworks for IoT devices. "Penetration Testing Framework for IOT" [6], "IoT-PEN: A Penetration Testing Framework for IoT" [7] and "IoT-PEN: An E2E Penetration Testing Framework for IoT" [8]. Do note that, most of the authors are the same across all the published papers.

The first paper proposes an "end to end" penetration testing framework for IoT devices. This framework is supposed to run both manually with a user following the framework and automatically with program algorithm. The report explains that the framework can be applied to all parts of the device including physical hardware, firmware, user software along with the networking for cloud connectivity.

The second and the third paper proposes a "plug and play" framework that can be applied to any IoT device. Said framework is hosted on a server and the IoT devices function as "clients". While not specified in the reports themselves, this framework only seems to work over IEEE 802.3 and 802.11. This means it leaves out networking interfaces such as LoRa, Bluetooth Low Energy (BLE). All three research papers does not contain that much technical information and only scratch the surface of IoT security [8].

Shanley and Johnstone, in their work "Selection of penetration testing methodologies: A comparison and evaluation" [5], present a comparison of security frameworks, suggesting future research should validate these frameworks using real-life scenarios. Our thesis, to the contrast aims to assess and adapt these frameworks, originally designed for non-IoT systems, networks, and web applications, for IoT devices.

1.3 Problem formulation

A lot of IoT devices found on the market come with security and privacy issues. While this puts confidentiality at risk for anyone who might use such devices, it also means that these devices can be an entry point for cybercriminals into a more confidential system. The rapid development and production of these devices means that they may not have the same security standards implemented as found on traditional electronic devices. While there are some IoT security guidelines today, they were nonexistent before 2017 [9].

Computer systems, the Internet, and networks have been around for years, and security experts have developed many different standards and frameworks to both secure and test these technologies. The penetration testing frameworks, on the other hand, have never been reviewed in terms of how effective they are at conducting successful testing operations on IoT devices. To continue, the research on how these frameworks work in real-life scenarios has gained little or no interest. By comparing the existing frameworks, it would be possible to identify both what methodologies and tools are effective as well as if there is any missing information when penetration testing the IoT domain. Given the well-documented common vulnerabilities of IoT devices, our investigation also examines the exploitability of these weaknesses across a selection of devices.

- **RQ1**: How does the existing standards/framework apply to IoT penetration testing?
- **RQ2**: How does a generic penetration testing framework aid in identifying common vulnerabilities?
- **RQ3**: What are the specific challenges when testing IoT devices?

1.4 Motivation

The motivation of this work is to compare existing penetration testing frameworks in terms of how well-suited they are when applied to IoT devices. Doing so would aid the creation of a generic penetration testing framework for IoT, which is a field with little current research. Additionally, we were motivated to create this framework as there is no recent research paper or organization that applies this approach to IoT with examples. With a proper framework it would be possible to identify vulnerabilities and mitigate the security misconfigurations before the malicious actor takes advantage of it.

There are many implementation standards intended to strengthen the security for IoT devices across many different application areas [10]. This includes but is not limited to: Industrial, medical monitoring as well as more personal sectors such as smart home appliances and home automation. The current state of the IoT industry threatens the security of modern society by not prioritizing the security measures needed to be a secure system [10]. As such testing these devices is important as it safeguards the organizations and individuals against both financial loss and loss of integrity [4].

1.5 Results

The expected results of this study are aimed at providing confirmation of the suitability of security testing frameworks for targeting IoT systems. Additionally, it is presumed to provide useful information regarding IoT security, security testing, and the unique challenges when testing these types of devices. The expected results are outlined below:

1. Confirmation of the suitability of security testing frameworks for IoT devices.
2. Creation of a new generic penetration testing framework specifically for IoT devices, taking into account their unique testing challenges.
3. An illustration of identifying IoT vulnerabilities by using the created framework as guidance. This with a comparison between the vulnerabilities detected and if they can be classified as common vulnerabilities for IoT.

1.6 Scope/Limitation

The methodologies/standards to be compared in this study has been carefully selected by reviewing online sources on what is used within the penetration testing industry [11, 12, 13, 5].

The comparison section of this research is limited to these four different standards that can be used in a penetration test: Open Source Security Testing Methodology Manual (OSSTMM) [14], Penetration Testing Execution Standard (PTES) [15], OWASP for IoT [16] and NIST Special Publication 800-115 [17].

1.7 Target group

This report targets individuals with intermediate IT security knowledge, aiming to guide them in applying their skills to real-world IoT device penetration testing. It also serves IoT researchers and developers by identifying potential exploits and addressing issues in current testing standards and or frameworks.

1.8 Outline

The structure of this report is as follows: Chapter 2 addresses the research methodology, research questions, and ethical considerations. Chapter 3 imparts the theoretical foundation and explicates technical terms pertinent to the research. Chapter 4 offers a succinct comparison of various security testing frameworks, theoretically evaluating their applicability to IoT targets. Chapter 5 introduces the developed generic framework, accompanied by IoT-specific testing methodologies and tools. Chapter 6 presents test cases guided by the aforementioned generic framework. Chapter 7 incorporates the results and analyzes the outcomes of the framework comparison, test cases, and challenges inherent in IoT device penetration testing. Discussions of the findings are housed in Chapter 8, with Chapter 9 concluding the report and outlining future work.

2 Method

This section of the paper offers a detailed description of the scientific methods used in the study. It outlines how the results were used to ensure the accuracy and reliability of the findings. The section also discusses the ethical considerations that were considered during the research and highlights the ethical implications of the study's results.

2.1 Research Project

This research uses a multi-method scientific process. The process consists of three phases that are conducted to support the design science research methodology [18]. It is used as a baseline to create the new artifact, a proposed generic framework for IoT penetration testing.

The three main phases start with a literature review to introduce the concepts of the research area, such as IoT technologies and their challenges, along with a definition of penetration testing methodologies. The second phase is a comparative study between the existing penetration testing standards and how they apply to IoT penetration testing. At last, the evaluation part is done by an experiment where the methodologies of the created artifact are used as guidance for a real-life penetration test.

The following subsections (2.2–2.5) show the different activities of the design science research methodology including the literature review.

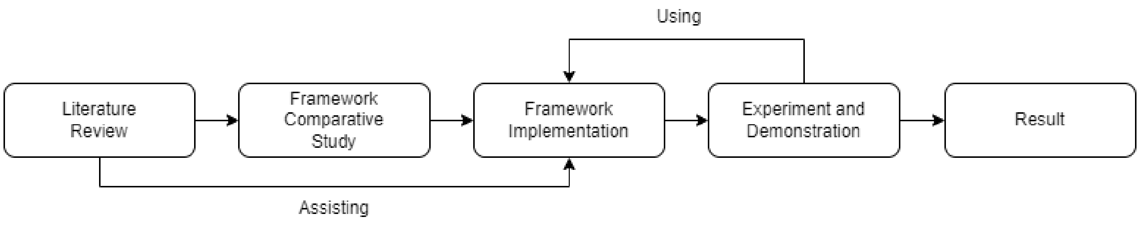


Figure 2.1: Research project process

2.2 Literature review

To locate scholarly articles pertinent to the domain of IoT security and testing, an extensive search was conducted utilizing academic databases, such as Google Scholar, IEEE Xplore, and OneSearch. This research were not restricted solely to academic publications but also encompassed grey literature, which includes digital blogs and institutional reports related to the subject matter, thereby enriching the research with a diverse range of perspectives. The chosen keywords for finding relevant scientific papers were "IoT", "Security", "Architecture", "Vulnerability", "Penetration testing", "Framework" and "Hacking". The aim of the literature review were to introduce the concepts of the research area and to form the theoretical background. Additionally, the literature review aid in finding IoT vulnerabilities and testing methodologies that were included in the created proposed framework.

2.3 Define the objectives for a solution

The objective of this work was to improve the effectiveness of IoT penetration testing by creating the grounds for a penetration testing framework. The methodologies and tools included in the framework are specifically intended for testing the security of IoT devices. Such a framework would be useful when evaluating these devices, as the existing frameworks and standards were originally intended for non IoT computer systems, networks, and web applications.

2.4 Design and development

To move from the objectives to design, information was gathered by doing a comparative study on the existing security standards and frameworks. To continue, two out of the four selected frameworks were theoretically evaluated on how they could be applied to test the most common IoT vulnerabilities. We argue that this approach help to identify limitations of the included frameworks. Identifying both the limitations and features of the existing frameworks contribute to creating the new artifact. Furthermore, reviewing other's work combined with active investigation of the current attacks vectors and vulnerabilities help to create methodologies along with the appropriate application suite.

2.5 Demonstration and evaluation

The research demonstrates and evaluates the effectiveness of the created artifact by using its methodologies as guidance in an experiment that involves performing penetration testing on a set of devices.

2.6 Reliability and Validity

The cases that were evaluated when using the framework were limited to a small set of devices. This means that all the different protocols and technologies that various IoT devices use could not be covered. By conducting an experiment with a larger set of devices that differs in the protocols and techniques used, it may have shown if there were any lacking areas in the implementation of the artifact.

To create the artifact, only peer-reviewed articles and industry standards have been examined. Moreover, the included methodologies and tools are not newly created, rather they are already existing in the industry but now customized for IoT testing specifically.

To achieve reasonable and reliant results we tried to select devices that differ as much as possible in architecture and protocols used. However, due to the lack of resources and available devices, these were limited to certain network standards.

2.7 Ethical considerations

There are ethical considerations when working with this research. As this work is within IT security, it is important to discuss the ethics of vulnerability disclosure, decompilation, and reverse engineering. If vulnerabilities are found using this proposed generic framework with the given test cases, then they need to be disclosed privately to the manufacturers of said IoT devices within a reasonable time frame. While there is no industry standard for disclosure time, at least 3 months is often recommended. This generic IoT penetration testing framework discuss IoT client applications and device firmware decompilation for security analysis. As such, it is important to stay within the legal boundaries and the end-user licensing agreement of the IoT product. This framework is not designed, nor shall it be used for malicious purposes. As such, this proposed generic framework will not discuss device persistence mechanisms for when a valid vulnerability has been exploited.

3 Theoretical Background

This section informs the readers about the technical terms and technologies used in this research. It is intended to give an overview of the IoT architecture and its security challenges. Secondly, penetration testing as a process and the existing standards are to be presented for the reader.

3.1 Internet of Things

IoT refers to the interconnected web of physical devices which come equipped with sensors, software, and various other technologies. These features enable them to communicate and share data with other devices and systems via the internet or the local network. However, there is no universal definition for an IoT device is or how it needs to function [19].

Kevin Ashton invented and presented the term "Internet of Things" for the first time in 2009. Ashton described it as: "*If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing, or recalling, and whether they were fresh or past their best.*" [20]. Today, the IoT technologies can be found in many different applicable domains such as smart cities, industrial automation, healthcare, agriculture, smart homes and more [21].

3.1.1 IoT architecture

As of right now there is no standardized architecture for IoT devices. This is due to how IoT devices may differ in terms of software, performance, and hardware requirements. As such, there are three different major architecture classifications that are circulating. These are the three, four and five layered architecture models [22].
The three-layered architecture approach is described below.

- Perception. This layer contains the sensors and its respective generated data that are connected to the IoT device.
- Network. This layer describes the protocols and connectivity that is used between the IoT device and the receiving backend. This can for example be an IoT device sending data to a external server which stores the sensor data.
- Application. This layer describes the front-end and application logic that the user interacts with. This can for example be a smartphone application that is connected to the IoT device.

The three-layer architecture does describe most IoT devices. However, it is limited in scope, especially when it comes to the network part of the layer which has led to the creation of the four and five layered architecture. The four and five layered architecture differs between articles. The IoT architecture study by Zhong, et al, [22] explains a four- and five-layer architecture. In the five-layer architecture, the network and application layer are separated into two subcategories.

- Perception layer goes unchanged.

- Network access layer defines the preliminary processing done to the collected data in order to convert it to the needed protocol.
- Network transmission layer describes the data being transferred by different networking protocols. E.g., Zigbee, Wi-Fi and more.
- Application support layer describes all the in-between technology used to process and deliver the data to the front end of the application. This can for example be specific queries sent to a database, data processing, middleware data processing etcetera.
- Application presentation layer describes how the managed data is presented to the end user. This can be for example with a user mobile application, web application etcetera.

MongoDB's documentation describe the same three-layer architecture and also a five layered architecture that focuses less on networking and more on who uses the data and how it is used [23]. These five layered architectures are described below:

- Perception layer goes unchanged from the three layer approach.
- Transport layer describes the transfer of data between the sensors between the perception layer and the processing layer
- Processing layer which describes the processing needed for the collected data coming from the transport layer. This can for example be filtering, sorting, data compression, etcetera.
- Application layer which goes unchanged from the three layer approach.
- Business layer which describes the human element of the device and is above the application layer.

3.1.2 Security Challenges

The proliferation of smart IoT technologies and their integration with various devices amplifies potential entry points for hackers within a system or network. IoT device security is further complicated by their distinct design and functionality compared to traditional computing devices [19]. As IoT systems are often embedded in mission-critical solutions, such as pacemakers, insulin pumps [24], and other vital monitoring systems, ensuring their security and reliability is of paramount importance and a subject of extensive discussion [19].

To continue, due to the variety of technologies and protocols used in IoT devices another challenge is **Insufficient IoT Device Testing And Updates**. It becomes difficult to integrate and manage all devices securely which leads to security shortcomings. Additionally, many IoT devices are not configured to have the ability to be upgraded or the upgrading process is often a difficult and time-consuming process. In the same context, IoT devices often have unpatched vulnerabilities due to the problems with the upgrading and patching process [25] [19].

The Internet society brings up an example in their paper "The Internet of Things (IoT): An Overview":

"Consider the 2015 Fiat Chrysler recall of 1.4 million vehicles to fix a vulnerability that allowed an attacker to wirelessly hack into the vehicle. These cars must be taken to a Fiat Chrysler dealer for a manual upgrade, or the owner must perform the upgrade themselves with a USB key. The reality is that a high percentage of these autos probably will not be upgraded because the upgrade process presents an inconvenience for owners, leaving them perpetually vulnerable to cybersecurity threats, especially when the automobile appears to be performing well otherwise."[19]

Another security challenge is the **Physical Security** of IoT devices as they are likely to be deployed in areas where the physical access is often hard to monitor [19]. This would lead to physical-based and proximity attacks that could compromise the system and its integrity [26]. Without any physical protection an attacker could get full access to the device.

IoT devices with more powerful compute power often have a large attack surface due to their networking interfaces. For example, attackers can gain access to devices by exploiting vulnerabilities in the user-end software that interacts with the device directly. Once an attacker gains access to said device it can give the ability to work as a jumping-off point to gain access to other devices, effectively creating persistence within the target network. Attackers can also use IoT devices to launch distributed denial-of-service (DDoS) attacks, flooding servers with traffic and causing them to crash. Figure 3.2 shows an overview of the IoT security challenges based on reviewed sources [19, 25, 27].

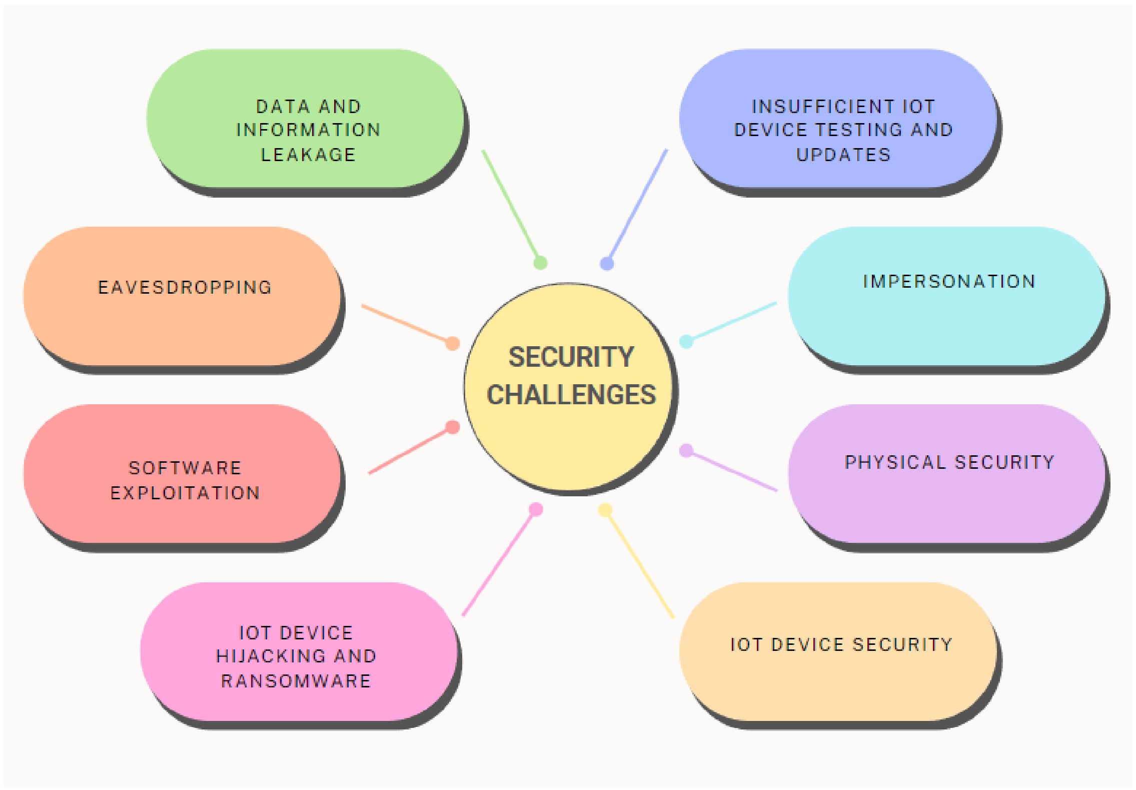

DATA AND INFORMATION LEAKAGE
INSUFFICIENT IOT DEVICE TESTING AND UPDATES
EAVESDROPPING
IMPERSONATION
SECURITY CHALLENGES
SOFTWARE EXPLOITATION
PHYSICAL SECURITY
IOT DEVICE HIJACKING AND RANSOMWARE
IOT DEVICE SECURITY

Figure 3.2: IoT security challenges: An overview

Data and information leakage: Security breaches are likely to occur in IoT networks if data and information are not secured by having appropriate security measures that can malicious intruders from accessing and leaking sensitive information [27].

Eavesdropping (MITM attacks): One common attack and challenge in IoT networks is the man-in-the-middle attack. Due to the IoT connectivity models, criminals can exploit an insecure communication channel to capture traffic and thereby steal and or modify transmitted information [2, 27].

Software Exploitation: With poor security mechanisms, malicious software or code can easily be injected to firmware upgrades, trusted boot along with other applications and services running on the device [27].

IoT device hijacking and ransomware: In the same way as data and information leakage can occur, poor security implementations could lead to devices being hijacked and exposed for ransomware and other malicious payloads [27].

Impersonation: Many devices have poor authentication and password handling such as weak, guessable, hardcoded passwords [2, 13, 25, 27]. This results in an uncomplicated way for malicious intruders to impersonate legitimate users [27] and in that way access sensitive information, install malicious code or even escalate to higher privileges.

IoT Device Security: Even though there are standards for securing IoT devices [10], manufacturers still struggle to implement valid security mechanisms [27]. As stated before, you can often find weak or hardcoded passwords, unpatched operating systems, and insecure communication channels.

3.2 Penetration Testing

A penetration test refers to the process of identifying security vulnerabilities in devices and systems before an outside attacker manages to exploit them [4]. Penetration tests are typically conducted by trained and certified security professionals who use a variety of tools and techniques to simulate real-world attack scenarios. These scenarios could include social engineering, phishing attacks, network scans, vulnerability scans, and more. Furthermore, a penetration test is often divided into these three distinct types of operations:

Black-box Penetration testing: A testing operation where the tester has no insight on how the system works and how it is built. This means that the tester can verify contradictions in the actual systems, but often these tests are hard to design [28].

White-box Penetration testing: A testing operation where the tester has complete access to the system which can be for example: an IP-address, source code, operating system details and more. A white-box penetration test is often referred to an internal attack where information is known [28].

Grey-box Penetration testing: A testing operation that is a combination of the black and white box which entails that the tester is provided limited information about the system. A grey-box penetration test is often referred to as an attack where the external hacker has gained information about the infrastructure [28].

3.2.1 Framework definition

Penetration testing standards and methodologies exist to help organizations and individuals identify security vulnerabilities. If a penetration testing methodology is agreed upon, it will help testers to narrow down the scope to be even more effective.

As mentioned earlier the chosen definition of framework is the following:

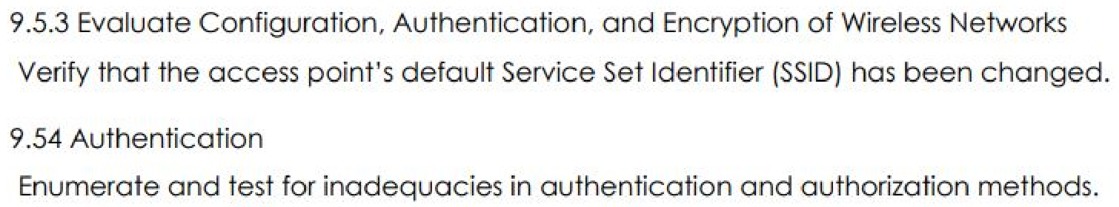
> "*A framework encapsulates methodology and methodology encapsulates tools, techniques, and resource*" [5].

The following standards and methodologies are guidelines that can help with various aspects of a penetration test. As all of them can be used in a penetration test, they will be included in the theoretical comparison. Even though the standards/methodologies refer to themselves as frameworks, each of them may not be appropriate for the definition of framework in this research.

3.2.2 Open Source Security Testing Methodology Manual (OSSTMM)

The Open Source Security Testing Methodology Manual (OSSTMM) is an open-source and peer reviewed project that defines guidelines and best practices for evaluating the security of information systems, networks, and applications by preforming a "OSSTMM test" [14].

OSSTTMM is primarily developed as a security auditing methodology to help organizations meet the security standards and regulations set by various regularities and industries. It covers a wide range of security testing areas such as human security testing, physical security testing, wireless security testing, telecommunications security testing and data networks security testing. However, it does not contain methodologies or tools on how to evaluate specific cases. The manual is more designed to help testers to identify areas that should be tested and measures to verify compliance with laws and regulations [14]. Figure 3.3 shows an example from the wireless security testing chapter with the recommendations for securing "Configuration, Authentication, and Encryption of Wireless Networks" and "Authentication".


9.5.3 Evaluate Configuration, Authentication, and Encryption of Wireless Networks

Verify that the access point's default Service Set Identifier (SSID) has been changed.

9.54 Authentication

Enumerate and test for inadequacies in authentication and authorization methods.

Figure 3.3: Chapter 9.5 Access Verification [14]

3.2.3 Penetration Testing Execution Standard (PTES)

The Penetration Testing Execution Standard is an open-source project that has been developed and maintained by a community of security professionals. A second version of PTES is under development [15].

PTES is a comprehensive guide that incorporates best practices for each of the seven steps involved. It offers an in-depth roadmap for executing each specific test included in the manual. Testers widely employ PTES to refine their scope and ensure that all areas, such as networks, applications, systems, and more, are thoroughly examined [15]. PTES includes the tools and methodologies required when doing a penetration test on a device. It also serves to enhance security awareness within organizations by outlining the expectations from a penetration test. It can function as a guideline or a checklist for a team of testers, which is its primary objective. The standard consists of these seven main phases [15]:

Pre-engagement Interactions: The preparation phase of a penetration test includes the baseline of tools needed to conduct a test. This phase also includes steps to make sure that all approvals and regulations are set for the specific test.

Intelligence Gathering: This phase aims to gather information about the system or application to be tested. The phase involves Open-source intelligence (OSINT) steps and also the use of scanning tools etcetera.

Threat Modeling: A phase to identify both the potential attackers and the vulnerable assets.

Vulnerability Analysis: This phase aims to identify vulnerabilities by using the provided methodologies and tools.

Exploitation: Discovering and evaluation of the identified security vulnerabilities.

Post Exploitation: This phase refers to the phase in which the tester has successfully gained access to a system or network and is attempting to maintain that access or escalate privileges to gain further access to the system. By identifying and exploiting vulnerabilities during post-exploitation, the tester can help the organization to improve its security measures against real-world threats.

Reporting: Defines a structure on how to document the penetration test.

PTES has been successfully used in other research to perform a penetration test. Astrida et al. [29] used PTES to target wireless network security systems.

3.2.4 OWASP for IoT

The OWASP IoT Project was developed by the Open Web Application Security Project (OWASP) foundation [16]. The project aims to help consumers, manufacturers, and developers to assess the security risks associated with IoT devices. It includes detailed information of common vulnerabilities and attack vectors that IoT systems may be vulnerable to. The framework, however, does not provide a methodology to perform the tests and it is limited in providing tools. The IoT project contains methodologies for firmware analysis, the tools necessary for which are developed by themselves. The OWASP IoT project contains valuable information about the IoT attack surface along with attacks and tests that can be used when conducting a penetration test. OWASP also provides external documentation to other organizations projects and methodologies to penetration test with automate security auditing of IoT devices [16].

The OWASP Testing Guide (OTG) [30] is another project by OWASP that provides guidelines and tools for a penetration test. OTG however, focuses on testing web applications and not on the devices themselves.

3.2.5 NIST 800-115

NIST Special Publication 800-115, issued by the National Institute of Standards and Technology (NIST), offers comprehensive guidance on conducting information security testing and assessments [17]. The publication is intended to offer guidance to organizations on how to plan and execute technical information security testing and assessments. It also provides guidance on how to analyze the results and produce strategies to mitigate identified risks. It contains methods towards identifying vulnerabilities in a network or system and verifying compliance with policies or other requirements [17].

NIST 800-115 covers separate phases such as:

- Security Testing and Examination Overview
- Review Techniques
- Target Identification and Analysis Techniques
- Target Vulnerability Validation Techniques
- Security Assessment Planning
- Security Assessment Execution
- Post-Testing Activities

There are multiple security testing techniques such as vulnerability and wireless scanning that are covered. The document focuses on why those testing techniques are needed, however, they do not include any guidelines on how they could be used. The standard also gives recommendations of different security testing tools for each testing technique. Furthermore, penetration testing is presented as a target vulnerability validation technique along with password cracking and social engineering. In this standard they present a four-stage penetration testing methodology [17] which will be shortly described below:

Planning: For this phase, the testers identifies rules and obtains final approval from management. This means that the testing goals are set, and everything is documented.

Discovery: This phase consists of two parts: First, the testers preforms information gathering by using different types of software scanning combined with OSINT. The second part is vulnerability analysis to identify potential security risks.

Attack: This phase is described as the "heart" of a penetration test. The attack phase can have many different outcomes as systems can be fully compromised or just give more information about the targeted system. The testers try to exploit a vulnerability, from there they may get access to sensitive information or find a way to escalate the system. Figure 3.4 shows the detailed attack phase presented in NIST 800-115.

Reporting: The reporting phase is always in progress during the other phases as all taken steps need to be documented. At the end of a penetration test, reporting is used to present the identified vulnerabilities along with a risk rating and its impact. Lastly, it should also give guidance and a discussion on how to mitigate the exploited vulnerabilities.

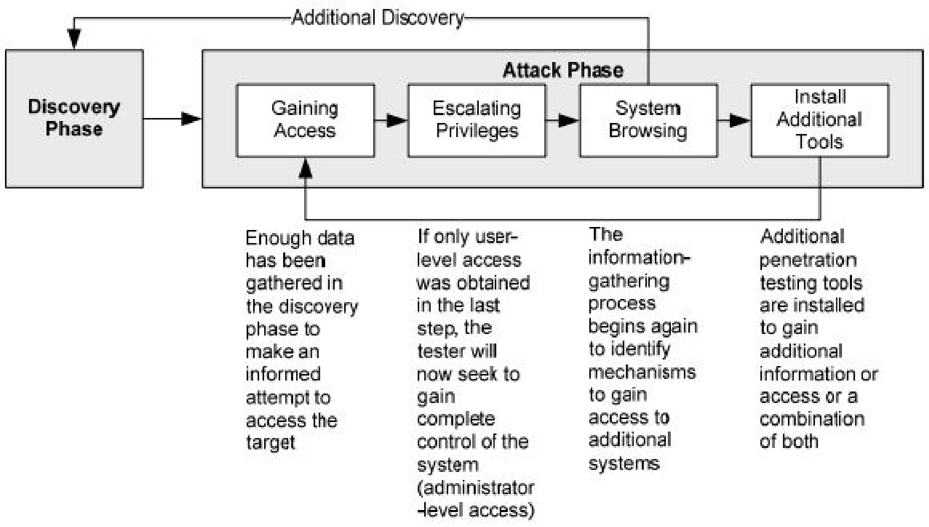

Additional Discovery
Discovery Phase
Attack Phase
Gaining Access
Escalating Privileges
System Browsing
Install Additional Tools
Enough data has been gathered in the discovery phase to make an informed attempt to access the target
If only user-level access was obtained in the last step, the tester will now seek to gain complete control of the system (administrator -level access)
The information-gathering process begins again to identify mechanisms to gain access to additional systems
Additional penetration testing tools are installed to gain additional information or access or a combination of both

Figure 3.4: Detailed Attack Phase [17]

4 Framework comparison

From the theoretical background, four standards/methodologies were presented and selected for potential use. The framework comparison is important to distinguish between penetration testing standards and security assessments as this research focuses on penetration testing frameworks. Next, a selection was made based on how well the standards applies to the following parameters:

The first parameter (**Framework**) describes how well the standards suit the definition of framework described in chapter 3.2.1. This parameter is important due to the possibility of using it as guidance in a penetration test. This includes how methodologies for performing specific tests are presented combined with if the standard recommends or presents tools to be used in the tests.

The second parameter (**Penetration testing specific**) is a crucial factor for this research since it's focused on standards that could be applied to penetration testing. This parameter is used to distinguish between if the standard is penetration specific or more of a security assessment.

The third parameter (**IoT Specific**) is used to verify if the standards are IoT specific. Meaning that it shall either have a subsection for IoT security and testing, or it should have methodologies and tools included that can be applied to IoT testing.

The fourth parameter (**Methodology**) describes how well the standards have included methodologies for specific tests. The framework should encapsulate methodologies, and methodologies should encapsulate tools (Application suite).

The last and fifth parameter (**Application Suite**) is an important parameter considered when evaluating the different standards. This parameter is used to identify if the standards include any tools, both hardware and software for penetration testing. Having a base application suite is a valuable source when conducting a test.

Candidate	Framework	Penetration Testing Specific	IoT Specific	Methodology	Application Suite
OSSTMM		Security Assessment			
PTES	✓	✓		✓	✓
OWASP IoT			✓	Limited	Limited to firmware
NIST 800-115	✓	✓		✓	✓

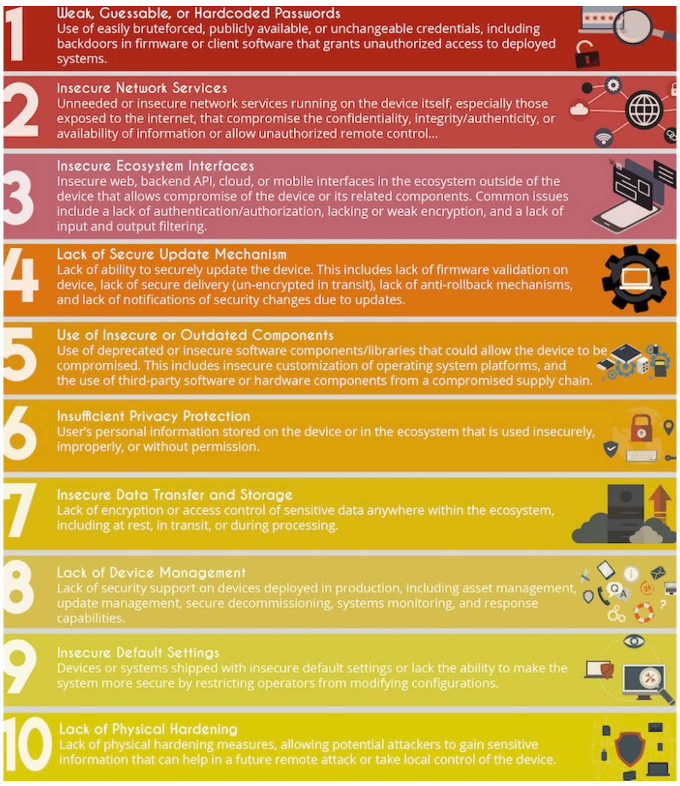
Table 4.1: Comparison Table

Based on the comparison and evaluation of standards, two of the originally selected standards conform to the definition of framework. The Open Source Security Testing Methodology Manual (OSSTMM) [14] fails to meet any of the parameters and its concluded to be more of a security assessment standard rather than penetration testing specific. OWASP for IoT [16] is the only candidate that targets IoT devices specifically with focus on defining the IoT attack surface. OWASP does include methodologies and an application suite but it's limited to firmware security testing, however this is the only candidate that involves tools and methodologies for firmware testing.

"The Penetration Testing Execution Standard" (PTES) and NIST Special Publication 800-115 meets the requirements to be defined as frameworks as they "*encapsulates methodology and methodology encapsulates tools, techniques, and resource*" [5]. PTES and NIST 800-115 do not target IoT testing specifically, and it does not contain a special subsection for those types of tests. However, both contain methodologies and tools to perform specific tests which makes them appropriate to use as a guidance when conducting a penetration test. The selection of candidates to be used in the next section was made based on which of them conformed to the definition of framework which resulted in The Penetration Testing Execution Standard (PTES) and NIST 800-115. The other candidates have been excluded for the next section as they did not contain enough methodologies or tools to conform to the definition of framework or to be used in guidance in a penetration test effectively. Nevertheless, OWASP for IoT and OSSTMM can be a valuable source of information for a penetration test. While OWASP specializes in identifying the most common IoT vulnerabilities and firmware testing, OSSTMM can help pinpoint specific areas that need to be evaluated to meet regulatory and legal requirements.

4.1 Addressing the common vulnerabilities

This section will investigate how and if PTES and NIST 800-115 could be applied to address and identify the top ten vulnerabilities and possible attacks presented in OWASP top 10 for IoT [16]. The frameworks may contain a detailed methodology or only recommended tools to perform a specific test and attack.


1 Weak, Guessable, or Hardcoded Passwords
Use of easily bruteforced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.

2 Insecure Network Services
Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control...

3 Insecure Ecosystem Interfaces
Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.

4 Lack of Secure Update Mechanism
Lack of ability to securely update the device. This includes lack of firmware validation on device, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates.

5 Use of Insecure or Outdated Components
Use of deprecated or insecure software components/libraries that could allow the device to be compromised. This includes insecure customization of operating system platforms, and the use of third-party software or hardware components from a compromised supply chain.

6 Insufficient Privacy Protection
User's personal information stored on the device or in the ecosystem that is used insecurely, improperly, or without permission.

7 Insecure Data Transfer and Storage
Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.

8 Lack of Device Management
Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.

9 Insecure Default Settings
Devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting operators from modifying configurations.

10 Lack of Physical Hardening
Lack of physical hardening measures, allowing potential attackers to gain sensitive information that can help in a future remote attack or take local control of the device.

Figure 4.5: OWASP Top 10 For IoT [16]

1. Weak, Guessable, or Hardcoded Passwords.

PTES: PTES has included methodologies and tools to both brute-force passwords and to extract passwords. The extraction part only refers to obtaining password hashes in windows. PTES has included tools for brute-forcing (4.2.5) passwords such as **Brutus, Hydra, Medusa and Ncrack.** Password cracking techniques as using rainbow tables are also included. PTES refers to lists of common passwords (3.2.1.1) that can be used when testing. Furthermore, PTES covers tools with the ability to potentially sniff password hashes and how to test the possibility to crack Wi-Fi passwords. Overall, PTES covers a broad list of methodologies and tools to test weak, guessable, or hardcoded passwords.

NIST 800-115: NIST also includes a lot of different methods about how to test weak, guessable or hardcoded passwords. Target Vulnerability Validation Techniques has a subsection for password cracking (5.1) that contains testing techniques such as using a dictionary attack, rainbow tables and the brute-forcing technique. Password cracking tools such as **Hydra, John the Ripper, RainbowCrack and Rcrack** is covered. NIST also covers techniques such as network sniffing and social engineering to catch password hashes or to get access without any authentication token.

As NIST focuses more on helping organizations to secure their environment, the methodologies to perform the tests is a bit more limited than PTES, but a set of tools and what areas to test are included.

2. Insecure Network Services

PTES: There are multiple ways to test unneeded or insecure network services running on the device. PTES covers this type of test by including tools and methodologies for intelligence gathering, vulnerability analysis, and exploitation. For the intelligence gathering phase, tools such as **Nmap** and **Metasploit** are mentioned. PTES recommends performing vulnerability analysis on the identified services to identify potential vulnerabilities by either using automated tools as **Nessus** and **OpenVAS** or manual testing techniques such as **code analysis** and **fuzzing**.

For exploitation, PTES covers many different attack vectors. This can include exploiting unpatched software vulnerabilities or misconfigurations in the network services.

NIST 880-115: Compared to PTES, NIST is not as detailed in its coverage of this specific vulnerability. The framework does include network scanning tools such as **Nmap**, and the vulnerability scanner **Nessus** even though it is mentioned as a "penetration testing" tool only and not bound to the specific area. For the actual exploitation, NIST does not cover any potential exploitation techniques or tools than those focused on password cracking.

3. Insecure Ecosystem Interfaces

PTES: PTES does cover web application scanners (3.1.3) such as **IBM AppScan** to identify vulnerabilities over insecure web interfaces. Additionally, PTES does include metrics for several types of injection type attacks such as XSS, SQL injections in terms of lack of input and output filtering, however it's covering is minimal. It does include network sniffing tools as **Wirekshark** and **Tcpdump** that could be used to capture traffic from and to insecure ecosystem interfaces.

As for the insecure backend API and cloud testing, PTES fails to provide valuable information for conducting tests on those types of integrations. APIs and a cloud-based connection model are commonly integrated into IoT devices. A popular tool like **Burp-Suite** that is used for security testing of web applications is not present. Furthermore, there are no baselines for assessing the security of APIs. Data breaches are often caused by broken, exposed, or hacked APIs, which can expose sensitive medical, financial, and personal data to unauthorized access. However, it is important to note that not all data carries the same level of sensitivity. As it's one of the most frequently used communication channels for IoT devices, it is essential to test and secure it. [31].

To summarize, PTES does not cover testing methodologies and tools to be used as a baseline for testing insecure ecosystem interfaces. It does provide guidance towards testing commonly used services and protocols but fails to cover the integration of cloud and APIs that is being frequently used in the IoT domain.

NIST 800-115: In the same way as PTES, NIST 800-115 does not cover any methodologies or tools to be used in detail when testing insecure ecosystem interfaces such as web services, backend APIs and cloud interfaces. It does include active and passive network scanning by using tools like **Wireshark**, nonetheless, it is focused on capturing traffic over Wi-Fi (IEEE 802.11).

NIST 800-115 does not specifically address testing insecure web, backend API, cloud, or mobile interfaces, it can be used as a reference for evaluating the security of these integration's more theoretically to use it as a baseline for conducting a penetration test.

4. Lack of Secure Update Mechanism

PTES: PTES fails to cover methodologies and tools to validate the firmware of a device. PTES is missing methods to do firmware analysis, which is the process of examining the firmware code and its behavior on a device to identify any security vulnerabilities, functionality issues, or other potential risks. Firmware analysis involves analyzing the firmware's structure, functionality, and interactions with the device's hardware and software components, as well as identifying potential attack surfaces and security weaknesses.

Furthermore, a common vulnerability for IoT devices is not having a proper anti-rollback mechanism which means that a failure can be a failed update or compromised firmware. PTES does not cover any test on Firmware modification, Installing unauthorized firmware or reverse engineering the firmware. All those firmware testing procedures mentioned above are commonly occasionally attacks [32].

NIST 800-115: In the same way as PTES, NIST 800-115 does not cover any methodologies and tools to test the lack of secure update mechanisms which mainly focuses on validating and testing the firmware.

5. Use of Insecure or Outdated Components

PTES: To test the use of insecure or outdated components, it would involve techniques such as using automated vulnerability scanning tools or manual inspections as source code analysis to identify possible exploits. PTES does cover automated vulnerability scanners as **OpenVas** and **Nessus** (3.1.2). It does also include passive vulnerability scanning tools like **Metasploit** (3.1.4). There is no coverage regarding code analysis of firmware, software, or applications that can be used to find bugs, exposure of sensitive data or any other security misconfiguration. Other ways to find vulnerable software are to perform direct tests rather than using known flaws found by a vulnerability scan. PTES does cover exploitation techniques such as buffer overflows, DoS, SQL Injection, and XSS tests. However, some techniques are not that detailed such as DoS, that only includes a variant of DoS that is the "802.11 RTS/CTS flood attack" and neither does it contain any tool to assess this.

NIST 800-115: NIST 800-115 does cover the usage of vulnerability scanner along with a methodology (4.3). NIST 800-115 includes other useful resources that could be used to identify known vulnerabilities. The additional source of information is Common Vulnerabilities and Exposures (CVE) databases that consists of known security issues that have been publicly announced. For example, the National Vulnerability Database is included.

As mentioned earlier, NIST 800-115 does not cover any methodologies or tools for the exploitation phase. It does however contain theoretical information about the potential exploitation's about for example buffer-overflows, Kernel Flaws and briefly about injection exploitations.

6. Insufficient Privacy Protection

PTES: A penetration tester may assess the security of a user's personal data stored on a device or within a network by attempting to bypass protections using tactics such as exploiting weak passwords, outdated software, or poorly configured access controls. This process also involves scrutinizing file systems, databases, and network traffic to pinpoint any potential vulnerabilities in the data management procedures.

PTES does cover some ways to uncover personal information by for example, capturing network traffic (WiFi, Bluetooth), brute-forcing a password, or retrieving information through SQL injections. PTES does not include any basic methodologies or tools to explore a filesystem, this could include techniques as dumping the filesystem and mounting it to another device to allow for offline exploring.

Overall, PTES does not cover testing with a focus on the privacy protection of a system. And neither discusses the importance of privacy protection or how personal data should be handled.

NIST 800-115: As NIST 800-115 also defines it as a security assessment it theoretically describes that its important of organization to address the privacy concerns of the personal information stored. It does cover file integrity checking tools as **Autopsy** and **Sleuthkit** but does not contain any methodology or description of what the software tools can be used for.

Overall, the penetration testing section does not include the importance of testing insufficient privacy protection.

7. Insecure Data Transfer and Storage

PTES: To test insecure data transfer in transit, PTES does cover the usage network sniffers that mainly focuses on WiFi traffic. This means that it leaves out IoT specific protocols like Zigbee and LoraWAN. Neither does PTES include any techniques nor tools targeting data recovery, reverse engineering of data extracted or data manipulation. PTES includes methodologies and tools for testing data storage by SQL injection but without any database vulnerability scanning tools. Testing insecure data transfer and storage in terms of IoT may target an SD card or any cloud synchronizing process.

Overall, PTES does cover some methodologies and tools to test the specific area, but it is not targeting the technologies integrated in IoT devices.

NIST 800-115: NIST 800-115 does not cover testing insecure data transfer or storage except using different types of wireless scanning. The security assessment part of the framework presents the importance of ensuring the security level of the data to be transmitted or stored within the system. For a penetration test, NIST 800-115 does not cover any methodologies and tools to evaluate insecure data transfer and storage. The focus is to address the problem with a proper security assessment.

8. Lack of Device Management

Except for the problem with update management covered in **Lack of Secure Update Mechanism** section, this security challenge may not be appropriate for a penetration test. As it may not be vulnerable per say, the problem by having improper update management, asset management, secure decommissioning, systems monitoring, and response capabilities will let the attacker explore the system without being noticed. The attack surface of the device grows without having device management implemented, physical attacks get easier as-well as establishing persistence.

9. Insecure Default Settings

This security challenge would include evaluating all types of security vulnerabilities that a newly shipped device with insecure default settings could have, such as bad password handling, not enabling two factor authentication by default, outdated components and more. The methodologies and tools to mitigate this challenge are covered in the other sections.

10. Lack of Physical Hardening

PTES: Physical security measures as locks and access control systems are implemented to protect from unauthorized access. Without any security measures, an attacker can potentially gain access to sensitive information or even take control of the device by stealing or tampering with the device. As mentioned, the Physical Security of IoT devices is a challenge as they are likely to be deployed in areas where the physical access could be hard to control [19]. Another challenge is the IoT architecture which makes it hard to implement any device management or access control.

PTES does not include any methodologies or tools for specific tests when having physical access. Physical security is not mentioned in the frameworks attack surface.

NIST 800-115: Physical security testing is included as a technique in the section about Target Vulnerability Validation Techniques. The framework states that testing physical security is important. However, NIST 800-115 does not include any physical testing methodologies or tools.

5 The generic penetration testing framework for IoT

This section of the paper presents the framework structure with the included testing methodologies and tools.

5.1 Introduction

The proposed framework will include both hardware and software aspects which will be used to identify the domain specific challenges for IoT penetration testing. It will cover both the IoT specific testing methodologies that are present in the existing frameworks and the methodologies and tools that are missing. We create the generic framework by doing research on existing testing methodologies and tools based on the framework comparison findings and the theoretical background research.

As it is for most penetration testing framework, it will focus on a security testing perspective. However, it will also cover some data privacy concerns that can occur if security is lacking on a tested device. The goal of using this framework is to find security or privacy vulnerabilities in an IoT device implementation or usage of said device.

The framework will not cover penetration testing of central servers that may interact or collect data from an IoT device. This is mainly since they are not owned by the IoT device owner.

5.1.1 Scope

The scope in this case refers to the defined boundaries and goals of the IoT penetration test. It can for example mean that you only have access to the software and not the physical hardware part of the IoT device. In such a case, then all hardware attacks should be skipped. Defining the scope of a penetration test is critical for ensuring that the testing is focused and does not cause unintended impacts on other connected devices.

Without a defined scope, the penetration test may inadvertently impact systems or applications that are not intended to be part of the security assessment. It is within this part of the framework were the user conducting the test must know if the penetration test should be done silently. If so, then certain subsections such as active reconnaissance will not be available as they can alert firewalls and or intrusion detection systems. Ideally if the user only wants to test the security of the device itself rather than the implementation and usage, then the device needs to be placed on a separate network just for the penetration test.

5.2 Reconnaissance & Intelligence gathering

Reconnaissance refers to the process of gathering information about a target system or network prior to launching an exploitation [33]. The goal of reconnaissance is to gain a better understanding of the target environment, including its IP addresses, firmware/operating system, software versions, and other key details that can be used to identify vulnerabilities and potential attack vectors. When something of interest is found then it should be noted down. Each collection of useful data is referred to as an information point which will be used in the exploitation part of the framework. These information points may differ depending on which IoT device is tested. While there are different standards for IoT, software and architecture can greatly be different from device to device. In this case, the framework must assume that the user either knows which network the IoT device is connected to or has access to the physical hardware of said device.

5.2.1 Software

Due to the broad definition of what an IoT device can be there are many software areas that need to be explored. IoT devices may have their own end user application that is connected directly to the device. It could be standardized protocols on a network that sends and receives traffic. However, an IoT device can also host a web server to serve a web-application. Or it can send data directly to a centralized server which collects and serves the data in some sort of way to the end user.

Much like stated before, if the device sends data to a non-controlled centralized server, then penetration test should no be done on said server.

OSINT: Open-Source Intelligence is a type of intelligence gathering technique that involves collecting and analyzing information from publicly available sources. OSINT can be used to gather information about the hardware revisions, network communications and software versions. But more importantly it can be used to look up known vulnerabilities in public databases such as **Metasploit**, **CVE databases** and or different online forums.

Passive end user software reconnaissance: If the IoT device relies on an external app (PC, Android, or iOS), then observing available functions helps build a basic understanding of the app's workings. There are a handful of questions that can be asked to aid in the application profile building.

How does the app connect to the device? Is there any form of authentication when connecting? What can you control with the application? Can you send data to the IoT device using the application? Is the app generic, I.e., do multiple OEMs use the same application? If the application is running on an Android device you can open the app in some sort of application analyzer, such as **Apk info**.

With that you can see the application package name which may reveal more information that you can search for online. Such application analysis often features a way of showing all needed device permissions which can also give a good insight into how the application works.

Passive networking monitoring: If the scope includes network access that is done over Wi-Fi, then by simply observing the network traffic using tools such as **Wireshark**, then you can see some device information. For example, ARP packets that can reveal the device IP address and MAC address. If the device does not use randomized MAC addresses, then that can be looked up to see what vendor and or manufacturer the device belongs to.

Active software reconnaissance foreword: Note that active reconnaissance on the IoT device should not be done without alerting IT personnel, especially if there are monitoring or intrusion detection systems on the same network. Disregard this if you are the sole owner of the device, silent penetration testing is disregarded, or have been given confirmation of approval to penetration test said device.

Network scanning: This involves scanning the network for IoT devices and mapping out the network topology. More specifically by sending packets to various ports on a target host to determine which ports are open and which services are running on those ports. Different types of port scans, such as TCP, UDP, or SYN scans, may be used depending on the target environment and the scanning tool being used. With the collection of the open ports a rough estimation can be created on what services are running on the IoT device. Tools such as **Nmap** or **ZMap** can be used to discover IoT devices and identify open ports and services [33]. Note that this method creates network traffic that could be logged by the IoT device or an intrusion detection system.

Banner grabbing: This involves sending requests to IoT devices to retrieve information from their banners or headers. This can provide information about the device model, firmware version, and other details that can be used to identify potential vulnerabilities. Tools such as **Nmap** do this automatically unless specified. Much like the network scan, this creates noticeable traffic on the receiving end that can be logged.

End user application reverse engineering: If an IOS IPA or Android APK application is supplied with the IoT application, then that can most likely be decompiled. Doing so can give insight into how communication is done between the user app and the IoT device. It can also reveal how authentication is done between the user and the IoT device. With this intelligence you can see if the app implementation contains any security risk or vulnerabilities.

IOS An IOS device uses ".ipa" files for its applications. Such apps can be programmed in Objective-C and Swift. The first plan of action is acquiring the app's binary file (.ipa). The file can be downloaded using an existing IOS device or by downloading the app from the App Store using a tool like Apple Configurator 2. As .ipa file works as an archive it can be unarchived to reveal its contents, including the app's executable file and resources (e.g., images, storyboards, and localization files). The app files are compiled using the

LLVM compiler, and their binaries are in the Mach-O format. To analyze the app's code, a decompile or disassemble the binary needs to be done. Tools like **IDA Pro** or **Ghidra** can achieve this. The next step is to analyze the app's classes, methods, and functions. The focus would be interesting or suspicious code, such as calls to private APIs, hardcoded credentials, or any code that could be seen as a potential security vulnerability. The code will be in assembly which makes larger apps more difficult to understand. As such start by checking the strings found in the file and cross reference that to the code structure itself. Certain files such as ".plist" or "storyboard" files can also contain sensitive information [34].

Android An Android device uses ".apk" files for its applications and uses Java and or Kothlin. While other languages such as C#, C++ can be found, they are less common. You can also find JavaScript combined with HTML and CSS when an application is using the Adobe PhoneGap framework.

Unlike the IOS counterpart, Android decompilation is much easier due to the open nature of the Android ecosystem and its straightforward way to enable developer mode on retail devices.

Much like the IOS section there is a need to get a hold of the base application file i.e., APK file. This can either be downloaded from an APK mirror website or extracted from an android device using ADB or a file explorer app when already downloaded from the app store.

By using tools such as **APKTool** or **jadx**, the APK can be decoded, converting the binary DEX files into readable Java source code. These tools will also give the option to extract resources (e.g., XML files) from the APK. Once the APK is decoded, the app's package structure, classes, methods, and variables can be explored. As with IOS apps, look for interesting or suspicious code, such as calls to hidden APIs, hardcoded credentials, or code that can be seen as potential security vulnerabilities. If there is a need, a dynamic analysis on the app can be done. This is achieved by running the app in a controlled environment such as an Android emulator or a rooted device. This can help uncover hidden features or security issues that may not be apparent from static analysis alone. If possible, attach a debugger such as Android Studio's built-in debugger to set breakpoints and step though code [35]. A good all in one tool to achieve static and dynamic analysis is the **Mobile-Security-Framework-MobSF** which can often convert the APK to java files.

Vulnerability analysis: Vulnerability analysis is the process of identifying vulnerabilities in systems and applications. It's divided into Identification and validation where identification is the vulnerability discovery effort and validation is the process where the tester tries the identified vulnerabilities. This can be done by using automated tools such as **openVAS** or **Nessus**. another approach is search for the model of the device of firmware version etcetera. at **CVE databases**.

5.2.2 Hardware

As hardware architecture may differ greatly between IoT devices it creates a unique security challenge compared to other computer devices. An IoT device can use a simple 32-bit microcontroller such as the ESP8266 used to send and receive data, or they can use more complex 32 bit or 64-bit RISC hardware. Something that is often universally standard within IoT devices is the use of a RISC computer architecture. This is due to the power draw limitations of the small devices [36]. Depending on the application, the device may have different sensors which send the data over different physical interfaces. The networking interface varies depending on the device.

If hardware is not within the scope of the assignment, then it removes certain hardware reconnaissance techniques. Those techniques will be noted as such.

Tamper mechanisms: IoT device itself may have physical tamper mechanisms, which on trigger may send an alert to the vendor or device administrator, locking out the device or rendering the device bricked. This tamper mechanism may come in the form of pogo pins attached to the ground, pressed buttons, switches, or other forms of pull-down resistors. These may trigger when the device gets opened.

The user must execute with caution whenever hardware is disassembled for reconnaissance. Another tamper proofing mechanism is the use of software or firmware signing. A checksum or signature for system files can be programmed into a lower-level part of the device. If a user attempts to modify system files of the device, a flag can be raised which may alert of affect device functionality [37].

Finding device version and hardware revision: Finding out which version and or hardware revision of the device can aid the penetration testing process.

If the device is sold internationally, you can often use the FCC ID for it which can contain useful information about the device. Information such as wireless frequencies used, manual and high-resolution internal photos of the device.
Websites such as `https://fccid.io/` which lookup the device in the FCC database can help with this.

If the device is only sold to European markets, then you can try searching for its serial number on any search engine or specific websites such as `https://device.report`.

Finding debug headers: IoT devices often contain debug headers such as UART and JTAG to debug and program the devices from factory. Using a multimeter set-in continuity mode, and then prodding at pads, ports, or unused pins to see where they are traced to the chip can help with this. Often the pads have a silkscreen labeling on what they do. UART will often be noted as RX, TX and sometimes also include pads for RTS and CTS for handshaking mechanism. If the device uses UART, then using tools such as a logic analyser with a RS232 decoder can help to find the correct baud rate for the device. Alternatively, you can often find the baud rate needed for communicating with the device in the SoC documentation. If the steps above do not work, hook up TX to an oscilloscope and set it to trigger a pulse. By measuring the time of the shortest pulse and inputting the microseconds into the following formula $\frac{1}{X} \cdot 10^6$ yields a baud rate that needs to be rounded to the nearest common baud rate [38].

5.3 Exploitation

This part of the framework will denote the most common ways to exploit an IoT device. It will cover the attack vectors in chronological order from least access (network access) to most access (hardware level).

5.3.1 Network

IoT networking uses a plethora of different standards. As such, to limit this framework in scope the six most common networking interfaces is covered:

Protocol	**Frequency (Europe)**
Wi-Fi	2.4 GHz, 5 GHz
Ethernet	Wired
Cellular (LTE)	Depends on country and network operator. Band 3 at 1800 MHz is commonly used.
Zigbee	2.4 GHz and 800 MHz with Zigbee PRO 2023
Z-Wave	868.4 MHz, 869.85 MHz
LoRaWAN	868 MHz

Table 5.2: Protocol frequencies in Europe

Do note that these frequencies will vary depending on which region the device is sold and developed in.

Password cracking: IoT devices are known for having limited and insecure password management due to their lack of hardware resources. As such they are prime targets for password testing. This involves bruteforcing and looking for hard-coded password or encryption keys. There are a lot of software tools to be used for password brute forcing or to be used with a word list of known passwords.

The first step would be to check the default password of a device before moving on to the software tools as **RainbowCrack** or **John the Ripper**. Some of the tools require that the tester has obtained the password hashes. However, there are also software tools that can target the authentication protocol and connection (SSH, Telnet etcetera.) directly as **Hydra** or **Metasploit**.

Web applications: Web applications and interfaces hosted by the IoT device will often contain a graphical user interface with a underlying server and database. The application could run as Software-as-a-service (SaaS) or by running connected with the firmware. Lack of authentication/authorization and input/output filtering may result in information leakage. XSS, SQL-injections and MITM are attacks that could be tested manually. An example is CVE-2017-17020 [39], where an attacker could exploit the device web server with a user interface to inject code and retrieve information directly from the filesystem.

Automated tools as **Wega**, **PwnXSS** can automatically be used to scan web applications for vulnerabilities. Furthermore, **Burpsuite** is a commonly used web application testing tools as it can be used to both scan, password crack and setup a man-in-the-middle proxy to test a web application.

API-testing: An API lets a service communicate with other devices and services. As IoT APIs are the entrance for IoT apps and how the devices communicate with IoT cloud platform resources, its important to ensure that they are secure. Li, Yilian, et al. managed find 21 APIs with vulnerabilities in their work by doing automated scanning [40]. APIs can be evaluated by sending requests to the API outside the API which tools as **SoupUI**, **Postman** and **Pulse** can be used for. However, modern IoT integrated APIs may use encryption and signatures to verify API request which could make the test harder to conduct with automated tools. There are ways to bypass security measures as using exposed dynamic replacement technology to send test request or providing the tester with more information [40].

Zigbee design issues: There is little documented information regarding design flaws in the Zigbee standard. Most attacks are theoretical and do not contain any proof-of-concept code or implementation. As such they have been left out of the framework due to their lack of repeatability. There are tools in which you can capture and replay packets such as **KillerBee** or **Attify's** GUI wrapper of the same program. However replaying packets is not possible on all devices and is very device version specific.

Z-Wave design issues: Normally a list of vulnerabilities does not need be included in the framework as that is already covered by the reconnaissance stage of the framework. However, in this case the following vulnerabilities are deemed noteworthy to have their own category. Z-Wave has a multitude of documented vulnerabilities that can lead to various outcomes. The chip for Z-wave nodes support AES-128 encryption along with the Diffie–Hellman key exchange which is used automatically for S2 authentication. However, there is a authentication downgrade attack using the so called Z-Shave attack which lowers the security down to S0. S0 uses a hardcoded encryption key which makes MITM of unencrypted data possible [41].

If the IoT device uses Z-Wave for communication, then it may be vulnerable to a multitude of different attacks. This applies if the device uses a chipset that is from Silicon Labs series 100 to 500 or series 700 that use S2 for authentication [42]. The following hardware vulnerabilities are seen below:

- Z-Wave devices based on Silicon Labs 100, 200, and 300 series chipsets do not support encryption.
- Z-Wave devices based on Silicon Labs 500 series chipsets using CRC-16 encapsulation do not implement encryption or replay protection.
- Z-Wave devices based on Silicon Labs 500 series chipsets using S0 authentication are susceptible to uncontrolled resource consumption which can lead to battery exhaustion.
- Z-Wave devices based on Silicon Labs 500 series chipsets using S2 are susceptible to denial of service and resource exhaustion via malformed SECURITY NONCE GET, SECURITY NONCE GET 2, NO OPERATION, or NIF REQUEST messages.
- Z-Wave devices based on Silicon Labs 500 and 700 series chipsets are susceptible to denial of service via malformed routing messages.
- Z-Wave devices based on Silicon Labs 700 series chipsets using S2 do not adequately authenticate or encrypt FIND_NODE_IN_RANGE frames.[42]

These vulnerabilities are very device dependent and as such require that information be collected in the reconnaissance stage.

Software Defined Radio (SDR) based attacks: Most IoT networks operate at 2.4Ghz or lower. This means that the communication of the protocol can be read out by any receiver that can achieve that frequency. If there are no or simple replay protections like incremental frame countering, then you can simply replay the captured messages after some modification.

This can ultimately create a denial-of-service attack if the SDR sends a more powerful signal at the same frequency. Alternatively, if the IoT device is using cellular traffic for sending and receiving data you can emulate a cellular base station using a SDR. If the device switches over to the rouge base station, then all data can be captured. This can be achieved with a BladeRF SDR and accompanying software like Nuand's YateBTS.

Replay attacks: If the device communications model is known it may be vulnerable to replay attacks. By intercepting the communication to and from the device, the intermediary could replay those packets if there are no countermeasures. Those captured packets can contain actions or other sensitive information which could be replayed to the attacker's advantage. Communication can be intercepted and replayed by using a SDR or software tools as **Aircrack-ng**, **Fiddler** and **mitmproxy** combined with python scripts.

Denial of service (DoS): Denial of Service (DoS) attacks are cyber-attacks in which an attacker seeks to disrupt the functionality of a device, service or network in some sort of way by overwhelming it with a flood of traffic, requests, or other unwanted activity. This is a common attack within IT security as a whole and as such also applies to IoT devices. DoS attacks can be done in many different ways depending on what sort of networking protocol is being used.

A simple way to achieve a DoS attack is by jamming the radio frequencies used by the target system using a SDR. This is especially needed if the device does not use Wi-Fi. With an SDR, an attacker can transmit a high-powered signal on the same frequency as the target system, effectively jamming the system's radio signals as it drowns out the legitimate data.

WiFi deauthentication: If the device is using WiFi on WPA2-PSK and lower for its data communication, then it may be vulnerable to a denial of service attack using WiFi deauthentication [43]. IoT devices that are based on the ESP8266 or ESP32 microcontroller only support WPA2 Personal/Enterprise. By sending deauthentication frames towards the device it will force the device to lose the connection. Thereafter, the device will try to reconnect to the same network. Here an attacker could capture the handshake to obtain a password hash. Another option would be to set up a rouge access point with the same SSID, if the device then connects to the rouge access point an attacker could capture all network traffic to and from the device.

With a network card in monitor mode, this test could be done by using software tools such as **Aircrack-ng**, **Airgeddon** for deauthentication or setting up a rouge access point and **Wireshark** for capturing network traffic.

ARP poisoning & ICMP based Man-in-the-middle attacks: If the device uses Wi-Fi or Ethernet as its main communication protocol, then it may be susceptible to ARP or ICMP based MITM. This will only work on non-encrypted data traffic and most encryption downgrade attacks such as SSL stripping is useless today if the networking implementation is done correctly. ARP poisoning is a very noisy attack and can be seen by anyone listening for ARP packets on the network. ICMP based MITM will not work on a network that uses static routes or not accept/process ICMP redirect packets. Tools as **Wireshark** and **Ettercap** could be used to create such an attack and gain useful information about the target.

5.3.2 Firmware

This section covers IoT device firmware analysis for the sake of finding vulnerabilities in the implementation. If hardware is outside the scope of then this part of the framework can be skipped.

Firmware readout/dumping: The process of reading firmware from a device can be a valuable technique in identifying vulnerabilities. This method provides a more comprehensive understanding of the device's operations, revealing implementation shortcomings that may be exploited. This approach intersects with the reconnaissance stage of the testing process, and one can begin by attempting to locate the firmware online. Producers or OEMs may provide firmware on their websites, especially when the device becomes unusable due to issues such as a bricked state. Alternatively, one can intercept traffic during an OTA firmware upgrade to determine the URL where the firmware is stored. In cases where none of the aforementioned options are available or firmware upgrades could eliminate potential vulnerability leads, physical firmware dumping becomes the best option. The type of IoT device being penetration test determines whether firmware dumping is possible. Since these devices require unique Wi-Fi SSIDs and passwords for each user, they cannot use read-only memory. Therefore, there is usually a UART header left on the devices from the factory that was used for initial programming. By consulting the microcontroller schematics and documentation, the pinout for reading firmware can be identified. However, it is essential to note that the device will often require either 3.3v or 5v externally if the chip is separated from the power circuit. If the device is based on an ESP32, the firmware can be easily dumped using **ESPtool** by specifying the flash memory size [44].

Firmware decompilation: If the firmware has been obtained, it can be analyzed using firmware analysis tools such as **Binwalk**. The filesystem configuration files and binaries can be carved from the target firmware to be assessed for security misconfigurations that could lead to exploitation. The file system can be examined for sensitive information that should not be public. These types of misconfigurations often lead to the possibility of gaining root access and executing unauthorized code.

Alternatively, you can convert the firmware binary code into human-readable assembly code. This is done by analyzing the machine code instructions and translating them into a more understandable format. This step can be performed using disassemblers like **IDA Pro** or **Ghidra**. With that the user can examine the assembly code to identify the various functions, data structures, and algorithms used in the firmware. This may involve tracing the execution flow of the program, identifying function calls, and understanding how the firmware interacts with the hardware. If possible, you can convert the assembly code into a high-level programming language, such as C or C++, to facilitate easier analysis and understanding. The mentioned decompilers can perform this task.

Do note that firmware decompilation and reverse engineering may be subject to legal restrictions, depending on the jurisdiction and the specific firmware being analyzed.

Downgrade attacks: A downgrade attack is the act of downgrading a device firmware to an older version which has known exploits or security vulnerabilities. This can be achieved by intercepting the data being sent from an over the air update or by flashing it via external tools or storage mediums. However, this attack is very device specific and often requires physical access or application access. This attack vector is often patched by default by using eFuses, update servers using TLS and or signed firmware.

Malicious firmware upload: This attack works much the same as a downgrade attack, however the attack instead uses custom firmware that can contain backdoor access or other modifications that diminish the security of the device.

5.3.3 Hardware

This section of the framework requires physical access to the IoT device. If hardware is not in the scope of the penetration test, then this part can be skipped.

Reading out data from external storage: There are some IoT devices that use external storage to store data such as sensor information or operating systems. This may be done as an alternative to storing data in a cloud service or adding a history to collected data. These interfaces can be in the form of SD memory cards, SIM cards, and USB storage devices. If there are no physical barriers, then a user can unplug the external storage and move it to another system where all data can be read out. If it contains system files then that can give crucial intel on system information and could show any hard coded security keys, networking address and certificates unless they are encrypted in any sort of way. Alternatively, while very device specific if the external storage only stores sensor data onto external storage, then it could be susceptible to buffer overflow attacks. This is dependent on whether the system uses a common file system and if the device is poorly programmed to handle incorrect file sizes. The program could crash if it uses a fixed array size when reading in data from an external device. By modifying existing files to be larger than expected it could write data outside the given variable [45].

Hardware tampering: While not a vulnerability per says, it needs to be stated within framework. If the IoT device is found in a physically open setting, then it needs to be physically locked down and secured to a good surface so no malicious user can tamper or remove it from its premises.

Side-channel attacks: Side channel attacks are techniques used by attackers to exploit the unintentional leakage of information from a system, typically by analyzing physical aspects of the device.

This is especially prevalent on IoT devices due to their constrained resources and widespread use. These attacks include but are not limited to

- Timing
- Temperature-based
- Electromagnetic
- Power analysis
- Acoustic

All these outside factors can be impacted on different operations done on the device; with enough data you can start to carve out what operations the device is doing.

Voltage glitching: Voltage glitching is a technique used to extract sensitive information from a device by manipulating its power supply voltage. The technique involves introducing a momentary voltage drop in the power supply or sending a larger than expected spike of voltage to a microcontroller or other integrated circuit during the execution of code. This power fluctuation can cause the IoT device to behave unexpectedly such as skipping cryptographic operations, dumping data, injecting code, or allowing the device to enter some sort of debug mode [46, 47] . The attack is very "low level" and requires high technical knowledge of how the device functions. This type of attack is intricate as it requires specific voltage level, timing, and time span for if the voltage spike leads to a skipped instruction.

The attack is also device specific as it depends on CPU type, power management, available pins, and the code itself. The simpler the circuit and SOC is, the "easier" it becomes to manipulate. As such, this attack should be used for worst case scenarios when none of the other attacks are viable. A voltage glitch can be achieved with the use of many different devices that can send specific timed pulses. A common device for achieving this glitch is the "ChipWhisperer" series of tools.

5.4 Post-Exploitation

While other frameworks do cover some form of persistence. This generic framework will not cover how to add persistence onto an exploited system. This is done due to the ethical considerations of the report.

5.5 Reporting

Finally, if you find any exploits or vulnerabilities in the IoT device you'll need to disclose them. This is usually done by writing a report. If you are the sole developer of the device, then this may not be necessary.

Writing a report on a security exploit involves documenting the details of the vulnerability, the steps taken to reproduce it, and any potential impact it may have on the system or organization using said device. A report should be easy to understand and should highlight if any risks were found. Additionally, it's important to follow any organizational or industry-specific reporting guidelines or procedures to ensure the report is submitted in a standardized and timely manner.

A report should contain the following points.

Brief introduction: Begin by introducing the exploit and providing context on why it's important to document it. Explain which part of the device was exploited, the device or the software that is used and the potential impact it may have on the device user.

Vulnerability description: Provide a detailed description of the vulnerability, including how it was discovered, the cause of the vulnerability, and how it can be exploited. Include any relevant technical details, such as code snippets or screenshots, to help illustrate the vulnerability. Do note that you'll need to proceed with caution with documenting. For example, if you find a way to dump the firmware on the device. then it might infringe on the copyright as it can be classified as proprietary software. You'll need to check with your local laws before publishing a report that may contain such sensitive information.

Discussion about impact: Describe the potential impact the exploit could have on the system or organization. This may include data loss, system downtime, or unauthorized access to sensitive information.

Outline the steps to reproduce: Provide a step-by-step guide on how to reproduce the vulnerability. This will help others verify the vulnerability and test for potential mitigations.

Potential mitigations discussion: Describe any potential mitigations that can be implemented to reduce the risk of the vulnerability being exploited. This may include software patches, firmware upgrades, hardware upgrades, configuration changes, or other security measures.

5.5.1 Vulnerability disclosure to vendors

Sometimes it is necessary to contact the vendor or device maker to create a safe disclosure of a found exploit. This is however only needed if the exploit uses an unknown vulnerability. Some vendors do have specific bug bounty programs in which the vulnerability can be safely disclosed. If that is not found, then you'll need to identify the right contact such as their security team or a designated email address for vulnerability reporting.

5.6 Application suite

An example of software penetration testing tools that could be used to target IoT devices and to be used with the framework methodologies.

Penetration Testing Targets	**Penetration Testing Tools**
Reconnaissance & Intelligence gathering	
Reconnaissance	Nmap/Zenmap, TCPdump, Wireshark, Shodan, Maltego
Vulnerability Scanning	openVAS, Nessus, Burpsuite, CVE Databases, Metasploit, Sqlmap
Exploitation	
Password testing	Rainbowcrack, Hydra, John the ripper, Metasploit, Hashcat, Cupp
Network	Aircrack-ng, Airgeddon, Wireshark, Fiddler, Mitmproxy, PCAPdroid, Ettercap, Aireplay-ng
Firmware	Binwalk, IDA Pro, Ghidra, Firmware test suite, Autopsy, Binary ninja, Esptool
API	SoupUI, Postman, Pulse
Web services	Wega, PwnXSS, Burpsuite
Reverse engineering	Mobile Security Framework, Apktool

Table 5.3: Application Suite - Software Tools

6 Cases

All the cases are using the methodologies and tools of the created framework as guidance for the tests. All the cases are grey box testing operations as we have partial knowledge of how the devices operate.

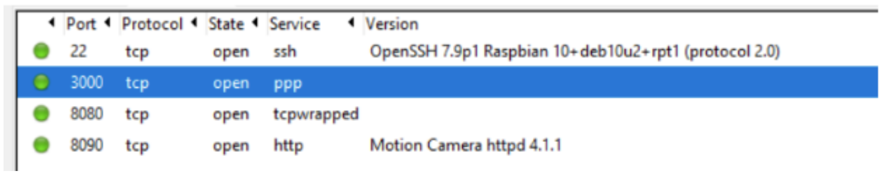
6.1 IoT beehive revision 1

Company X have created a Internet connected beehive that needs to be penetration tested. The device has sensors to monitor the heat, weight, audio of the beehive combined with camera monitoring. The device gets power from a POE adapter which subsequently transfers the collected data from the sensors. The scope for this test is both software and hardware security flaw analysis. As such we've been given network and hardware access to the device.

Device reconnaissance

The first plan of action when using the framework is to gather useful information regarding the device. This is both done for the hardware and the software of the device. As we have physical access to the device, we can simply observe the hardware that is used. In this case it is a Raspberry Pi B3+ single board computer. It uses an external storage solution in the form of an SD card.

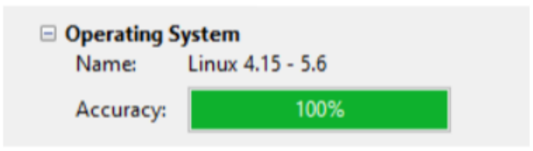
Moving on to the software reconnaissance part of the framework. As we are connected to the same network as the device, we tried scanning the device for any open ports. Here we can find a plethora of different ports open.



Port	Protocol	State	Service	Version
22	tcp	open	ssh	OpenSSH 7.9p1 Raspbian 10+deb10u2+rpt1 (protocol 2.0)
3000	tcp	open	ppp	
8080	tcp	open	tcpwrapped	
8090	tcp	open	http	Motion Camera httpd 4.1.1

Figure 6.6: A portscan using Zenmap of the beehive

We can also confirm that the device without doubt runs some sort of Linux based operating system. As the device is using a Raspberry Pi, we can most likely assume that it runs Raspberry Pi OS (formerly known as Raspbian) with the help of the SSH server information.



Operating System
Name: Linux 4.15 - 5.6
Accuracy: 100%

Figure 6.7: Device enumeration using Zenmap of the beehive

The next plan of action is to grab the banners for each of the ports which is done automatically with Zenmap. We can observe which version and service is running on each of the ports.

Port Information	
22	ssh - OpenSSH server Version 7.9p1
3000	ppp
8080	tcpwrapped
8090	http - Motion Camera httpd 4.1.1

Testing all missing service version ports in a web browser reveals additional information. Port 3000 is a http service called Grafana used for graphing data. This would make sense as the device collects various amounts of sensor data. It is running a non-up to date version of the service (v7.5.7).

The Port 8080 is a live http camera feed from the built in camera header on the Raspberry Pi. It is a service called Motion 4.1.1 which seems to be the same as detected on port 8090. There is a sub-directory on the website for controlling camera functions which does not have any form of authentication. Here it seems like we can write and configure certain camera functions. There is also the ability to restart and stop the service by clicking on one of the html links which is done with a simple POST.

Moving on to the active reconnaissance part of the framework. Running a directory traversal script on all the ports revealed nothing of value.

Device exploitation

Starting with the user software, there are a lot of entry points for this device. It has many open ports with one being the camera feed.

As the device uses Raspbian as its operating system, we can test the default username and password for SSH. By scanning the device with **Zenmap** we found a server on the device used for the camera interface. Without any authentication, the camera could be accessed and controlled through the web interface. With that we created a simple python script to send requests to restart the camera multiple times a second. This ultimately crashed the camera web server.

Having exhausted all possibilities of doing remote exploitation we move on to the hardware.

Following the framework, we unplug the power of the device and connect the SD card to a computer to make a firmware dump of the SD card. After that is done, we reconnect the original SD card to the IoT device and connect power. This does create downtime for the IoT device, which can be noticeable but, in our case, it wasn't an issue.

With the SD card dump, we are not bound to the physical location of the IoT device anymore. As the IoT device uses "off the self-hardware", the device can be recreated and set-up anywhere to be penetration tested.

As we now also have access to the complete dump of the firmware, we can perform firmware analysis to see if the disk image contains any sensitive information. Opening the disk image in Autopsy reveals the complete filesystem along with all the running scripts. We see the default "pi" user home folder containing a lot of different files. We can observe that the code responsible for managing the sensor data is written in python and is in the home folder of the pi user. This code is not compiled which makes it easy to analyze its logic. The script contains in plaintext the IP address of the receiving server, the server is publicly open to the Internet and could be accessed. As the script does not contain any form of authentication a malicious user could spam fake data by knowing the payload format. A comment in the code reveals how the payload in a post request to the server should be structured.

Moving on we see a script that gets imported by the main script which is used for just sending the weight to separate servers. That does use authentication in the form of security certificates. However, these certificate files are located on the device itself and thus can easily be copied.

Continuing, we can see that there are remnants of postgres files indicating that there is a postgres server running. Looking at the postgres history we can determine that this server was used to store sensor data.

We can also see a Wi-Fi name and password in plaintext found inside the wpa_supplicant.conf file. This is most likely left over from a development test as our supplied product uses POE for power delivery.

Device findings

We found by using the framework we were able to dump the disk and see how the device works. With that we found some sensitive data that no user should have access to. As there are no physical barriers or alarms in any way to block a malicious user, anyone can make a copy of the disk with off-the-shelf hardware and then steal the certificates, IP addresses and or the entire running code.

The camera interfaces could be accessed by being on the same network as no authentication was needed. A script could be written to restart the camera continuously as a denial-of-service.

With access to the IP address and its simplistic authentication, a rouge user could spam fake sensor data to the external servers making them cluttered.

6.2 IoT beehive revision 2

Company X have created an Internet connected beehive that needs to be penetration testing. The scope for this test is software exploitation. As such we've have been given the device to be tested. The device uses LTE-M to send and receive sensor data. This device, unlike the first revision, does feature some hardware tamper deterrent in the form of a pull up resistor connected to an IO pin.

Device reconnaissance

As we have been given the physical device we can find that the smart beehive uses a Pycom "FiPy" microcontroller. As the schematics is public for the device, we can see some useful information about the hardware details. The microcontroller is based on an ESP32 with extra support for other "IoT specific" networks such as LoRa, Sigfox and Bluetooth Low Energy (LE). It uses firmware that works as a micropython interpreter. As such the device will run micropython code. We can see that also supports TLS and different forms of hashing algorithms which could become a hindrance to the exploitation part of the framework if they are implemented. We can also see in the documentation that the microcontroller has support for multiple wired data protocols. These protocols include UART, I^{2}C, JTAG, SD cards etcetera.

Device exploitation

Due to the limited access and interfaces to the device in a real life scenario, the attack surface is very small. As there are little to no inputs in which we can follow with the framework we are left with the mobile network connection. While infeasible to physically test in our case, by using a software defined radio we could intercept the data. If the signal generated by SDR is more powerful than the other cell tower, then the device will automatically connect to the SDR. With that a user could capture all the data traffic sent to and from the device. That would give the destination address to which the data is sent without access to the source code. By creating a script, you could then spam send to that address with bogus sensor data, rendering the receiving end unusable.

Device findings

There were little to no findings on this device when following the framework. The mobile networking can be intercepted capture the data sent to and from the device. However, that step requires an SDR and accompanying software that can emulate a rouge cellular base station. In conclusion the device was found to be rather secure due to the lack of interfaces and its small attack surface.

Device discussion

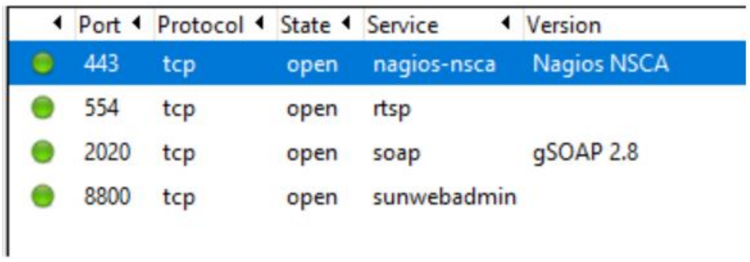
As Fipy uses serial/UART to upload the python files we could connect to PIN 0 (RX0) and PIN 1 (TX0) and thus get access to the running scripts. When we connected via serial, we could find sensitive information in an non compiled config file. This file contained variables for the protocol used, port number, a hardcoded authentication message and IP addresses of the receiving server. This could be seen as a vulnerability as this information could be classified as sensitive. However, this was left out of the device findings as this attack requires physical access which was outside of the scope.

6.3 IoT camera

Company Y uses an Internet connected camera for monitoring a smaller storehouse. The camera needs to be penetration tested for security vulnerabilities. The scope for this penetration test is software only. We have an image of what the device looks like, the end user Android app and we have access to the same network as the device is connected to.

Device reconnaissance

The first plan of action when using the framework is to gather useful information regarding the device. In our case as we are limited to network access, we'll need to move to the software reconnaissance part of the framework. As we are connected to the same network as the device, we try scanning the device for any open ports. Here we can find many different ports open.



	Port	Protocol	State	Service	Version
●	443	tcp	open	nagios-nsca	Nagios NSCA
●	554	tcp	open	rtsp	
●	2020	tcp	open	soap	gSOAP 2.8
●	8800	tcp	open	sunwebadmin	

Figure 6.8: A portscan using Zenmap of the camera

By using the banner grabbing of Zenmap we can see that the vendor is a TP-link. By looking up IP cameras of the same maker and cross-referencing it with the photo we can boil it down to two different models which it can be. These devices support a multitude of different features, such as microphone, speaker, camera movement, OTA (Over-the-air) updates and SD card storage or cloud upload.

Moving on in the framework we get to the end user app decompilation stage. As a non-configured APK app was supplied, it was decompiled and analyzed using the program "Mobile Security framework" (MobSF). An attempt to find the URL for the over-the-air update was done using the string search function. It gave some results when searching for web addresses, but nothing related to the over-the-air update was found. By looking at the app code, it seems like the URL for the binary file is generated dynamically from a set of variables.

There was a mention of a debug mode in the file in which you can send and update a firmware bin file. This was found in the following file:

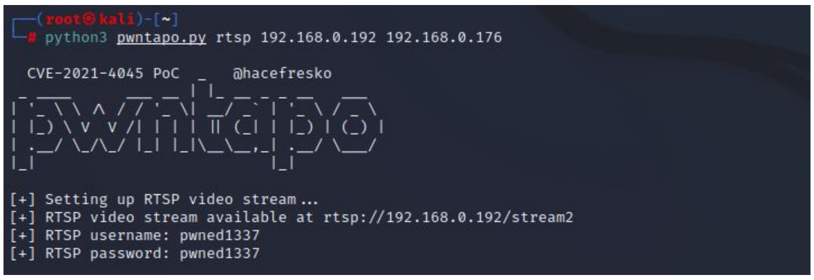
`com/tplink/iot/debug/device/DeviceDebugActivity.java`

Continuing with the information we have. As we roughly know the model of the device, the next step when following the framework is to lookup known vulnerabilities as we know the hardware details. By searching for the model number at the National Vulnerability Database we found CVE-2021-4045 [48] that may affect this device if its firmware has not been updated. Looking into the device manuals, the firmware update mechanism needs to be triggered manually through their app. Without knowing the firmware version, we try launching the known exploit.

Device exploitation

The found vulnerability CVE-2021-4045 is an exploit that affects this IP camera with a firmware version 1.1.15 or below. The exploit makes use of a vulnerability present in the uHTTPd binary that runs as root by default [48]. This was found by doing reverse engineering on the uHTTPd binary with the tool Ghidra. It was discovered that an attacker could inject code that was directly executed by the camera without any authentication by sending a "set language" post request. Remote code execution would give an attacker full control over the device.

By following a write-up on the CVE [49], we could obtain a python script that was used successfully to obtain a root shell against the camera. Furthermore, this script has two different attack modes, one to establish the root shell and another mode to access the camera's live stream unauthorized. So, by using the python script we were able to spawn a root shell against the camera as-well as viewing the cameras video stream over the rtsp (Real-Time Streaming Protocol) protocol, the video could be accessed by loading the RTSP stream in VLC media player.



```
┌──(root㉿kali)-[~]
└─# python3 pwntapo.py rtsp 192.168.0.192 192.168.0.176

  CVE-2021-4045 PoC  _   @hacefresko
                    | |
 _ ____      ___ __ | |_ __ _ _ __   ___
| '_ \ \ /\ / / '_ \| __/ _` | '_ \ / _ \
| |_) \ V  V /| | | | || (_| | |_) | (_) |
| .__/ \_/\_/ |_| |_|\__\__,_| .__/ \___/
|_|                          |_|

[+] Setting up RTSP video stream...
[+] RTSP video stream available at rtsp://192.168.0.192/stream2
[+] RTSP username: pwned1337
[+] RTSP password: pwned1337
```

Figure 6.9: Launching the script with the rtsp parameter

Device findings

The device findings of the camera were a CVE that allowed us to gain complete access over the device by spawning a root shell. Additionally, the vulnerability allowed us to view the video stream without any authentication. The CVE still exists since the device required manual firmware updates which had not been done by company Y.

6.4 IoT Wi-Fi Smart Lamp

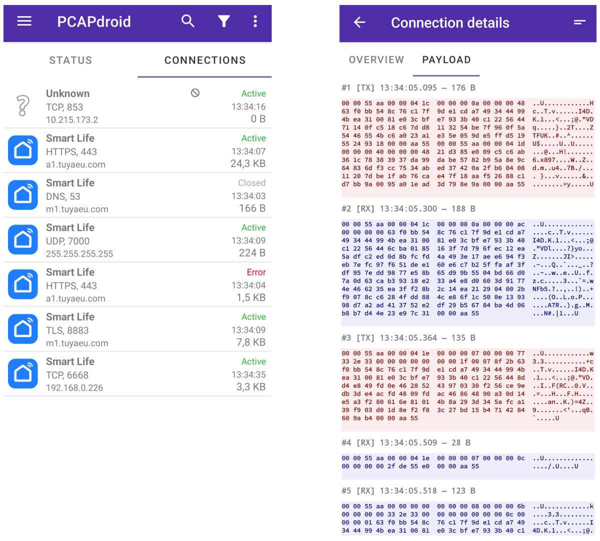
Company Z sells a smart LED light that is controlled over Wi-Fi. It's an RGB color spotlight lamp which uses a GU10 connector. The device can be bought over the counter at many retail stores that sell home improvement or electrical equipment. The scope of this case is software with limitations to the android application, network, and hardware access. The device has been connected to the network using the supplied end user application.

Device reconnaissance

The first step when following the framework is passive software reconnaissance followed by active reconnaissance.

The app supplied with the lamp is called "smart life" on Android based devices. Its package name is called "com.tuya.smartlife". With that we can google and find that Tuya is Chinese based company focusing on creating IoT solutions as an OEM. As such we can conclude that Company Z sells a re-branded Tuya device. These devices are extensively documented online, and users have created custom modules for different programming languages to control these devices. With a simple search we can find that the devices use three parameters for controlling the device. There is the IP-address, "local key" and the "device id". The device id can be found when using tools such as "tinytuya scan". However, the "local key" is more secretive and requires some prodding around to find. Looking at the "tinytuya" python library we can find that the lamps all use port 6668 to send and receive data.

Moving on to the active reconnaissance stage of the framework. As the device comes with an app, we can capture the data being sent and received using an app such as PCAP-Droid.



```
Unknown        Active
TCP, 853       13:34:16
10.215.173.2   0 B

Smart Life     Active
HTTPS, 443     13:34:07
a1.tuyaeu.com  24,3 KB

Smart Life     Closed
DNS, 53        13:34:03
m1.tuyaeu.com  166 B

Smart Life       Active
UDP, 7000        13:34:09
255.255.255.255  224 B

Smart Life     Error
HTTPS, 443     13:34:04
a1.tuyaeu.com  1,5 KB

Smart Life     Active
TLS, 8883      13:34:09
m1.tuyaeu.com  7,8 KB

Smart Life     Active
TCP, 6668      13:34:35
192.168.0.226  3,3 KB
```

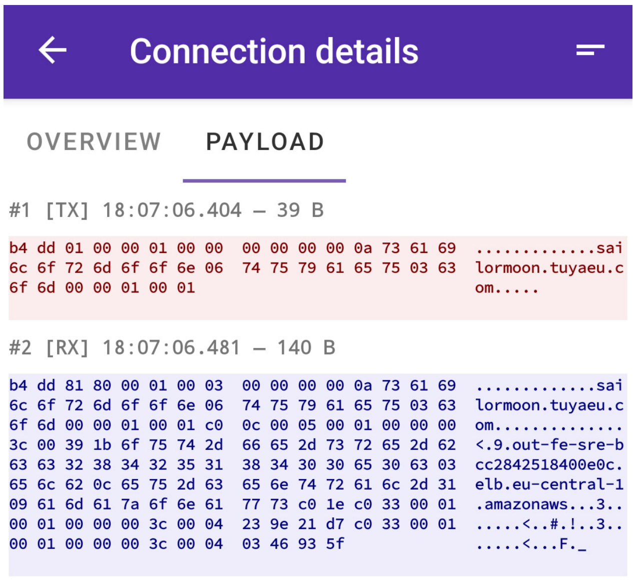
```
#1 [TX] 13:34:05.095 — 176 B

00 00 55 aa 00 00 04 1c  00 00 00 0a 00 00 00 48  ..U............H
63 f0 bb 54 8c 76 c1 7f  9d e1 cd a7 49 34 44 99  c..T.v......I4D.
4b ea 31 00 81 e0 3c bf  e7 93 3b 40 c1 22 56 44  K.1...<...;@."VD
71 14 0f c5 18 c6 7d d8  11 32 54 be 7f 96 0f 5a  q.....}..2T....Z
54 46 55 4b c6 a0 23 a1  e3 5e 05 9d e5 ff d5 19  TFUK..#..^......
55 24 93 18 00 00 aa 55  00 00 55 aa 00 00 04 1d  U$.....U..U.....
00 00 00 40 00 00 00 48  21 d3 85 e0 09 c5 c6 ab  ...@...H!.......
36 1c 78 38 39 37 da 99  da be 57 82 b9 5a 8e 9c  6.x897....W..Z..
64 83 6d f3 cc 75 34 ab  ed 37 42 0a 2f b6 04 08  d.m..u4..7B./...
11 20 7d be 1f ab 76 ca  e4 7f 18 aa f5 26 88 c1  . }...v......&..
d7 bb 9a 00 95 a0 1e ad  3d 79 8e 9a 00 00 aa 55  ........=y.....U

#2 [RX] 13:34:05.300 — 188 B

00 00 55 aa 00 00 04 1c  00 00 00 0a 00 00 00 ac  ..U.............
00 00 00 00 63 f0 bb 54  8c 76 c1 7f 9d e1 cd a7  ....c..T.v......
49 34 44 99 4b ea 31 00  81 e0 3c bf e7 93 3b 40  I4D.K.1...<...;@
c1 22 56 44 6c ba 01 85  16 3f 7d 79 6f ec 12 ea  ."VDl....?}yo...
5a df c2 ed 0d 8b fc fd  4a 49 3e 17 ae e6 94 f3  Z.......JI>.....
eb 7e fc 97 f6 51 de e1  60 e6 c7 b2 5f fa af 3f  .~...Q..`..._..?
df 95 7e dd 98 77 e5 8b  65 d9 9b 55 04 bd 66 d0  ..~..w..e..U..f.
7a 0d 63 ca b3 93 18 e2  33 a4 e8 d0 60 3d 91 77  z.c.....3...`=.w
4e 46 62 35 ea 3f f2 8b  2c 14 ea 21 29 04 00 2b  NFb5.?..,..!)..+
f9 07 8c c6 28 4f dd 88  4c e8 6f 1c 50 0e 13 93  ....(O..L.o.P...
98 d7 a2 ad 41 37 52 e2  df 29 b5 67 84 ba 4d 06  ....A7R..).g..M.
b8 b7 d4 4e 23 e9 7c 31  00 00 aa 55              ...N#.|1...U

#3 [TX] 13:34:05.364 — 135 B

00 00 55 aa 00 00 04 1e  00 00 00 07 00 00 00 77  ..U............w
33 2e 33 00 00 00 00 00  00 00 1f 00 07 8f 2b 63  3.3...........+c
f0 bb 54 8c 76 c1 7f 9d  e1 cd a7 49 34 44 99 4b  ..T.v......I4D.K
ea 31 00 81 e0 3c bf e7  93 3b 40 c1 22 56 44 8d  .1...<...;@."VD.
d4 e8 49 fd 0e 46 28 52  43 97 03 30 f2 56 ce 9e  ..I..F(RC..0.V..
db 3d e4 ac fd 48 09 fd  ac 46 86 48 90 a3 0d 14  .=...H...F.H....
e5 a3 f2 80 61 6e 81 01  4b 8a 29 3d 34 5a fc a1  ....an..K.)=4Z..
39 f9 03 d0 1d 8e f2 f8  3c 27 bd 15 b4 71 42 84  9.......<'...qB.
60 9a b4 00 00 aa 55                               `.....U

#4 [RX] 13:34:05.509 — 28 B

00 00 55 aa 00 00 04 1e  00 00 00 07 00 00 00 0c  ..U.............
00 00 00 00 2f de 55 e0  00 00 aa 55              ..../.U....U

#5 [RX] 13:34:05.518 — 123 B

00 00 55 aa 00 00 00 00  00 00 00 08 00 00 00 6b  ..U............k
00 00 00 00 33 2e 33 00  00 00 00 00 00 00 0c 00  ....3.3.........
00 00 01 63 f0 bb 54 8c  76 c1 7f 9d e1 cd a7 49  ...c..T.v......I
34 44 99 4b ea 31 00 81  e0 3c bf e7 93 3b 40 c1  4D.K.1...<...;@.
```

Figure 6.10: Data interception and viewing of the smart life app using PCAPDroid

We see that it sends data to multiple servers and the device itself. Figure 6.10 shows the data sent to the lamp itself. The data seems to be encoded in some sort of way.

We did a network capture during the device setup in which you send over the SSID and password for your Wi-Fi. The SSID and password does not seem to be sent directly to the device, instead a middleman server by Tuya gets the data and returns some sort of token that the device receives instead. The middleman server uses HTTPS which makes this hypothesis difficult to confirm. A try to install a root certificate to decrypt the HTTPS data wasn't possible as the server rejected the certificate and just closed the connection.

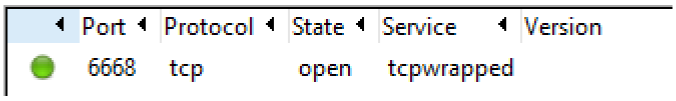
Some data which most likely is the "token" is sent over unencrypted TCP to the device itself which could be intercepted and in theory replayed by an external user doing an MITM attack. However, this wasn't tested.



```
#1 [TX] 18:07:06.404 – 39 B

b4 dd 01 00 00 01 00 00  00 00 00 00 0a 73 61 69  .............sai
6c 6f 72 6d 6f 6f 6e 06  74 75 79 61 65 75 03 63  lormoon.tuyaeu.c
6f 6d 00 00 01 00 01                               om.....

#2 [RX] 18:07:06.481 – 140 B

b4 dd 81 80 00 01 00 03  00 00 00 00 0a 73 61 69  .............sai
6c 6f 72 6d 6f 6f 6e 06  74 75 79 61 65 75 03 63  lormoon.tuyaeu.c
6f 6d 00 00 01 00 01 c0  0c 00 05 00 01 00 00 00  om..............
3c 00 39 1b 6f 75 74 2d  66 65 2d 73 72 65 2d 62  <.9.out-fe-sre-b
63 63 32 38 34 32 35 31  38 34 30 30 65 30 63 03  cc2842518400e0c.
65 6c 62 0c 65 75 2d 63  65 6e 74 72 61 6c 2d 31  elb.eu-central-1
09 61 6d 61 7a 6f 6e 61  77 73 c0 1e c0 33 00 01  .amazonaws...3..
00 01 00 00 00 3c 00 04  23 9e 21 d7 c0 33 00 01  .....<..#.!..3..
00 01 00 00 00 3c 00 04  03 46 93 5f              .....<...F._
```

Figure 6.11: Data interception and viewing of the smart life app using PCAPDroid

A notewothy observation in figure 6.11 is that the app sends unknown data to a "sailor-moon.tuyaeu.com" AWS domain. Moving on to scanning the device we can confirm that it uses port 6668.



	Port	Protocol	State	Service	Version
●	6668	tcp	open	tcpwrapped	

Figure 6.12: A portscan using Zenmap of the smart lamp

As software-based testing was out of the question, the next step in the framework is hardware-based hacking. Looking at the lamp shows that it does not have any physical screws or any other way to access the PCB. As such we resulted in cutting the plastic casing in half. This revealed the 230v lamp power connector, its power circuit, a 3.3v converter, an unknown microcontroller, and the LEDs themselves. The LEDs are separated into a daughterboard. By looking at other people doing the same thing, they found that the board include RX and TX pins for programming the chip [50]. They were able to see a UART log containing the entire bootup process. They also noted that the device stores the Wi-Fi name, password, "device ID" and the "Local Key" in plaintext inside the firmware dump. The firmware dump was done using ESPtool.

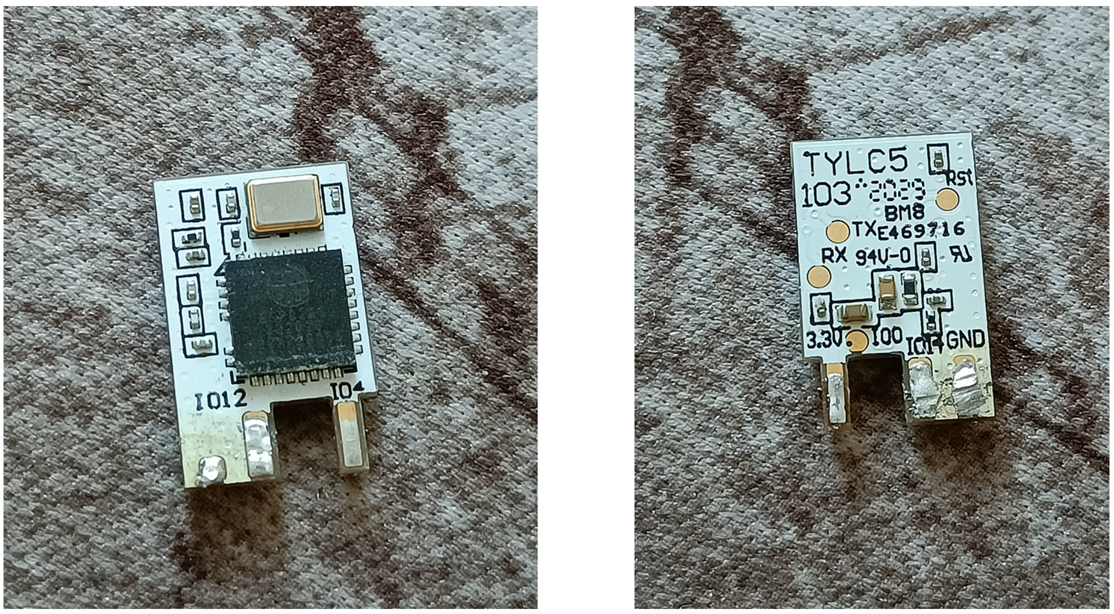


Figure 6.13: The esp2866 board desoldered from the main power circuit

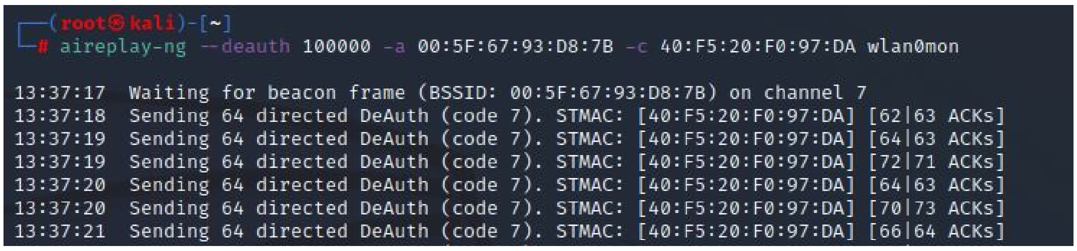
The microcontroller is separated onto a board that is soldered onto the power circuit board. So, after desoldering the device we can also see some more information. The device is based upon an ESP8285 chip as noted on the chip die printing. Looking at the PCB silkscreen we can also see that it is named "TYLC5". After an online search we can find that the board is documented on the Tuya developer website [51].

More interestingly we can also find some debug pads on the PCB. Pads labeled as RX, TX, RST and IO0 can be found. This indicates that UART is also available on this device. As the pads are very small, we used 0.4mm copper wire to solder directly to the pads. Then we connected RX, TX, 3.3v and GND to their respective pins on a client computer UART serial adapter. As stated in the documentation of the microcontroller, the crystal oscillator sets the baud rate for UART. Unfortunately, we didn't get any data from the device despite resetting the device with the RST pad. This is most likely due to the fragile nature of wiring up the device.

Device exploitation

As we didn't get any data from the UART we moved on to testing for common network vulnerabilities. As the device is using an ESP8285 which does not support WPA3 it connects to the Wi-Fi using downgraded WPA2.

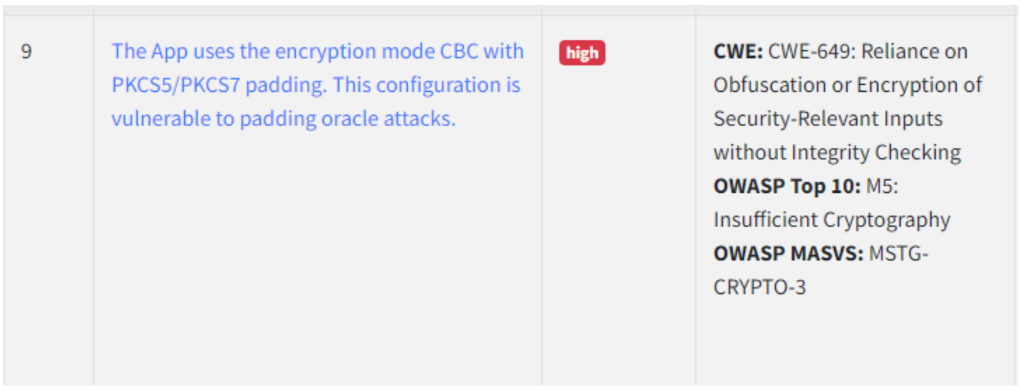
As the device itself does not contain any sensitive information we strive to use it as a way into the internal network. By using **Aireplay-ng** to scan the network, it was possible to identify the lamp by its MAC address. Thereafter, by sending deauthentication packets to the lamp it became inaccessible, and we could capture the 4-way handshake when it tried to reconnect to the same access point. **Aircrack-ng** was then used to crack the obtained password hash from the handshake, thereby network access has been accomplished by targeting the communication channel of the lamp. Even though it is the network that is vulnerable to this attack, it was found by targeting the smart lamp. We do need to note that this attack can be done on any device that is using the older Wi-Fi WPA2 authentication standard. However, since the device does not support WPA3, It would've been impossible to fix without replacing the device.



```
┌──(root㉿kali)-[~]
└─# aireplay-ng --deauth 100000 -a 00:5F:67:93:D8:7B -c 40:F5:20:F0:97:DA wlan0mon

13:37:17  Waiting for beacon frame (BSSID: 00:5F:67:93:D8:7B) on channel 7
13:37:18  Sending 64 directed DeAuth (code 7). STMAC: [40:F5:20:F0:97:DA] [62|63 ACKs]
13:37:19  Sending 64 directed DeAuth (code 7). STMAC: [40:F5:20:F0:97:DA] [64|63 ACKs]
13:37:19  Sending 64 directed DeAuth (code 7). STMAC: [40:F5:20:F0:97:DA] [72|71 ACKs]
13:37:20  Sending 64 directed DeAuth (code 7). STMAC: [40:F5:20:F0:97:DA] [64|63 ACKs]
13:37:20  Sending 64 directed DeAuth (code 7). STMAC: [40:F5:20:F0:97:DA] [70|73 ACKs]
13:37:21  Sending 64 directed DeAuth (code 7). STMAC: [40:F5:20:F0:97:DA] [66|64 ACKs]
```

Figure 6.14: Sending deauthentication packets to the lamp

The only thing left to test was to reverse engineer the corresponding Android application (Android APK) to see if there were any vulnerabilities. Weak encryption was found in some of the Java files, but we could not conclude if it affected the communication with the smart lamp.



9	The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks.	high	**CWE:** CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking **OWASP Top 10:** M5: Insufficient Cryptography **OWASP MASVS:** MSTG-CRYPTO-3

Figure 6.15: MobSF Scan; Weak Encryption

Device findings

We did find by following the framework that the device stores the Wi-Fi username and password in plaintext on the device. So, if a corporation or an individual user throws out the LED lamp, you could tear down the device, connect it to UART and display all its data. As described in the device reconnaissance stage, we were unable to do this most likely due to poor wiring.

The lamp did not support WPA3, which resulted in a connection over downgraded WPA2. The access point was configured for WPA2/WPA3, which made it possible to establish a connection. By using WPA2, the handshake could be obtained when the the lamp reconnects to the network after being deauthenticated and thus the password hash could be cracked to gain the Wi-Fi password.

Additionally, the application supplied with the device had weak encryption, but it could not be proven if it affected the communication between the device itself and the Tuya cloud.

7 Results and Analysis

This section is divided into three sub-sections that represent the results and analysis of each research question respectively.

7.1 How does the existing standards/framework apply to IoT penetration testing? (RQ1)

The results from section 4.1, show how the existing frameworks address the IoT common vulnerabilities from OWASP [16]. The theoretical comparison is summarized in table 7.4. The common vulnerability **Insecure Default Settings** is not included in the table since the testing methods are the same which are covered by the other common vulnerabilities. Additionally, the common vulnerability **Lack of Device Management** is not included as it's not within the scope for a penetration test.

The parameter "Yes" indicates that the respective framework has included methodologies and tools to conduct a test towards the specific vulnerability.

The parameter "No" indicates the respective framework does not cover any methodologies or tools to conduct a test towards the specific vulnerabilities.

The parameter "Partially" indicates that the respective frameworks have included methodologies and tools to test a specific vulnerability to some degree. The parameter "Partially" is used when a framework does include ways to test a vulnerability but cannot be applied to IoT devices or leaving out IoT specific communication models and hardware aspects.

Common vulnerability	**PTES**	**NIST 800-115**
Weak, Guessable, or Hardcoded Passwords	Yes	Yes
Insecure Network Services	Yes	No
Insecure Ecosystem Interfaces	Partially	No
Lack of Secure Update Mechanism	No	No
Use of Insecure or Outdated Components	Partially	Partially
Insufficient Privacy Protection	No	Partially
Insecure Data Transfer and Storage	Partially	No
Lack of Physical Hardening	No	No

Table 7.4: Summary how existing frameworks addresses common vulnerabilities

The table provides the results from how PTES and NIST 800-115 could be used to address and conduct a test for the respective common vulnerability. This has been theoretically examined.

The results in table 7.4 indicate that the chosen frameworks are very limited in their coverage against IoT penetration testing. Both frameworks could partially be used to test some of the vulnerabilities but as the result shows neither PTES and NIST 800-115 include tests targeting IoT specific protocols and hardware aspects.

The results indicates that if the IoT device is using any other communication protocol than Wi-Fi, then both the frameworks are very limited in its coverage. However, both the frameworks could be a valuable source of information for testing some of the common vulnerabilities as **Weak, Guessable, or Hardcoded Passwords** and **unsecure Network Services**. Additionally, the frameworks is a valuable source of information regarding what areas should be tested and how the testing process and reporting shall be done. The research done about how the existing framework addresses the common IoT vulnerabilities resulted in that PTES and NIST 800-115 lacks information about testing the following aspects:

- IoT specific communication protocols as Zigbee, Z-Wave and LoRa.
- Hardware and physical access testing.
- Firmware testing.
- API testing.
- File system exploration.
- Reading out data from external storage.

From the results, we can see that the frameworks do not cover testing of IoT-specific common vulnerabilities. The frameworks do not include methods to mitigate and test most of the vulnerabilities. PTES does cover more testing methodologies and tools that can be used in a penetration test than NIST 800-115. On the other hand, NIST 800-115 provides more information about the legal aspects of a test and how it can be used to create and maintain a security assessment. Based on these findings, PTES would be more appropriate to use as guidance when conducting a test, even though it may lack information for testing IoT devices. This concludes that there is no penetration testing framework that covers testing IoT technologies.

7.2 Cases - Mapping vulnerabilities that were found when using the artifact (RQ2)

Table 7.5 summarizes the findings when conducting penetration tests for each case in Section 6. The table describes the scope and findings of each case and whether the device findings were an exploit that could be linked to a common vulnerability.

Case	**Device findings**	**OWASP vulnerabillity**
Beehive rev 1 Scope: Full acess	Found sensitive data and got full controll over the device. The camera could be controlled and was vulnerable to a DoS attack.	Insecure Ecosystem Interfaces, Insecure Data Transfer and Storage, Lack of Physical Hardening
Beehive rev 2 Scope: No physical access	No findings without physical access.	None
IoT Camera Scope: No physical access	Known CVE was found which could be exploited due to old firmware. Could gain complete access of the device by creating a reverse root shell. Could view the video stream over rtsp unauthorized.	Insecure Network Interfaces, Lack of Secure Update Mechanism Use of Insecure and Outdated Components
IoT smart-lamp Scope: Full access	SSID and WiFi password was stored in plaintext. Wi-Fi and App communication uses weak encryption. Deauthenticated to capture handshake.	Insecure Data Transfer and Storage Insecure Default Settings Lack of Physical Hardening

Table 7.5: Cases - Mapping vulnerabillity

From the results of the tests done on Beehive revision 1, it was shown that it was possible to gain full control over the device and to view sensitive information by having physical access. The device findings about the vulnerable camera interface and the corresponding web server could be linked to the OWASP vulnerability **Insecure Ecosystem Interfaces**. It links to this vulnerability because it is a device-related component that had issues due to a lack of authentication; the camera could be controlled without any authentication. The web interface for the camera didn't use any authentication hence a script could be written to restart the camera continuously as a denial-of-service.

The **Lack of Physical Hardening** and **Insecure Data Transfer and Storage** were additional common vulnerabilities that could be identified. As the device did not have any physical protection, it was possible to dump the firmware and filesystem, thereby we could examine the complete filesystem. The examination of the filesystem resulted in sensitive information being accessed due to a lack of physical hardening and insecure data storage.

The results from the tests done on Beehive revision 2 show that there were no device findings when the scope was limited to physical access. When redefining the scope to physical access, sensitive information could be retrieved by filesystem exploration and code analysis. The pycom device used LTE-M as its communication channel which made it hard to test and attack remotely as a result. With that, no common vulnerability could be exploited on Beehive revision 2 with the original scope.

From the results of the tests done on IoT camera, it is shown that it was possible to gain full control over the device by spawning a root shell. Additionally, the rtsp video stream could be accessed over the network without any authentication. Both exploits were found by a CVE present in uHTTPd binary. One of the identified vulnerabilities is **Insecure Network Interfaces** as the uHTTPd network/web service was exposed to the rest of the network and used to spawn the root shell and access the rtsp video stream. The network service was concluded to be an insecure network service running on the device which led to unauthorized remote control and compromised integrity. As the vulnerability was present in the uHTTPd binary and was proven to be an insecure component that compromised the device, the OWASP vulnerability **Use of Insecure and Outdated Components** could be identified. Furthermore, the CVE could be used due to the fact that no firmware update had been done, which required a manual update through the supplied mobile app. The OWASP vulnerability **Lack of Secure Update Mechanism** could be identified. Additional firmware testing was done, but nothing was found. The identified CVE was due to a physical attack to retrieve the uHTTPd binary, therefore we can argue that there was a lack of physical hardening even though we not specifically exploited this.

The tests done on the smart lamp resulted in two major device findings. Firstly, in the reconnaissance stage we found that the Wi-Fi SSID and password are stored in plaintext on that specific SoC by doing physical testing. As such whenever a firmware dump is done, opening the bin file in a hex editor reveals the SSID and password. The OWASP vulnerabilities **Insecure Data Transfer and Storage** and **Lack of physical hardening** links to the device findings as sensitive information were stored insecurely. In addition to sensitive information being stored in plaintext, it was found that the supplied app used for communication between the device and the cloud service used weak encryption algorithms somewhere within the device ecosystem. The other OWASP vulnerability **Insecure Default Settings** was identified, as the use of WPA3 was not supported. This makes the device vulnerable to deauthentication attacks and made it possible for us to capture the handshake and obtain the password hash to gain network access by cracking it.

To summarize, the results of the device findings show that three out of four tested devices had common vulnerabilities that could be exploited. The device indicates that the IoT devices were more vulnerable to physical-based attacks than network based attacks. Multiple common vulnerabilities could be identified and exploited, which compromised the confidentiality, integrity, and availability of these IoT devices. The results also show that the created generic framework could be used in conjunction when conducting a penetration test against IoT devices.

7.3 The challenges when testing IoT devices (RQ3)

We found that there are a multitude of different challenges when security testing IoT devices compared to traditional computing and networking devices.

To start, there is no uniform set of penetration testing standards that can be used as guidance for testing IoT devices as shown in this research. This lack of standardization could make it difficult for penetration testers to conduct a test targeting IoT devices. Each type of IoT device uses unique communication protocols and architectures, resulting in a complex and challenging penetration testing process. Understanding the complexity of different IoT devices and their ecosystems required us to gain specialized knowledge about each device which was very time consuming. This combined with a broad set of different networking and hardware standards for IoT makes an automated penetration testing solution more difficult.

The tests done on beehive revision 2 indicate that physical protection greatly shrinks the attack surface, which resulted in no common vulnerability that could be exploited. With no physical access combined with the devices' usage of IoT-specific communication protocols such as LTE-M, the testing phase becomes more complicated. The attack surface shrinks and additional hardware components as an SDR are needed to test the communication channel to test the device. This raises the starting costs and resources compared to non IoT penetration testing.

Devices often automatically interact with external cloud services which may prove a hindrance to penetration testing. This was found when testing the case for the Wi-Fi smart lamp. Some parts of lamp initial configuration are sent and processed on external servers which are not controlled by the user who owns the IoT product. This can make penetration testing difficult as there is often little to no way in getting approval to penetration test the cloud services themselves. Finally, ethical considerations should be considered, such as the potential harm that could arise from exploiting vulnerabilities in real-world IoT deployments. While penetration testing a smart lamp may not have ethical complications, conducting a test on a more critical system such as smart healthcare products may do so.

8 Discussion

The purpose of this report was to find out if the existing penetration frameworks could be applied to IoT. To continue, to answer what challenges there are when testing these IoT devices compared to non IoT devices. Finally, when presented with a generic penetration testing framework for IoT if applied to test cases, could any common IoT vulnerabilities be found when cross referenced with OWASP Top 10 for IoT?

The answers to these research questions were found through a multi-method scientific process.

To answer the first question, a theoretical comparative study was done on the existing penetration testing frameworks. The frameworks were evaluated to see if they could be used as guidance in a penetration test for only IoT. OSSTM and OWASP for IoT were excluded as it did not fit the criteria set for being defined as a framework. PTES and NIST 800-115 did fit the criteria in being defined as a framework. In comparison with the study by Shanley and Johnston [5], our research resulted in defining PTES as a framework to which they did not conform. The next section investigated how well those frameworks could be applied to testing the common vulnerabilities for IoT. Testing IoT devices with PTES and NIST 800-115 is not feasible due to the lack of adequate testing methodologies that can address the common vulnerabilities and attack surface. This proves that there was no penetration testing framework for IoT devices. Building a complete framework for penetration testing is a challenging task, specifically as IoT devices exhibit variations in form factors, functionalities, and architectures. The created framework in this paper aims to cover most of the IoT specific challenges and protocols in this challenging area of research. From the evaluation of the existing frameworks, we found that it would not be possible to apply its testing methodologies to IoT as there were many attack and testing vectors missing. Based on the insights obtained from this section, the newly developed framework incorporates previously missing testing methodologies, including hardware and firmware testing, among others.

When it came to finding IoT testing methodologies, it was challenging to find sources with a more comprehensive analysis of the method. Additionally, many sources covered only theoretical methods that had not been applied to real-life devices and protocols.

However, penetration testing frameworks are not the only way to test and enforce IoT security. Other ways to detect security issues in IoT devices are through security standards, security assessments and protocols as shown in the research done by Johansson [52].

The second part of this research was testing the security on a set of devices by following the framework. Moreover, we investigated if any of the common vulnerabilities seen in the OWASP Top 10 could be identified and exploited. The experiments resulted in finding vulnerabilities on three out of the four tested devices. If the scope for Beehive revision 2 was redefined, we found that all the devices were vulnerable to physical attacks. As these devices are likely to be deployed in areas where physical access could be hard to control [19] or where insiders could access the device unnoticed, physical protection, device management, and anti-tamper mechanisms need to be in place to maintain the device security.

Williams et al. [53], used the vulnerability scanner Shodan to scan consumer IoT devices exposed to the Internet. Their study showed that 20,237 of the 156,680 (12.92%) scanned devices had vulnerabilities of different risks. In our research, the more consumer-oriented products like the camera and the lamp could be exploited. From the cases in this research, we can argue that it was easier and less time consuming to exploit the more commercial products compared to the beehive devices. There were no devices in the cases that used protocols like Zigbee, LoRaWAN or Z-Wave which the framework could be tested against.

Moving on, the design issues of these types of communication protocols are covered by the framework. Assessing the beehive that used LTE-M proved to be challenging, mainly due to the direct form of data communication. When devices use this type of low-power radio communications, penetration testers would require additional hardware testing tools, such as an SDR. In this research, we did not have access to this type of hardware. With that, there were limitations to which devices, protocols, and communication channels could be tested fairly.

From all the knowledge gained, we could show that the security of IoT devices is not particularly strong, which is why they pose a huge security risk when being integrated into daily systems. While these common vulnerabilities are known, manufacturers and developers still fail to mitigate these challenges. The results show that the common vulnerabilities could be detected by following the guidelines in the created framework. This work also discusses other ways to prevent these issues by following security assessments and using protocols with higher security.

9 Conclusions and Future Work

In this thesis, the results show that existing penetration testing frameworks cannot be applied to IoT devices without significant modifications. We also identified the challenges of testing IoT devices and the importance of addressing common vulnerabilities. The report proposed a new framework for testing IoT devices that covers IoT-specific testing methodologies, which were successfully used to detect vulnerabilities in IoT devices. Section 6 with the cases shows how the manufacturers and developers could have identified these common vulnerabilities by having a proper penetration testing framework with guidelines to follow. The results of this research can be used both within the industry and for science, to expand the knowledge and interest about IoT security issues and testing methodologies. The created framework could be used by organizations to enforce security and detect security vulnerabilities among their IoT devices connected to their networks and assets.

In conclusion, as IoT devices often come with security vulnerabilities, it becomes important for the manufacturers, developers, and the consumers to test these devices for vulnerabilities before putting them in production. By conducting regular penetration tests, it would be possible to mitigate the vulnerabilities before a malicious actor can take advantage of it.

Future work from this research could be to conduct penetration tests on a larger set of devices by following the created framework. As this report did not target all IoT specific protocols it would be interesting to develop more comprehensive methods to target protocols like Zigbee, Z-Wave and LoRA. Additionally future work would be to extend the framework with more testing methods. As the results indicate that the tested IoT devices were more vulnerable to physical attacks, it would be interesting to see research about mitigation strategies and testing methodologies targeting the physical security aspects only.

References

[1] "Iot connected devices worldwide 2019-2030," Statista, 05 2022. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "Iot privacy and security: Challenges and solutions," *Applied Sciences*, vol. 10, no. 12, p. 4102, Jun 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/12/4102/pdf

[3] 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, 2019.

[4] H. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," in *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2018, pp. 1–7.

[5] A. Shanley and M. N. Johnstone, "Selection of penetration testing methodologies: A comparison and evaluation," *13th Australian Information Security Management Conference*, p. 65–72, 2015.

[6] G. Yadav, A. Allakany, V. Kumar, K. Paul, and K. Okamura, "Penetration testing framework for iot," in *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*, 2019, pp. 477–482.

[7] G. Yadav, K. Paul, A. Allakany, and K. Okamura, "Iot-pen: A penetration testing framework for iot," in *2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 196–201.

[8] G. Yadav, K. Paul, A. Allakany, and K. Okamura, "Iot-pen: An e2e penetration testing framework for iot," *Journal of Information Processing*, vol. 28, no. 0, p. 633–642, 2020.

[9] I. Brass, L. Tanczer, M. Carr, M. Elsden, and J. Blackstock, "Standardising a moving target: The development and evolution of iot security standards," *SSRN Electronic Journal*, 2018.

[10] J. Saleem, M. Hammoudeh, U. Raza, B. Adebisi, and R. Ande, "Iot standardisation," *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems - ICFNDS '18*, 2018.

[11] Vumetric, "Top 5 penetration testing methodologies and standards," Sep 2019. [Online]. Available: https://www.vumetric.com/blog/top-penetration-testing-methodologies/

[12] EC-Council, "Five methodologies that can improve your penetration testing roi," Mar 2022. [Online]. Available: https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/improve-penetration-testing-roi/

[13] OWASP, "Penetration testing methodologies." [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v41/3-The_OWASP_Testing_Framework/1-Penetration_Testing_Methodologies

[14] P. Herzog and ISECOM, “The open source security testing methodology manual (osstmm),” 2010. [Online]. Available: https://www.isecom.org/OSSTMM.3.pdf

[15] PTES, “Ptes technical guidelines - the penetration testing execution standard,” 2011. [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines

[16] OWASP, “Owasp internet of things,” 2018. [Online]. Available: https://owasp.org/www-project-internet-of-things/

[17] NIST, “Special publication 800-115 technical guide to information security testing and assessment recommendations of the national institute of standards and technology,” Sep 2008. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf

[18] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of Management Information Systems*, vol. 24, pp. 45–77, 01 2007.

[19] K. Rose, S. Eldridge, and L. Chapin, “The internet of things: An overview,” *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.

[20] K. Ashton *et al.*, “That ‘internet of things’ thing,” *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[21] 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, Dec 2019. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0268-2

[22] C.-l. Zhong, Z. Zhu, and R.-G. Huang, “Study on the iot architecture and access technology,” in *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, 2017, pp. 113–116.

[23] “What is iot architecture? guide and examples.” [Online]. Available: https://www.mongodb.com/cloud-explained/iot-architecture

[24] W. Saltzstein, “Bluetooth wireless technology cybersecurity and diabetes technology devices,” *Journal of Diabetes Science and Technology*, vol. 14, no. 3, p. 193229681986441, Jul 2019.

[25] Balbix, “Iot security challenges and problems,” Jan 2020. [Online]. Available: https://www.balbix.com/insights/addressing-iot-security-challenges/

[26] S. Bhatt and P. R. Ragiri, “Security trends in internet of things: A survey,” *SN Applied Sciences*, vol. 3, pp. 1–14, 2021.

[27] N. M. Karie, N. M. Sahri, W. Yang, C. Valli, and V. R. Kebande, “A review of security standards and frameworks for iot-based smart environments,” *IEEE Access*, vol. 9, pp. 121 975–121 995, 2021.

[28] Tutorialspoint, “Types of penetration testing - tutorialspoint,” 2019. [Online]. Available: https://www.tutorialspoint.com/penetration_testing/types_of_penetration_testing.htm

[29] D. N. Astrida, A. R. Saputra, and A. I. Assaufi, “Analysis and evaluation of wireless network security with the penetration testing execution standard (ptes),” *Sinkron: jurnal dan penelitian teknik informatika*, vol. 7, no. 1, pp. 147–154, 2022.

[30] OWASP, “4.0 testing guide,” 2014. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf

[31] RedHat, “What is api security?” Jan 2019. [Online]. Available: https://www.redhat.com/en/topics/security/api-security

[32] M. Bettayeb, Q. Nasir, and M. A. Talib, “Firmware update attacks and security for iot devices: Survey,” in *Proceedings of the ArabWIC 6th Annual International Conference Research Track*, ser. ArabWIC 2019. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi-org.proxy.lnu.se/10.1145/3333165.3333169

[33] W. Mazurczyk and L. Caviglione, “Cyber reconnaissance techniques,” *Communications of the ACM*, vol. 64, no. 3, pp. 86–95, 2021.

[34] C. Holguera, S. Schleier, B. Mueller, and J. Willemsen, “Owasp mobile application security testing guide,” Sep 2022. [Online]. Available: https://github.com/OWASP/owasp-mastg/blob/master/Document/0x06c-Reverse-Engineering-and-Tampering.md

[35] OWASP, “Android tampering and reverse engineering - owasp mobile application security.” [Online]. Available: https://mas.owasp.org/MASTG/Android/0x05c-Reverse-Engineering-and-Tampering/

[36] IBM, “Choosing the best hardware for your next iot project,” 2017. [Online]. Available: https://developer.ibm.com/articles/iot-lp101-best-hardware-devices-iot-project/

[37] X. He, S. Alqahtani, R. Gamble, and M. Papa, “Securing over-the-air iot firmware updates using blockchain,” 05 2019, pp. 164–171.

[38] Kumari, “Determing unknown baud rate,” Apr 2023. [Online]. Available: https://web.archive.org/web/20230426103924/https://www.kumari.net/index.php/random/37-determing-unknown-baud-rate

[39] N. N. V. Database, “Nvd - cve-2017-17020,” 2017. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2017-17020

[40] Y. Li, Y. Yang, X. Yu, T. Yang, L. Dong, and W. Wang, “Iot-apiscanner: Detecting api unauthorized access vulnerabilities of iot platform,” in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020, pp. 1–5.

[41] A. Tierney, “Z-shave. exploiting z-wave downgrade attacks | pen test partners,” Apr 2023. [Online]. Available: https://web.archive.org/web/20230414103525/https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks/

[42] Jan 2022. [Online]. Available: https://kb.cert.org/vuls/id/142629

[43] Y. Kristiyanto and E. Ernastuti, “Analysis of deauthentication attack on ieee 802.11 connectivity based on iot technology using external penetration test,” *CommIT (Communication and Information Technology) Journal*, vol. 14, p. 45, 05 2020.

[44] Apr 2023. [Online]. Available: https://github.com/espressif/esptool

[45] OWASP, “Buffer overflow | owasp,” owasp.org, 05 2022. [Online]. Available: https://owasp.org/www-community/vulnerabilities/Buffer_Overflow

[46] Y. Lu, “Injecting software vulnerabilities with voltage glitching,” 02 2019.

[47] Feb 2020. [Online]. Available: https://cwe.mitre.org/data/definitions/1247.html

[48] N. N. V. Database, “Nvd - cve-2021-4045,” 2021. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-4045

[49] V. Fresco, “Cve-2021-4045 poc,” Mar 2023. [Online]. Available: https://github.com/hacefresko/CVE-2021-4045-PoC

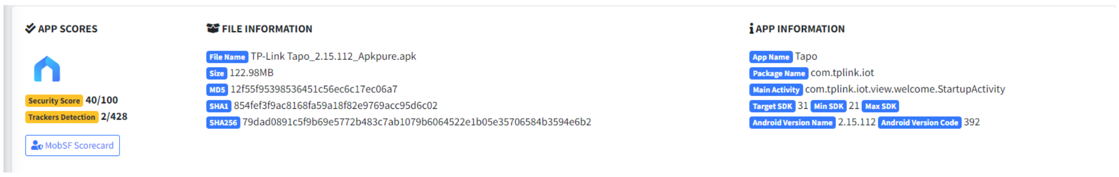
[50] LimitedResults, “Tuya smart life : How to hack and pwn this lightbulb - limitedresults,” Nov 2018. [Online]. Available: https://web.archive.org/save/https://limitedresults.com/2018/11/pwn-the-tuya-lightbulbs/

[51] Tuya, “Tylc5 module datasheet-tuya iot development platform-tuya developer,” Aug 2022. [Online]. Available: https://developer.tuya.com/en/docs/iot/wifilc5module?id=K9605t3bpxf75

[52] M. Johansson, “Internet of things security in healthcare: A test-suite and standard review,” 2018.

[53] R. Williams, E. McMahon, S. Samtani, M. Patton, and H. Chen, “Identifying vulnerabilities of consumer internet of things (iot) devices: A scalable approach,” in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2017, pp. 179–181.

A Application reverse engineering



APP SCORES

Security Score 40/100

Trackers Detection 2/428

MobSF Scorecard

FILE INFORMATION

File Name TP-Link Tapo_2.15.112_Apkpure.apk

Size 122.98MB

MD5 12f55f95398536451c56ec6c17ec06a7

SHA1 854fef3f9ac8168fa59a18f82e9769acc95d6c02

SHA256 79dad0891c5f9b69e5772b483c7ab1079b6064522e1b05e35706584b3594e6b2

APP INFORMATION

App Name Tapo

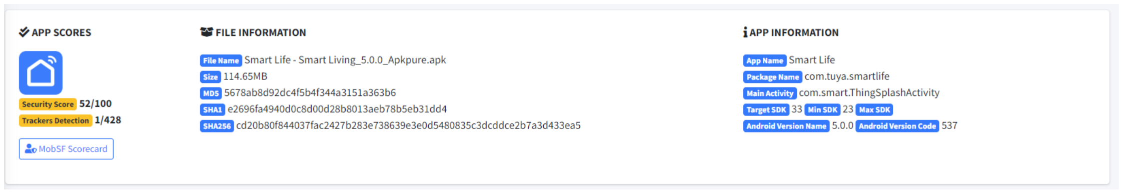
Package Name com.tplink.iot

Main Activity com.tplink.iot.view.welcome.StartupActivity

Target SDK 31 Min SDK 21 Max SDK

Android Version Name 2.15.112 Android Version Code 392

Figure 1.16: MobSF security score; IoT camera Android application



APP SCORES

Security Score 52/100

Trackers Detection 1/428

MobSF Scorecard

FILE INFORMATION

File Name Smart Life - Smart Living_5.0.0_Apkpure.apk

Size 114.65MB

MD5 5678ab8d92dc4f5b4f344a3151a363b6

SHA1 e2696fa4940d0c8d00d28b8013aeb78b5eb31dd4

SHA256 cd20b80f844037fac2427b283e738639e3e0d5480835c3dcddce2b7a3d433ea5

APP INFORMATION

App Name Smart Life

Package Name com.tuya.smartlife

Main Activity com.smart.ThingSplashActivity

Target SDK 33 Min SDK 23 Max SDK

Android Version Name 5.0.0 Android Version Code 537

Figure 1.17: MobSF security score; IoT lamp Android application

B List of abbreviations

Abbreviations	
OEM	Original equipment manufacturer
ADB	Android Debug Bridge
DDoS	Distributed Denial-Of-Service
DoS	Denial-Of-Service
MITM	Man-In-The-Middle
OSINT	Open-source intelligence
XSS	Cross site scripting
SQL	Structured Query Language
API	Application Programming Interface
RTS/CTS	Request to Send and Clear to Send
CVE	Common Vulnerabilities and Exposures
RISC	Reduced Instruction Set Computer
SaaS	Software-as-a-service
AES	Advanced Encryption Standard
SDR	Software Defined Radio
uHTTPd	micro HTTP daemon
RTSP	Real-Time Streaming Protocol
SoC	System on chip