EXAMENSARBETE INOM DATATEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, SVERIGE 2020

Ethical Hacking of an IoT-device: Threat Assessment and Penetration Testing

A Survey on Security of a Smart Refrigerator

FREDRIK RADHOLM

NIKLAS ABEFELT

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Ethical Hacking of an IoT-device: Threat Assessment and Penetration Testing

A Survey on Security of a Smart Refrigerator

FREDRIK RADHOLM

NIKLAS ABEFELT

Bachelor in Computer Science
First Cycle, 15 Credits
Supervisor: Pontus Johnson
Examiner: Robert Lagerström
Examiner: Ibrahim Orhan
School of Electrical Engineering and Computer Science
Swedish title: *Etisk Hackning av en IoT-enhet: Hotbedömning och Penetrationstestning*

Abstract

Internet of things (IoT) devices are becoming more prevalent. Due to a rapidly growing market of these appliances, improper security measures lead to an expanding range of attacks. There is a devoir of testing and securing these devices to contribute to a more sustainable society. This thesis has evaluated the security of an IoT-refrigerator by using ethical hacking, where a threat model was produced to identify vulnerabilities. Penetration tests were performed based on the threat model. The results from the penetration tests did not find any exploitable vulnerabilities. The conclusion from evaluating the security of this *Samsung* refrigerator can say the product is secure and contributes to a connected, secure, and sustainable society.

Keywords
Internet of things (IoT), device, security, penetration testing, threat assessment, vulnerabilities

Sammanfattning

Internet of Things (IoT) enheter blir mer allmänt förekommande. På grund av en snabbt expanderande marknad av dessa apparater, har bristfälliga säkerhetsåtgärder resulterat till en mängd olika attacker. Det finns ett behov att testa dessa enheter for att bidra till ett mer säkert och hållbart samhälle. Denna avhandling har utvärderat säkerheten av ett IoT-kylskåp genom att producera en hot modell för att identifiera sårbarheter. Penetrationstester har utförts på enheten, baserade på hot modellen. Resultatet av penetrationstesterna hittade inga utnyttjningsbara sårbarheter. Slutsatsen från utvärderingen av säkerheten på *Samsung*-kylskåpet är att produkten är säker och bidrar till ett uppkopplat, säkert, och hållbart samhälle.

Keywords

Internet of things (IoT), enhet, säkerhet, penetrationstester, hotbedömning, sårbarheter

Acknowledgement

We would like to thank our supervisor Pontus Johnson and examiners Robert Lagerström and Ibrahim Orhan for allowing us to conduct this thesis. The support and guidance have been excellent during this period. Our knowledge regarding computer security has been greatly increased and will benefit us in the future.

Fredrik and Niklas
June 2020

Contents

Introduction 1

1.1 Problem Definition 2

1.2 Objective 2

1.3 Delimitations 3

Background 4

2.1 Introduction to the *Internet of Things* 5

2.1.1 Definition of IoT 5

2.1.2 The Security Concern 5

2.2 Consequences of Cyber Attacks 5

2.2.1 *Mirai* Botnet DDoS Attack 5

2.2.2 The *Stuxnet Worm* Attack 6

2.2.3 Economic Impact 6

2.3 Introduction to Ethical Hacking 7

2.4 Mitigating Security Risks and Threats 7

2.4.1 NCSC Security Analysis for the UK Telecom Sector 8

2.4.2 STRIDE Threat Analysis 9

2.4.3 DREAD Rating System 9

Theoretical Background 10

3.1 Ports and Protocols 11

3.1.1 Introduction to Ports 11

3.1.2 The Internet Protocol – IP 11

3.1.3 The Hypertext Transfer Protocol – HTTP 12

3.1.4 The Security Protocols – SSL/TLS 12

3.1.5 Constrained Application Protocol – CoAP 12

3.2 Firmware 13

3.3 Threat Modelling 13

Research Methodology 15

4.1 Ethical Hacking 16

4.1.1 Method of Threat Assessment 16

4.1.2 Penetration Testing Methodology 16

4.2 Environment and Utility 18

4.2.1 Information Gathering Utilities 18

4.2.2 Tools for Man-in-the-middle 19

4.2.3 Tools for Target System Exploitation 20

4.2.4 Utilities for Decompiling Mobile Applications 20

4.2.5 Documentation and Journaling 21

System Under Consideration 22

5.1 Samsung Family Hub Side-by-side Refrigerator 23
5.1.1 *Tizen* Operating System 23
5.1.1.1 *Tizen* Security Model 24
5.1.2 Samsung SmartThings Mobile Application 24

Related Work 26

6.1 Related Work in Areas of Concern 27

Implementation 28

7.1 Planning and Reconnaissance 29
7.1.1 Information Gathering 29
7.1.1.1 Passive Information Gathering 29
7.1.1.2 Active Information Gathering 30
7.1.1.3 Acquiring Firmware 32

7.2 Scanning and Discovery 33
7.2.1 Single Port Probing and Analysis 33

7.3 Threat Assessment 36
7.3.1 Identifying Assets 37
7.3.2 Decomposing the System Under Consideration 38
7.3.3 Identifying Threats 41
7.3.3.1 Relevant Attack Vectors and Documenting Threats 43
7.3.3.2 Rating Threat Impact and Function Sensitivity 46

7.4 Attack and Gaining Access 46
7.4.1 Access User and Application Data 46
7.4.1.1 Same-origin Policy Bypass 47
7.4.1.2 Open Redirect and Address Bar Spoofing 48
7.4.1.3 Further Penetration Testing with BeEF 49
7.4.2 Abuse Open and Unsecured Ports 49
7.4.2.1 Vulnerability Scan and Brute-force Attack with *Nikto* 49
7.4.2.2 TCP SYN Flood Attack 51
7.4.2.3 CoAP Resource Discovery Tampering 52
7.4.3 Spoof External Services 54
7.4.3.1 Spoofing Against Cloud-services 54
7.4.3.2 Impersonating the Cloud-services 54
7.4.3.3 Decompiling and Analysing the Mobile Application 55

Result 57

8.1 Threat Traceability Matrix 58

Discussion 59

9.1 Result 60

9.2 Sustainability and Ethics 60

9.3 Further Research 61

Conclusion 62

Bibliography 64

Chapter I
Introduction

Overview of the project. Contains the classification of the thesis and revised topics such as the problem definition, objectives, and delimitations.

1.1 Problem Definition

Internet of things (*IoT*) devices are becoming more popular and according to the *International Data Corporation* (*IDC*) the number of these connected apparatuses will be estimated to 42 billion in 5 years[1]. An IoT device can be described as a physical object that have embedded cameras, sensors and with the purpose of transferring data from and to systems and other devices[2]. The data amount generated by IoT's will produce up to 79 Zettabytes in the year of 2025. These devices will be connected to the internet and in some cases to each other, to build a more smart and sustainable society where these products will take care of certain tasks.

Cyber-attacks are becoming more common and the need for securing these devices are becoming more vital as the attack surface is constantly growing, Symantec reported in 2017 a 600% increase in IoT attacks in 2017 [1]. Several units such as refrigerators, cars, lamps, and locks are connected, and security is becoming an increasingly important factor. The need for improving the security of these devices are vital when the use of IoT's are rapidly increasing.

1.2 Objective

The goal of this report is to evaluate the security of an IoT-device by using ethical hacking, where the product is proactively tested – with penetration testing – to contribute to a more secure and sustainable society. Actions that will be taken will be in the manner of a remote attacker. If a vulnerability in this report is found, the manufacturer will be notified about the problem before the public is informed. Security methods and attack simulations provided by this report can used as input for similar products. A threat model will be created to analyse the product for vulnerabilities and threats, which the penetration tests will be based on.

[1] https://www.idc.com/getdoc.jsp?containerId=prUS45213219
[2] https://www.oracle.com/internet-of-things/what-is-iot.html

1.3 Delimitations

This device has a large complex system with several components; thus, limitations must be made to exclude some components and attack surfaces. An early delimitation that was arranged was not to have to physically analyse the hardware in the refrigerator. Hence, having to access the chipset and associated peripherals. To access the mainboard, the refrigerator must be disassembled and there is limited knowledge regarding this matter.

The *Bluetooth* protocol will not be investigated due its limited functions, such as streaming audio, the focus will be on the more comprehensive and larger wireless communication: Wi-Fi.

No malicious applications will be produced and installed on the device – owing to the requirement of a *Tizen* and *Samsung* developer signature and certificate to distribute the application. Locally, one could develop an application via an emulator, but installing it on the refrigerator would not be supported due to the lack of developer options, which disables the functionality of connecting to the device via a *development-* or *debug bridge*[3]. According to the *license agreement* of the *Tizen Software Development Kit* (*SDK*)[4], the development kit must not be used to create malicious applications whether if the application is distributed or not.

Attempts towards acquiring firmware of the device will be conducted. However, regarding similar firmware *Tizen license agreement*[5] states: "2.2.4 You will not disassemble, decompile, reverse engineer, modify or create derivative works of the Software or documentation nor permit any third party to do so, except to the extent such restrictions are prohibited by applicable mandatory local law". Analysing the firmware will therefore be treated lightly and according to terms.

[3] A command-line tool that enables porting applications from a computer onto a device via internet connection

[4] https://developer.tizen.org/tizen-sdk-license-agreement

[5] Downloading firmware prompts the *end user license agreement* on the page https://developer.samsung.com/tizen/TizenDeviceFirmware.html

Chapter II
Background

This chapter contains definitions and an introductory background of cyber-attacks and its consequences. Also, preliminaries regarding the task at hand and its relative methodologies in areas of concern.

2.1 Introduction to the *Internet of Things*

2.1.1 Definition of IoT

The *Internet of things* (*IoT*) are devices that are connected to different systems via the internet, among other things, these products will be used privately, industrially, and commercially to facilitate and operate services.

2.1.2 The Security Concern

Implementing IoT devices comes at a cost, as the strain it employs on keeping everything secure is constantly increasing with an expanding range of products.

There is currently no regulated international legislative on cyber security regarding IoTs. There are however domestic laws and authorities that are pushing the issue on demanding a higher security threshold in IoT's[6]. Currently, the safety aspect is not a strong incentive for publishers, as it is difficult for them to be held responsible for a vulnerability [2]. The responsibility is therefore tilted more towards the consumer rather than the producer. Hence, the threat of attacks on these products relies on implementation of risk mitigation by individual consumers and organizations.

2.2 Consequences of Cyber Attacks

2.2.1 *Mirai* Botnet DDoS Attack

One of the largest cyberattacks was made in 2016 with a bot network called *Mirai*, which consisted of 400,000 connected IoT devices that carried out a variety of attacks with a great impact. The service provider *Dyn* was taken down, which resulted in hundreds of websites including Twitter, Netflix was unavailable for several hours [3]. A French web operator was attacked by this network which had a measured effect of 1.1 Terabyte per second. There were about 145,000 webcams carrying out the attack [4].

[6] https://securityboulevard.com/2020/01/new-iot-security-regulations-what-you-need-to-know-2/

Internet-connected cameras have been used to monitor houses and to observe the health of animals and humans. These were also used in the *Mirai* attack in 2016, but it is not just overload attacks that can be exploited. Unauthorized operators have taken control of these cameras and used them to monitor the owners of the device instead[7].

2.2.2 The *Stuxnet Worm* Attack

In 2010 a 500-kilobyte *computer worm* called *Stuxnet* attacked at least 14 industrial sites in Iran, where one of the targets was a uranium-enrichment plant [5]. This worm started by replicating itself on *Microsoft Windows* networks and devices. It targeted *Program industrial Control systems* (*PLC*). A *PLC* monitors input and output of analog and digital signals and can be used a supervisory control and data acquisition (*SCADA*), where it is used as a controller to interact with other machines like an IoT. When the worm finally got control of the *PLC*, it could spy on the connected industrial systems and make the centrifuges spin so fast that they tore themselves apart [5].

Thomas M. Chen and Saeed Abu-Nimeh concluded this attack: "*Stuxnet has opened security researchers' eyes to the fact that malware isn't restricted to computers. Malware can affect critical physical infrastructures, which are mostly controlled by software. This implies that threats might extend to real lives.*" (*Chen & Abu-Nimeh*, 2013) [6].

2.2.3 Economic Impact

Cyber-crime according to a report by *McAfee* and *Strategic and International Studies* (*CSIS*) in 2018 the cybercrime cost the world $600 billion. The whole internet economy is $4.2 trillion, and the cost of cybercrime could be "viewed as a 14% tax on growth" [7, 8]. Ransomware is one type of attack that is rapidly being deployed and used against corporations, where the enterprises must pay a fee for the malware to be removed, in 2016 alone $209 million was paid in the first quarter of

[7] https://www.washingtonpost.com/technology/2019/04/23/how-nest-designed-keep-intruders-out-peoples-homes-effectively-allowed-hackers-get/

2016 [7]. This type of attack can spread through the network, IoT and other devices with lacking security [7].

2.3 Introduction to Ethical Hacking

With an increase in products with varying security, the need for ethical hacking is growing. An ethical hacker performs the same type of attacks as an adversary but with the intention of contributing information and awareness about the security of a specific product [9]. An ethical hacker will not perform any unlawful actions such as breaking licensing agreements.

Ethical hacking is interchangeable with the process of vulnerability assessment and penetration testing [10]:

- Vulnerability assessment – is also known as threat assessment or security analysis, is the process of uncovering and documenting liabilities in an asset. The threat assessment can be visualized and represented by a threat model, attack-vectors and assets.
- Penetration Testing – is simulated or performed attack(s) on a target system to evaluate the overall security of the system. Identifies both weaknesses as well as strengths to enable a full risk assessment.

2.4 Mitigating Security Risks and Threats

Alleviating threats and risks involve overviewing the system and its critical functions. Dividing the system into components is a way of simplifying a complex system, where weak points inform what type of appropriate safeguards to apply.

There are various methodologies to enumerate potential threats[8], none of which is a recommended standard option, but rather preferred to specific scenarios. Threat modelling methods are used to create a – visual – abstraction of the system, profiles of methods and goals of adversaries, a catalogue of potential threats, and a subsequent ranking of threats or risks. The goal of this profiling is to correctly specify potential attacks and what elements of the system are at risk of being hit [11].

[8] https://insights.sei.cmu.edu/sei_blog/2018/12/threat-modeling-12-available-methods.html

2.4.1 NCSC Security Analysis for the UK Telecom Sector

The United Kingdom's IT agency – *National Cyber Security Centre* who is a part of *Government Communications Headquarters* – conducted a security analysis to enumerate the risks and threats faced by the UK telecoms sector in January 2020 [12]. To evaluate these risks and attacks; different models and tables were developed and assessed. *Attack trees* were used to overview exploitation routes and its outcomes – a combined visual of individual *attack vectors*. These *attack vectors* were then assigned to a specific model to overlook potential *entry points* and *attack surface*s. The functions were then assessed and ranked on different sensitivity levels to ensure security focus and mending.

The methodology of which *NCSC* operated by was to gather adversaries and their attacks to defend against threats. A visual representation can be arranged to fully recognize these risks. These are the cornerstones of the security analysis:

Attack vector is a path or strategy taken by an adversary to consummate a malicious objective on the target.

Attack/Threat trees are arrays of attack vectors used to assess cyber risks. Involves identifying higher-level impacts or outcomes and linking these to lower-level methods or exploitation routes that could contribute to such events occurring. Attack trees allows a substantial amount of possible risks and threats to be explored, creating grouping of risks into areas of concern.

Threat assessment is the threat associated with an attack vector and is evaluated and scored based on a wide range of factors: likelihood, cost, complexity, reputational impact and repeatability, additionally, the *impact* of the attack vector like damage done or data loss.

By scoring, the attack vectors require establishing the *sensitivity* of classes of the system- or device functions considering the *worst case*. *Sensitivity* is based upon:

- availability impact – denial of service (DoS) or similar attacks that deprives the functionality of the system entirely
- integrity impact – disruption caused by changing internal data of which the equipment has no control
- confidentiality impact – compromise of privacy or private data within the system.

The scoring supports the process of appropriate mitigations. The more sensitive a function is, the more critical it is for it to be protected.

2.4.2 STRIDE Threat Analysis

STRIDE stands for *Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege* and is a threat model that can be used for identifying threats for cyber systems. The first step in *STRIDE* is to analyse a system and visualize the architecture to see how data is transferred *from* and *to* the device. This approach that can be easily applied to a system to produce a threat model [13]. The acronym of *STRIDE* describes the following [14]:

- Spoofing – gain access to a system by using a forged identity. This can be achieved by taking advantage of stolen credentials or a false IP address.
- Tampering – unauthorized modification and manipulation of data.
- Repudiation – the ability to deny performing specific actions (both by authorized and unauthorized users).
- Information disclosure – exposure of sensitive data. Can be accomplished by illicit webservers or webpages and weak exception handling revealing internal system level details.
- Denial of service – make a service or application unavailable. Can be carried out by *flooding* a server or service with requests to consume all system resources.
- Elevation of privileges – gain administrative privileges as a limited user, process or account.

2.4.3 DREAD Rating System

Damage, Reliability, Exploitability, Affected user, Discoverability (*DREAD*) is a rating system that can be applied to each threat a system faces. *DREAD* is a commonly used threat evaluation that rates each individual category mentioned in the acronym – with points – from 1 to 3 [14].

Microsoft who created this scheme abandoned this method in 2010, evaluating the method to be too subjective [13].

Chapter III
Theoretical Background

This section specifies the technicalities of the device under investigation. The complexity of the system is compressed due to potential diversion from the subject in question. Remember that this is not a generalisation, rather a simplification of the components of the system. The episode also exhibits relevant entities of communication.

3.1 Ports and Protocols

3.1.1 Introduction to Ports

Ports are used for end-to-end communication between a client and a server. Operating systems today can run multiple applications concurrently and ports act as an identifier to the processes for both the server and the client on a local level. *Ephemeral ports* are transient and short-lived ports often located in the port-range of 49152 to 65535 [15]. These ports are often provoked and allocated by a client service and request.

3.1.2 The Internet Protocol – IP

The *Internet Protocol* (*IP*) suite is the primary communication protocol for transmitting packets across a network grid. This routing technique basically establishes the Internet as a whole. The packet structures contain source and destination information including *IP*- and *MAC*[9]*-addresses* and *ports* to identify hosts and their services.

The transport layer of the *IP* suite offers the datagrams of *Transmission Control Protocol* (*TCP*) and *User Datagram Protocol* (*UDP*).

TCP proposes the concept of delivering *reliable* data between hosts using the *Internet Protocol*. One of these reliabilities is the *three-way handshake*. This technique and dependency is used whenever establishing a connection between a client and a server using *TCP*: The client begins with initiating a connection by sending a *SYN* request to the server, the server responds with a *SYN-ACK*, the client acknowledges (*ACK*) the response and the data transfer can begin[10].

UDP is a protocol that on the opposite, provide an unreliable stream of data. It provides no *handshaking* or guarantees of error-checking or -correction. This can be

[9] Unique identifier of hexadecimals assigned to a specific hardware or device internet controller
[10] https://support.microsoft.com/en-in/help/172983/explanation-of-the-three-way-handshake-via-tcp-ip

useful for different purposes – such as in time-sensitive applications, where data must be processed fast and carefree.

3.1.3 The Hypertext Transfer Protocol – HTTP

The *Hypertext Transfer Protocol* (*HTTP*) is in the application layer of the *Internet protocol suite*. *HTTP* is the pinnacle and groundwork of internet exchanges for the information system known as the *World Wide Web*.

A client may request a server through a web-browser by using an *Uniform Resource Locator* (*URL*) to receive resources provided by the server – such as *Hypertext Markup Language* (*HTML*) files. The *HTML* documents can include other resources such as *JavaScript* and *Cascading Style Sheets* (*CSS*) and are rendered as well as displayed in the browser of the client. More in-depth, *JavaScript* is a scripting language that can actively alter the behaviour and content in a *HTML* file. It is a language that is capable of facilitating on how input and output is handled and parsed.

3.1.4 The Security Protocols – SSL/TLS

Secure Socket Layer (*SSL*) or *Transport Layer Security* (*TLS*) are protocols to ensure *confidentiality*, data *integrity* and *authentication* in *TCP* transmissions. Confidentiality ensures that the message is enclosed and sent to the right receiver. Data integrity checks that the message has not been altered. Authentication is certifying that the data is transferred to a proven and trusted recipient. The authentication process is done by a negotiation process called *handshaking*, where the cipher suite is decided. Cipher suites will be used to encrypt the communication. To identify a peer, a certificate can be used to verify that the *public key* sent in the negotiation is owned by that entity. The *certification authority* (*CA*) is used to validate this information [15].

3.1.5 Constrained Application Protocol – CoAP

The *Constrained Application Protocol* (*CoAP*) is used for web transfers between IoT-devices on networks. The protocol is used for *machine-to-machine* (*M2M*) communication. Servers make resources available through a *RESTful* model, where methods like *PUT*, *POST*, *DELETE* and *GET* can be applied[11]. The security of the communication is done by *Datagram Transport Layer Security* (*DTLS*), where the default parameters is proportionate to a 3072-bit *RSA* keys which is the minimum requirement to operate on a "*TOP SECRET*" (*NSA*, 2015) level.

3.2 Firmware

Firmware is what would be considered the middleware between hardware and software. The usual contents of a firmware offer instructions that provide logic for the functionalities of a device. More specifically, the software that offers low-level and fundamental control for a (IoT) device's hardware. In some cases, a higher-level software is also implemented – what is called a *Real-time Operating System* (*RTOS*).

The firmware of a device is typically stored in a *Read-only Memory* (*ROM*), meaning it is allocated *on-chip* or *on-board*, typically as *flash memory*, which can be erased and rewritten to by updates. The intentions of a firmware upgrade are to reduce or fix bugs, implement new functions or simply to improve security [16].

3.3 Threat Modelling

Threat modelling is a concept where you can illustrate functionalities of a device and its related technical components. These models dissect components individually, where security controls and countermeasures can be identified. When this process is accomplished the attack surface of the device can be evaluated with cooperation of threat identification methods such as *STRIDE* to decide wherever penetration tests can be applied. Rating the threat severity is also a part of the modelling procedure. Rating systems such as *DREAD* or *NCSC's* aids in whichever are the most critical threats and what to effectively spend most time on evaluating. Tools such as *draw.io*[12] or the *Microsoft* threat modelling tool[13] can be used to help illustrate the threat modelling process.

[11] https://coap.technology/
[12] https://drawio-app.com/analysing-vulnerabilities-with-threat-modelling-in-draw-io/
[13] https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

The threat modelling method is used to evaluate smartphones and IoT’s such as a *machine monitoring sensor* [17] and cell provider networks [18].

Chapter IV
Research Methodology

Methods used in this premise outlines vulnerabilities that could be present in an IoT-device. Furthermore, the *process* of evaluating these potential threats and security mitigations to protect intellectual property.

This chapter involves research methods and tools used, choice of environment together with resource demand and tool capability.

4.1 Ethical Hacking

4.1.1 Method of Threat Assessment

The choice between the different methods and threat categorisation mentioned in section 2.4.1 - 2.4.3 resulted in using the approach mentioned in the *IoT Penetration Testing Cookbook [14]*:

1. Identify IoT assets
2. Decompose the IoT device
3. Identify threats – by using the *STRIDE* threat categorization
4. Score the *sensitivity of functions* based on attack vector's cost, complexity, reputational impact, repeatability, and damage impact

In this methodology, the scoring of threats will be performed according to *NCSC* and not the *DREAD* rating system – as mentioned in the book. As to demonstrate in this thesis, scoring will be based on low, medium, and high scores – similar to the *DREAD* rating system.

4.1.2 Penetration Testing Methodology

Attacks and security measures are redone through the books of *Ethical Hacking and Penetration Testing Guide* and *IoT Penetration Testing Cookbook*. Steps done throughout may be excluded or diverged from, depending on relevancy and delimitations. Additionally, some more complex methods that will not match the scope of the thesis will be omitted.

A combination of two standardized methodologies in penetration testing, proves to be a fitting scenario to work towards to reach the end goal: The four-staged penetration testing methodology by *National Institute of Standards and Technology [18]* and the five-staged penetration testing method by *International Council of Electronic Commerce Consultants [19]*. If the methods are merged, they create a solid incentive for penetration testing:

Planning/Reconnaissance is to gather all essential information about the target system or device. This can be manuals, open-source code, or previous attacks etcetera. Plan worthy assets to review.

Scanning/Discovery is to actively apply tools and techniques to gather more in-depth information on the target. Identify, quantify, and prioritize potential vulnerabilities.

Attack/Gaining access is the method to attempt to exploit the vulnerabilities and gain high privileged access. If successful, rethink the discovery phase into additional steps moving forward.

Reporting is done through contacting the enterprise and report the exploit. Contemplate on risk mitigation and prevention.

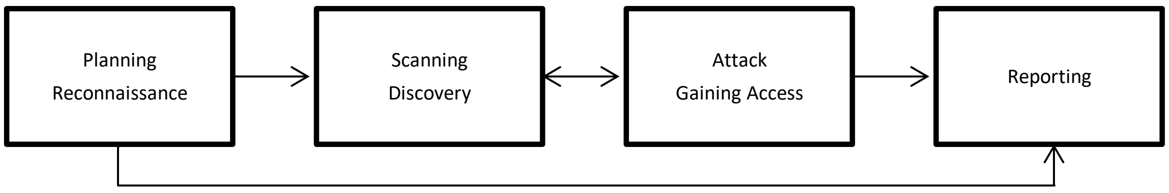

Figure 1: The four stages of the combined penetration testing methods [18] [19]

The planning and reconnaissance phase is the first step towards gathering information about the system or device. *Information gathering* has two subcategories [20]: Passive information gathering and active information gathering. Passive information gathering collect valuable intelligence about the device without the targeted system's knowledge or connection. Use search engines, instruction manuals, publicly known information-security vulnerabilities, and exposures. *Sniffing* the network is a passive approach to gather information and relevant tools such as *Wireshark* is discussed in section 4.2.2.

Active information gathering is the direct contact with systems. Detect and identify active hosts, their IP addresses, operating systems, and architecture as well as which services are run or supported. In this case, the *Linux* tool *Nmap* will be used to demonstrate this, along with several similar utilities discussed in section 4.2.1. Active reconnaissance is the more aggressive approach and is sometimes called *rattling the doorknobs*.

Vulnerabilities can be found by analysing responses by specific sent data. This reply by the target system is also known as a *fingerprint*. By obtaining and assembling enough groups of data and information; attacks can be amassed to exploit the potential vulnerabilities.

Acquiring firmware from an IoT device – according to the *IoT Penetration Cookbook* [14] – is done in four relevant approaches:

- Obtain firmware from vendor's website
- *Mirror* or Proxy network traffic when updating the device
- *Googling*/researching
- Decompiling associated mobile applications

Documentation and journaling are done for each relevant discovery and attack.

4.2 Environment and Utility

The environment setup is based on the one in the book: *IoT Penetration Testing Cookbook* [14] where it presents the operating system, *Kali Linux.*

Kali Linux is a *Debian-derived Linux* distribution and operating system that is designed and tailored for digital forensics and penetration testing. Within the platform contains a multitude of preinstalled programs such as port scanners and packet analysers. This will be the environment for gathering information and conducting tests.

4.2.1 Information Gathering Utilities

As previously mentioned in 4.1.2, *information gathering* is accomplished with the tool *Nmap*[14], which is a commonly used tool to identify a target system. It can also be applied for vulnerability analysis by using scripts that verifies unpatched exploits. The objectives for gathering data on a device are:

1. Finding open and accessible ports
2. What services the ports are running
3. If unable to identify service, try identifying the *fingerprint*

By gathering information using *Nmap*, discoveries that has been made marks the next step. This process was mentioned in section 4.1.2.

[14] https://nmap.org/

To find a vulnerability; scripts can be used. This is also referred to as a *vulnerability analysis*. Well-known exploits are tested against the system, not the attack itself, but rather to see if the protection is present.

SSLyze[15] is a tool that can be useful to identify the validity of secure connections established between client and server. It tests the connection to recognize any type of vulnerability in the cipher suite. The *SSLyze* tool is included in the *Kali Linux* environment.

4.2.2 Tools for Man-in-the-middle

Data-packets that are exchanged in a network can be eavesdropped and analysed. In the *IoT Penetration Testing Cookbook* the tool *TCPDump*[16] is used to sniff packets. The *TCPDump* tool can be *piped* to the network protocol analyser in *Wireshark*. It utilizes a more detailed view over which data is sent, by looking at the packet content i.e. bits, hexadecimal or cleartext. Information such as a packet's destination and source linked with its protocols and ports[17].

Ettercap[18] is a tool that supports active and passive examination of protocols (including encrypted) exchanged between victims. The utility presents and simplifies techniques such as *DNS Spoofing* and *ARP poisoning [21]*. This makes it possible to manipulate and taint packets. The tool is presented and demonstrated in the *IoT Penetration Testing Cookbook*.

The *Macchanger*[19] tool is implemented in the *Kali Linux* environment. This tool aids in faking a network card's hardware MAC address when trying to *spoof* against external servers.

[15] https://tools.kali.org/information-gathering/sslyze
[16] https://www.tcpdump.org/manpages/tcpdump.1.html
[17] https://www.securitynewspaper.com/2018/12/14/monitor-traffic-using-mitm-man-in-the-middle-attack/
[18] https://www.ettercap-project.org/
[19] https://linuxconfig.org/how-to-change-mac-address-using-macchanger-on-kali-linux

4.2.3 Tools for Target System Exploitation

Nikto[20] is a webserver vulnerability scanner and brute-force utility. It tests the server against missing headers, *Common Gateway Interface* (*CGI)* directories[21] and common file structures present in webservers. This tool is helpful if the device is running an accessible webserver. The *Nikto* tool is included in the *Kali Linux* environment.

Cf-browser[22] is a utility that executes through a *maven* build. This tool is specialized in sending and receiving *CoAP* requests. It supports the *RESTful API* with methods such as *GET, POST, PUT* and *DELETE*. Requests can be attached with *payloads* to specify change of a resource. This tool is presented in *Hands-on with CoAP [22]*.

Hosting an *Apache*[23] server lets a target machine to connect and communicate with its host. This is useful when conducting tests since it supports the idea of exploiting the web browser of the target machine. There is an available framework in *The Browser Exploitation Framework Project*[24] (*BeEF*) that exactly endorses this concept. An *Apache* web server can also be used as a *proxy* server to serve as a man-in-the-middle, controlling ingoing and outgoing requests between a client and a server.

4.2.4 Utilities for Decompiling Mobile Applications

In the report *Security Analysis of Android Application by Using Reverse Engineering* (2019) [23], the participants reverse engineered mobile applications by using the *apktool*[25] and *dex2jar*[26] tools. These utilities help decomposing an *Android* mobile application (*APK*) into readable *Java* classes and methods. This is beneficial to evaluate the security of the application.

[20] https://tools.kali.org/information-gathering/nikto
[21] By a *HTTP* request, a *CGI* directory can receive arguments to execute commands
[22] https://www.eclipse.org/californium/
[23] https://www.apache.org/
[24] https://beefproject.com/
[25] https://ibotpeaches.github.io/Apktool/
[26] https://github.com/pxb1988/dex2jar

4.2.5 Documentation and Journaling

Strategies, discoveries, and attacks are organized by using *KeepNote*[27]. This utility will give an overview and progress of all work done in the form of writings and screenshot images. Lastly, a threat traceability matrix will be created to assess attacks evaluated through the threat assessment and penetration testing.

[27] http://keepnote.org/

Chapter V

System Under Consideration

This episode will introduce the system under consideration: *Samsung Family Hub side-by-side* refrigerator.

5.1 Samsung Family Hub Side-by-side Refrigerator

The device this thesis will represent is the *Samsung* refrigerator RS68N8941SL. The difference between a smart refrigerator and a traditional refrigerator is a series of changes – one of many is the camera placed in the smart refrigerator's door. This camera marks the groceries in the fridge to keep track of the best before dates.

A 21.5 full HD screen is also located on the other side of the refrigerator door, where the unit's various applications are pre-installed. Applications such as *Samsung* internet browser, *gallery, calendar and more apps* can be utilized. The environment is run by the *Linux* based operating system called *Tizen*. Communication with the refrigerator can take place via *Bluetooth* and *Wi-Fi*.

5.1.1 *Tizen* Operating System

As mentioned, the refrigerator operates on a *Linux* based operating system called *Tizen*[28], this operating system is used among many different *Samsung* products including mobile phones, televisions, smart watches and cars. The security defined in the architecture takes care of certificate control, secure control distribution and asset control[29]. *SMACK* is a tool used to control what processes have access to according to predefined rules[30].

Amihai Niederman reviewed *Tizen's* operating system and said 2017 in *Kaspersky Security Analyst* at *St. Martin* that the operating system "felt like 2005"[31]. The security holes found could remotely control devices that used this operating system. 40 so-called *zero-day vulnerabilities* enabled this remote control. It was also reported that there were various vulnerabilities that could be exploited in both the *Samsung* Internet browser and *Tizen* store. Where the exploitation of the weakness of *Tizen* store could be done with the highest privileges in the system, which would allow malicious applications to be installed.

[28] https://www.tizen.org/about
[29] https://www.tizen.org/sites/default/files/tizen-architecture-linuxcollab.pdf
[30] http://schaufler-ca.com/description_from_the_linux_source_tree
[31] https://www.kaspersky.com/blog/tizen-40-bugs/14525/

5.1.1.1 *Tizen* Security Model

The *Tizen* operating system presents a *security model* of the system[32].

The application environment is based foremost on non-root privileges, hence restricting access to the filesystem and system domain. Daemons and graphics-based programs are sandboxed and run in containers. This mechanism of access control is called *Simplified Mandatory Access Control in Kernel (SMACK)*. *SMACK* is a technology that specifies three components: subject, object and access type. Subject defines domain of process, object identifies which resource(s) the process wants to address, and access type describes the constraint of procedure the subject aims to do with the object: read, write, execute etcetera. Non-root applications are unable to read or write data to other domains and files. To reduce the number of rules implemented in *SMACK*, *Cynara* was introduced. *Cynara* is a policy-checker that was developed to increase performance related to verifying rules and decreasing the load on *SMACK*.
Developing applications on *Tizen* requires an author- and distributor signature and certificate. Application privileges are contained and finalized by the store distributor, to protect its devices. All permissions of applications are stored in a manifest file locally. The present file is accessed only through system daemons.

Tizen implements the *Content Security Framework* (*CSF*) that provides an additional layer of security for web applications where it encrypts files at installation and decrypts at runtime. This is to prevent *cross-site scripting*, *clickjacking* and other injection of malicious intent.

5.1.2 Samsung SmartThings Mobile Application

The *Samsung* refrigerator compliments the usage of a mobile application also provided by *Samsung*. This non obligatory application can monitor, and control IoT-devices bundled on the same network. The application can supervise fundamentals of the refrigerator, such as temperature, defrost and groceries. The mobile application is available for *iOS* and *Android* devices and communicates over *Wi-Fi*

[32] https://wiki.tizen.org/Security/Tizen_3.X_Overview

or the mobile provider network (*4G*). This thesis will cover the *Android* version of the mobile application.

Chapter VI

Related Work

This chapter contains relevant and related work in the areas of concern.

6.1 Related Work in Areas of Concern

Passive information gathering using search engines and scholar resources has yielded relevant research regarding the domain and sphere of the thesis.

Intel Security Group member Irfan Asrar (2014) made research regarding the "*Attack Surface of the Tizen OS*" *[24]*. This report introduces the concepts of the attack surface and controls integrated into *Tizen*. These will be taken in consideration when developing and observing the threat assessment.

Ajin Abraham (2015) of *Open Security* composed a conference proceeding concerning the "*Hacking Tizen: The OS of Everything*" *[25]*. In this report an introduction to vulnerabilities and exploits in the *WebKit browser engine* are presented. Dhiraj Mishra (2017) made a same-origin policy bypass[33] of the *Samsung* internet browser versions prior to *5.4*.

Andreas Spiess (2017) hacked the *IKEA Trådfri Smart Lightning System* by using *CoAP* resources[34]. Also worthy of mention is the *DDoS* capabilities of attacks on this protocol [26].

Natalija Vlajic et al. (2019) demonstrated how cloud services could be susceptible to IP *spoofing* attacks [27].

Daniel Mayer et al. (2015) illustrated and conducted tests regarding the *Samsung SmartThings* mobile application [28]. In the report they try to impersonate and *spoof* related services of the mobile application.

May Thu Kyaw et al. (2019) exhibited how to reverse engineer an *Android* mobile application and analyse its security [23].

Robert Lagerström and Xiong Wenjun (2019) proposed using threat modelling as an resource and measure towards resolving system security [29].

[33] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-17692
[34] https://www.sensorsiot.org/145-ikea-tradfri-hack-with-gateway/

Chapter VII
Implementation

Previous disclosed methods of research were now to be applied. *Information gathering* and discoveries made via scanning contributed to a synopsis of the device and system. A threat modelling diagram would be instated to advocate risk- and threat-assessment. Threat and attack vectors were based on these models and related work, to conduct appropriate penetration testing.

Abductive reasoning may be a fallback in some cases when there is not enough adequate information presented. Resulting in which speculation is the only option.

7.1 Planning and Reconnaissance

First stage of the penetration testing method was to apply a thorough investigation on the target device. Remember that this section will describe the paths and tactics towards finding worthy assets to review.

7.1.1 Information Gathering

7.1.1.1 Passive Information Gathering

Monitoring incoming and outgoing network traffic with *Wireshark* is a way of examining the target device's communicational behaviour. This is called *mirroring* as it portrays all the internet packets sent and received by the local network. Thus, uncovering *fingerprints* and potential *entry points*.

```
# ssh root@<routers ip> tcpdump -U -w - -i br0 not port 22 | wireshark -k -i -
```

Figure 2: The command to monitor all internet traffic by the device

By first look in *Wireshark*, as the device was passive; different *DNS* requests could be seen, resolving domain names for the client.

With user interaction of the web browser there were *TCP* handshakes, more *DNS* resolves and *HTTP* requests. Also, when performing sensitive actions e.g. registering or login user, the *TLS 1.2* encryption became present, protecting data.

Examining how the device operates its communicational ports theorises in what services that were present. Of the data exchanged between client and server, *ephemerally* or dynamically allocated ports were found, as the device used the uppermost range of ports – 32768 to 61000 in *Linux kernels*. This meant that the device itself did not receive any types of request, it rather used *polling* to a *cloud service* in order to update its current state. Furthermore, this implied that the device could not be invoked by any server or peer requests as it only anticipated replies by self-initiating connections.

Even if the *Linux kernel* uniquely used an identifiable scheme of ports, it still was too early to tell what operating system this was. Since operating systems of *Microsoft* and *Apple Inc.* possesses a similar ephemeral port range of the *IANA* standard[35] – 49152 to 65535 – prediction was not yet a valid option.

Identifying the machine remotely required another set of actions. One of which via pinging the target and identifying its *Time to live* (*TTL*) since operating systems have different lifespans of this mechanism. *ICMP* requests or pings are however not a passive information gathering approach, so this is performed in section 7.1.1.2.

An unorthodox tactic can be made in order to find more background on the target system: Looking at the *MAC* address revealed the unique identifier of the vendor "*Sejongji*" which is a city in *South Korea* and hometown of *Samsung*. Also, the device was exchanging network traffic between it and a *Samsung cloud service*. If gathering these informational pieces – along with the allocation of dynamical ports, one can assume the target system of the device. In this case, *Samsung* is not associated with *Apple Inc.* products, so the remaining possible operating system turned out to be *Linux* based.

7.1.1.2 Active Information Gathering

Continuing previous segment: pinging the target device was now in scope. Upon getting a reply on a ping request, the *TTL* value received was 64, which confirmed that the operating system was indeed *Linux* based[36].

To gather more information about the system the scanning tool *Nmap* was used. Ports are communication endpoints that are assimilated to services run by the operating system which helps identifying it *if* ports are discovered.

On first approach, a regular *Nmap* scan was conducted:

```
# nmap -O TIZEN
Starting Nmap 7.80 (https://nmap.org) at 2020-01-27 04:50 EDT
Nmap scan report for TIZEN (192.168.1.XXX)
Host is up (0.067s latency).
rDNS record for 192.168.1.134: TIZEN.XXX.XXX.se
All 1000 scanned ports on TIZEN (192.168.1.134) are closed
MAC Address: XX:XX:XX:02:1C:0C (Unknown)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop
```

[35] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=133

[36] The *Linux kernel* has a *TTL*=64, *Windows* has *TTL=128*

```
OS detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 4.50 seconds
```

Figure 3: Terminal output from performing a regular *Nmap* scan

As seen in Figure 3, the scan of the most common 1000 ports for the *TCP* protocol yielded *no* results. The `-O` argument tries to determine the target operating system but the data received is not too specific to identify.

As there are 65535 ports for each IP address, the scan had to be widened to scan *every* single port on the device. This was achieved by passing more arguments to the *Nmap* command.

```
# nmap -sU -sS -T4 -v -p0-65535 TIZEN
Starting Nmap 7.80 (https://nmap.org) at 2020-01-27 05:28 EST
Initiating Ping Scan at 05:28
Scanning TIZEN (192.168.1.134) [4 ports]
Completed Ping Scan at 05:28, 0.04s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 05:28
Completed Parallel DNS resolution of 1 host. at 05:28, 0.02s elapsed
Initiating SYN Stealth Scan at 05:28
Scanning TIZEN (192.168.1.134) [65536 ports]
Discovered open port 17654/tcp on 192.168.1.134
Discovered open port 39164/tcp on 192.168.1.134
Discovered open port 51544/tcp on 192.168.1.134
Completed SYN Stealth Scan at 06:40, 6746.34s elapsed (65536 total ports)
Initiating UDP Scan at 06:40
Scanning TIZEN (192.168.1.134) [65536 ports]
Discovered open port 53/udp on 192.168.1.134
Discovered open port 33514/udp on 192.168.1.134
Completed UDP Scan at 06:42, 111.43s elapsed (65536 total ports)
Nmap scan report for TIZEN (192.168.1.134)
Host is up (0.0055s latency).
rDNS record for 192.168.1.134: TIZEN.XXX.XXX.se
Not shown: 65534 open|filtered ports, 65532 closed ports
PORT      STATE    SERVICE
0/tcp     filtered unknown
17654/tcp open     unknown
39164/tcp open     unknown
51544/tcp open     unknown
5683/udp  open     coap
5684/udp  open     coaps
Read data files from: /usr/bin/../share/nmap Nmap done: 1 IP address (1
host up) scanned in 6858.06 seconds Raw packets sent: 204483 (6.906MB) |
Rcvd: 73332 (2.933MB)
```

Figure 4: Terminal output from performing an extensive *Nmap* scan

By performing the scan in Figure 4, new discoveries were made. Although, no obvious signs of any anomalies – like an open (and unencrypted) *HTTP* port – at first sight.

Ports of TCP and UDP in the upper echelon and range – 32768 to 61000 in *Linux kernels* – indicates the use of ephemeral ports. This was detected earlier by passive information gathering with *Wireshark*, mentioned in section 7.1.1.1. These ports are short-lived and temporary as well as only valid for the duration of the communication session.

The port `17654/tcp` had to be investigated further as this was in the range of a registered or static port. No recorded activity of this port had yet to surface by using *Wireshark*, rendering the service it ran undetermined. Passive information gathering utilizing search engines, conceded *no* known services run by the port. Additional discoveries of this port had to be made to verify its usage.

The ports `5683/udp` and `5684/udp` were using the *CoAP* service. This protocol is intended to adapt to the small resources of IoT devices. It uses the *RESTful API* to multicast resources around the network. More in-depth, the protocol uses `coaps` as the secure communication line with the preferred cryptographic method of *DTLS1.2* – which is the stream-oriented version of *TLS1.2*.

Also, when hosting a webserver, the target machine enters the website resulting in an *user-agent fingerprint*:

```
Mozilla/5.0 (Linux; Tizen 4.0 Famliy Hub) AppleWebKit/537.3 (KHTML, like
Gecko) SamsungBrowser/1.0 Mobile Safari/537.3
```

Figure 5: *User-agent fingerprint* of the target machine

The *user-agent fingerprint* suggests the usage of the *Samsung* internet browser version *1.0* and the *Tizen 4.0* operating system.

7.1.1.3 Acquiring Firmware

The firmware in the refrigerator is *FHUB4N.KM.EU.RS8000.MP*. This firmware was not available for download anywhere and since the refrigerator was up to date, *mirroring* or *proxying* traffic while updating was not an option. An alternative would be to dump the firmware directly from the device, but the delimitations does not support this option, since that would require physical access to the mainboard. The last possibility would be to decompile associated mobile applications, and this was the option to go for, especially the *Samsung SmartThings* mobile application.

One might suggest using the *TM1 Tizen* firmware[37], but this firmware plugin is not at all related to the firmware existing on the refrigerator and depends on totally different hardware modules.

7.2 Scanning and Discovery

Second stage of the penetration testing method. Covered previously on this approach: *Actively apply tools and techniques to gather more in-depth information on the target. Identify, quantify, and prioritize potential vulnerabilities.*

To advance, more evidence about the system needed to be revealed. As earlier scans did not provide enough information to draw any conclusions. To acknowledge new data, *Nmap* scans on each *individual* open port were conducted.

7.2.1 Single Port Probing and Analysis

To probe the ports, meant that the scans had to be evolved even further. Addressing one port at a time. Beginning with port `17654/tcp`:

```
# nmap -v -A -p 17654 --version-intensity 9 TIZEN
...
PORT      STATE SERVICE VERSION
17654/tcp open  unknown
| fingerprint-strings:
|   FourOhFourRequest, GetRequest, HTTPOptions, RTSPRequest, SIPOptions:
|     HTTP/1.1 404 Not Found
|     Connection: close
|_    Content-Length: 0
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-
bin/submit.cgi?new-service :
SF-Port17654-TCP:V=7.80%I=7%D=5/4%Time=5EB00F52%P=x86_64-pc-linux-gnu%r(Ge
SF:tRequest,40,"HTTP/1\.1\x20404\x20Not\x20Found\r\nConnection:\x20close\r
SF:\nContent-Length:\x200\r\n\r\n")%r(HTTPOptions,40,"HTTP/1\.1\x20404\x20
SF:Not\x20Found\r\nConnection:\x20close\r\nContent-Length:\x200\r\n\r\n")%
SF:r(RTSPRequest,40,"HTTP/1\.1\x20404\x20Not\x20Found\r\nConnection:\x20cl
SF:ose\r\nContent-Length:\x200\r\n\r\n")%r(FourOhFourRequest,40,"HTTP/1\.1
SF:\x20404\x20Not\x20Found\r\nConnection:\x20close\r\nContent-Length:\x200
SF:\r\n\r\n")%r(SIPOptions,40,"HTTP/1\.1\x20404\x20Not\x20Found\r\nConnect
SF:ion:\x20close\r\nContent-Length:\x200\r\n\r\n");
MAC Address: XX:XX:XX:02:1C:0C (Unknown)
```

[37] https://developer.samsung.com/tizen/TizenDeviceFirmware.html

```
Warning: OSScan results may be unreliable because we could not find at least 1
open and 1 closed port
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=260 (Good luck!)
IP ID Sequence Generation: All zeros
Read data files from: /usr/bin/../share/nmap
OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 176.59 seconds
Raw packets sent: 24 (1.850KB) | Rcvd: 16 (1.322KB)
```

Figure 6: Partial transcript of probing port `17654/tcp`

As seen in Figure 6, there was information to process, and it was essential to parse the data to fully comprehend what potential usage this port served.

The output indicated it unrecognized the service run by the port. However, it did return data and a *fingerprint* on receiving input. Also received was the details of the operating system, which indicated the existence of a *Linux kernel*.

By further investigation, *Nmap* probed the port with *fingerprint strings* to await a reply. Upon receiving a response on a *HTTP* request, the protocol was revealed. This port was hosting a webserver, pending to receive *HTTP* requests. As the *fingerprint* stated, the requested *GET* methods only got the response of "404 Not Found". The *fingerprint* did also not give any hints towards revealing a file structure of any kind. Nevertheless, this webservice had to be explored farther.

Next was the inspection of the port `39164/tcp`. This port and `51544/tcp` were ephemeral ports under the same protocol, although, both were investigated – exhibiting only one in this section:

```
# nmap -v -sV --version-intensity 9 -p 39164 TIZEN
...
Discovered open port 39164/tcp on 192.168.1.134
rDNS record for 192.168.1.134: TIZEN.XXX.XXX.se

PORT      STATE SERVICE     VERSION
39164/tcp open  ssl/unknown
...
```

Figure 7: Output of scanning the port `39164/tcp`.

Figure 7 stated the usage of *SSL/TLS* encryption when using dynamic ports with client to server connection. To get more in-depth information, the tool *SSLyze* was used:

```
# sslyze 192.168.1.134:39164 --regular

 SCAN RESULTS FOR 192.168.1.134:39164 - 192.168.1.134
 ----------------------------------------------------

 * Downgrade Attacks:
       TLS_FALLBACK_SCSV:                 OK - Supported

 * TLSV1_1 Cipher Suites:
      Server rejected all cipher suites.

 * OpenSSL CCS Injection:
                                          OK - Not vulnerable to OpenSSL CCS injection

 * SSLV2 Cipher Suites:
      Server rejected all cipher suites.

 * SSLV3 Cipher Suites:
      Server rejected all cipher suites.

 * Session Renegotiation:
Unhandled exception while running --reneg:
SslHandshakeRejected - TLS / Alert: handshake failure

 * OpenSSL Heartbleed:
                                          OK - Not vulnerable to Heartbleed

 * TLSV1 Cipher Suites:
      Server rejected all cipher suites.

 * Deflate Compression:
                                          OK - Compression disabled

 * TLS 1.2 Session Resumption Support:
      With Session IDs:                  NOT SUPPORTED (0 successful, 5 failed, 0 errors, 5
total attempts).
      With TLS Tickets:                  NOT SUPPORTED - TLS ticket not assigned.

 * TLSV1_3 Cipher Suites:
      Server rejected all cipher suites.

 * Certificate Information:
     Content
       SHA1 Fingerprint:                  89a0c8a2236590efe9bfccde1a91d281247db59d
       Common Name:                       OCF Device: Appliance TZ (7aa75009-682e-4245-9720-
ec8cb31afe2a)
       Issuer:                            Samsung Electronics OCF HA Device SubCA v2
       Serial Number:                     412501082447657828913679542643043683923292993848
       Not Before:                        2019-06-04 04:40:08
       Not After:                         2069-12-31 14:59:59
       Signature Algorithm:               sha256
       Public Key Algorithm:              EllipticCurve
       Key Size:                          256
       Curve:                             secp256r1
       DNS Subject Alternative Names:     []

     Trust
       Hostname Validation:               FAILED - Certificate does NOT match 192.168.1.134
       Android CA Store (9.0.0_r9):       FAILED - Certificate is NOT Trusted
       Apple CA Store (iOS 13, iPadOS 13, macOS 10.15, watchOS 6, and tvOS 13):FAILED -
Certificate is NOT Trusted
       Mozilla CA Store (2019-11-28):     FAILED - Certificate is NOT Trusted
       Windows CA Store (2019-11-10):     FAILED - Certificate is NOT Trusted
       Java CA Store (jdk-13.0.2):        ERROR: SslHandshakeRejected - TLS / Alert:
handshake failure
       Symantec 2018 Deprecation:         OK - Not a Symantec-issued certificate
       Received Chain:                   OCF Device: Appliance TZ (7aa75009-682e-4245-9720-
ec8cb31afe2a) --> Samsung Electronics OCF HA Device SubCA v2
       Verified Chain:                   ERROR - Could not build verified chain
(certificate untrusted?)
       Received Chain Contains Anchor:    ERROR - Could not build verified chain
(certificate untrusted?)
       Received Chain Order:             OK - Order is valid
       Verified Chain contains SHA1:      ERROR - Could not build verified chain
(certificate untrusted?)


 * ROBOT Attack:
                                         OK - Not vulnerable, RSA cipher suites not
supported

 * TLSV1_2 Cipher Suites:
       Forward Secrecy                   OK - Supported
       RC4                               OK - Not Supported

     Preferred:
        TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256                          128 bits
     Accepted:
        TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384                          256 bits
        TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384                          256 bits
        TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256                          128 bits
        TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256                          128 bits
        ECDHE_ECDSA_WITH_AES_128_CCM_8                                   128 bits
        ECDHE_ECDSA_WITH_AES_128_CCM                                     128 bits


 SCAN COMPLETED IN 18.05 S
 -------------------------
```

Figure 8: Transcript of using *SSLyze* on the port `39164/tcp`.

It is important to notice that the *SSLyze* utility was checking the *SSL* protocol service for vulnerabilities, *weak* ciphers and server certificate validation.

The vulnerability analysis tested the *SSL*/*TLS* protocol against common exploits: *TLS* downgrade, *Heartbleed* (buffer over-read), *CCS* injection (force usage of weak private keys), *CRIME* (compression leak), insecure renegotiations and *ROBOT* (encryption liability). None of these vulnerabilities and exploits were proven to be existent on the system and its *SSL* protocol.

To summarize this section – through scans and port probing – there were discoveries made that could be built on in order to amass attacks.

7.3 Threat Assessment

Before continuing onward, a threat assessment of the system had to be considered.

7.3.1 Identifying Assets

Assets were recognized by *passive*- and *active information gathering*. Identifying assets are established to "understand where to focus probable attacks in the interest of time." (*Guzman & Gupta*, 2017).

ID	**Asset**	**Description**
1	Refrigerator	The refrigerator offers a display with the *Tizen 4.0* operating system. This system presents the usage of applications – such as an internet browser. Peripherals such as a camera can be located inside the refrigerator which has a feed to an application in the operating system. The device and system support a variety of network protocols, such as IP, TCP with TLSv1.2, UDP, DNS, DDNS, MDNS, NTP, ICMP, HTTP, HTTPS and CoAP with DTLSv1.2.
2	Camera	The camera does not provide a live-feed but rather uploads a picture to a cloud-service. This picture is available to applications on the fridge or a secondary device that has been paired with the refrigerator by authentication on the associated platform (*Samsung SmartThings*).
3	Firmware	The firmware of the refrigerator is foremostly based on the *Linux* kernel 4.9. System functions and domains are controlled and delegated by the firmware.
4	Web applications	The *Tizen* operating system offers the usage of the *Samsung* internet browser.

		A running webserver on port 17654 has been found but with an unknown service and purpose.
5	Mobile applications	*Android* and *iOS* applications are an *alternative* to controlling various settings in the refrigerator. Also, uploading and downloading media between the mobile and refrigerator. All network traffic is sent and received via the *Samsung Cloud API* (*SmartThings Cloud*) through the mobile device's network. The mobile application (*Samsung SmartThings*) connects to the cloud environment in order to authenticate against the refrigerator. A *Samsung* account (credentials) is needed to perform any actions and must be paired and matched with the one account used on the refrigerator.
6	Cloud-services	The refrigerator uses the *Samsung Cloud* and *SmartThings Cloud* to update its current state. It opens a temporary connection (ephemeral port) to the cloud service in order to receive or send a request or reply.
7	Networking	All connections to internet are done through Wi-Fi on the IEEE 802.11 protocol – except for the mobile applications, which could be interconnected to a mobile network.

Table 1: Assets of the system under consideration

The assets have been identified and the next step was to deconstruct the application into relevant categories and classes.

7.3.2 Decomposing the System Under Consideration

The refrigerator had to be investigated through decomposing of the system. By analysing the dataflow and environment of the device, locating vulnerable or exposed entry points becomes more apparent.

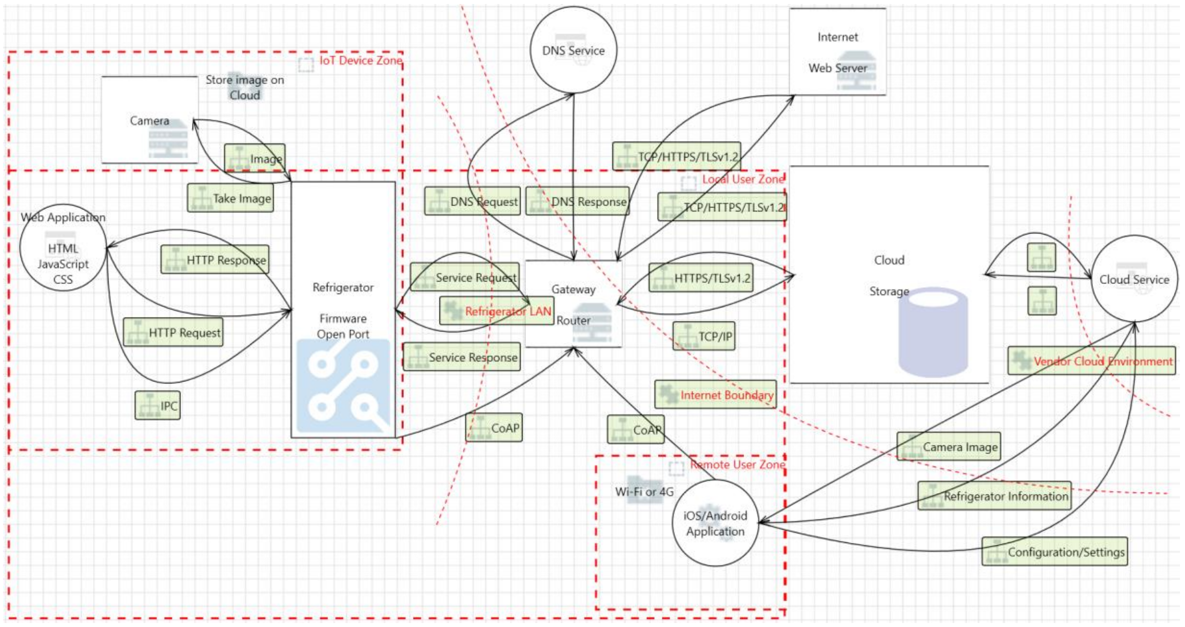

Figure 9: The decomposed system of the refrigerator

After the dataflow of the device was mapped, documentation of the *attack surface* and especially *entry points* were organized.

ID	Entry point	Description
1	Refrigerator	The refrigerator uses a web browser and compliments a mobile application. The refrigerator uses a cloud service for updates and storage. Open ports of the refrigerator have been discovered and analysed.
2	Web application	The web application present on the device and operating system uses the TLS 1.2 encryption over the TCP protocol. When performing a HTTP request, the system enforces the usage of TLS 1.2 encryption and is not allowed to downgrade encryption certificates and protocols.

		The web application supports *JavaScript* and *CSS*. Note: The web application is *Samsung* internet browser
3	Cloud-services	The refrigerator communicates with an external cloud service. Application data and communication are encrypted through TLS 1.2. The cloud service provides information to applications and changes made through the mobile application. The refrigerator sends an image of the physical storage whenever the door has opened and closed. This image is stored on the cloud and could be accessed from the mobile application. This communication craves the usage of a trusted *Samsung* certificate, presented in 7.1.1.2.
3	Firmware	The refrigerator runs on a firmware to control the device. This firmware may only be acquired through vendor technical support. Applications can utilize the firmware to manage behaviour, only if it is granted by access control.
4	Camera	The camera is operated by the firmware and provides an image when the refrigerator door has closed.
5	Mobile applications	The mobile application can monitor the refrigerator and apply configurations. The application must be paired with the refrigerator on the same network with credentials to match with a *Samsung* account. All the data is fed to the vendor cloud environment to configure and monitor the refrigerator.

7	Networking	All connections to internet are done through Wi-Fi on the IEEE 802.11 protocol – except for the mobile applications, which could be interconnected to a mobile network.

Table 2: The entry points of the refrigerator

As the *entry points* have been listed, the succeeding stage was to identify each risk they possess.

7.3.3 Identifying Threats

To detect threats, the *STRIDE* approach was used.

Threat type	**Analysis**
Spoofing	Examine the device and system for threats related to the spoofing of the refrigerator's identity. Look for ways to potentially overcome trust boundaries. Analyse the authentication and authorization process with the refrigerator's application interfaces. Evaluate the *SmartThings* application communication for the capacity to forge requests.
Tampering	Assess the refrigerator's messaging communication between applications and devices. Identify points in the refrigerator that offer an opportunity to tamper with data. Make efforts towards tampering with firmware and applications to perform unauthorized actions.
Repudiation	Identify areas where illicit access is allowed without logging.

	Disable web and mobile application tracing functionalities.
Information disclosure	Fuzz application parameters or arguments to impact application error disclosures. Identify open ports with their respective services. Incite confidentiality and integrity in the browser interface. Identify clear text communications. Review usage of *HTTP* headers and user-agent profile. Pinpoint usages of *API* endpoints and application backend technologies.
Denial of service	Identify points where data can be sent to the refrigerator. This could be open ports with no congestion control. Find ways to exhaust or *drown out* legitimate requests.
Elevation of privilege	Examine administrative capabilities of lower application users. Identify weaknesses of segregation in terms of administrative and user-level privileges.

Table 3: Using the *STRIDE* approach to assess potential threats

It is important to notice how some of the paragraphs already can be ruled out, since of the findings and evaluations in the reconnaissance and discovery phases. An example would be from the *information disclosure* heading, describing how to “identify clear text communications”, which has been demonstrated and tested through *mirroring* the network traffic of the device while performing different tasks.

7.3.3.1 Relevant Attack Vectors and Documenting Threats

Preparation for attacking the device, yields listing attack vectors and exploitation routes. Relevant attack vectors are also based on discoveries made, mention in 7.1 and 7.2. The *IoT Penetration Testing Cookbook* lists *high-level* threats based on the different components of the system. The *high-level* threats in this case that an attacker could perform – ignoring already conducted tests:

- Abuse open and unsecured ports
- Spoof external services e.g. cloud-services or mobile applications to gain or tamper with sensitive data
- Install custom and malicious firmware on the refrigerator
- Overload the refrigerator with malicious requests to prevent legitimate requests
- Access user and application data e.g. by a malicious application or *JavaScript*

Threat description	**Access user and application data**
Threat target	*Samsung* internet browser, Refrigerator user.
Attack techniques	An attacker could host an illicit webserver containing malicious *HTML* and *JavaScript* to gain credentials – a *phishing* attack.
Countermeasures	Rightful implementation of the *Content Security Framework* in the browser.
Scoring (likelihood and impact)	
Cost	Low
Complexity	Low
Reputational impact	Low
Repeatability	Medium

Damage impact	High

Table 4: Access user and application data, Threat #1

Threat description	**Abuse open and unsecured ports**
Threat target	Refrigerator network applications/services.
Attack techniques	Attacker uses a webserver vulnerability scanner and brute-forces its way to expose file structures. Attacker could flood the system with requests causing in a *Denial of Service* attack, blocking out legitimate requests. An attacker could send CoAP requests to receive or alter configurations in the device.
Countermeasures	Used services on ports are secured and idle services are closed.
Scoring (likelihood and impact)	
Cost	Medium
Complexity	Low/Medium
Reputational impact	Medium
Repeatability	Medium
Damage impact	Medium

Table 5: Abuse open and unsecured ports, Threat #2

Threat description	**Spoof external services**
Threat target	Cloud-services, *Samsung SmartThings*.
Attack techniques	Attacker sends and receives requests from a false MAC and IP address that the services trust. An attacker could disguise itself as a cloud-service in order to alter behaviour on the device. Attacker analyses the mobile application to reveal security flaws or information that could offer compromise. An attacker could dismantle the mobile application and modify it in order to uncover vulnerabilities.
Countermeasures	Packet filtering – conflicting IP addresses are blocked, cryptographic network protocols (TLS 1.2, DTLS 1.2),
Scoring (likelihood and impact)	
Cost	Medium
Complexity	Medium
Reputational impact	High
Repeatability	Medium
Damage impact	High

Table 6: Spoof external services, Threat #3

In the next section, the assessment of these functions relevant to their sensitivity is scored based on threat impact in the *worst case*, this was stated in 4.1.1.

7.3.3.2 Rating Threat Impact and Function Sensitivity

NCSC addressed "scoring the sensitivity of functions" – as mentioned earlier in 4.1.1. To readdress, the scoring of sensitivity is based on the factors of the impact and outcome. More specifically, this scoring is based on the impact relative to availability, integrity and confidentiality.

Critically sensitive	**High and moderate sensitivity**
Samsung SmartThings	*Samsung* internet browser
Cloud-services	Refrigerator network applications/services

Table 7: The sensitivity of system functions

To conclude this episode; the evaluation of the available attack surface has been applied specifically to the thesis objectives, goals, and delimitations.

7.4 Attack and Gaining Access

Initial penetration testing with the goal of exploiting and gaining access to the system via payloads. This section contains the *attack surfaces* and *entry points* combined with the theory behind them, methods and approaches used, and finally the result and discussion of findings – as stated in section 4.1.2. Subsections in this episode reveal the *attack surfaces* present on the device.

7.4.1 Access User and Application Data

The target system enters a locally hosted website which will work as a foundation for conducting tests. This webserver is intended to hold malicious scripts and actions that the web browser (*Samsung* internet browser) of the target machine might be vulnerable to.

7.4.1.1 Same-origin Policy Bypass

A policy often seen in browsers is the demand of encapsulating documents and scripts assembled from a distinct origin. This means that `http://www.example.com/` would not allow `https://www.example.com/` to be interacted with, because the protocol differs. Rightful implementation of this policy would verify the origin chain of protocols, hostnames and ports.

The victim enters a malicious website and presses a button to open a tab to a legit website, where a triggered prompt is shown awaiting username and password input. Upon providing these credentials, the victim is brought back to the malicious website where the username and password is presented, bypassing the same-origin policy.

This penetration testing requires code of *JavaScript* on a hosted website. The script function must trigger on a button click written in *HTML* or *PHP*.

First, the script is run normally on the malicious website without opening a tab to the legit website to verify the *JavaScript* functionality without trying to bypass the same-origin policy. Then again use the same approach, but with the `window.open()` method to open a new tab, hoping the malicious function still executes in the background.

As for the results, the *JavaScript* failed execution without any redirects. Also, when opening a new tab, the function seizes to execute, confirming that the *Content Security Framework* and same-origin policy were present. Different modifications to the script were made in order to correctly evaluate the presence of this policy: setting `document.domain` on the malicious website to the legitimate website, to entice the authentic origin to be consistent throughout. Though, resulting in no difference.

As the attack or bypass was unsuccessful, it confirmed that the browser was up to date. Mentioned earlier; this attack was based on a reported *CVE*, but since the *user-agent fingerprint* suggested that the browser was running version *1.0*, this had to be tested and confirmed.

The same-origin policy is important to maintain since an exploit would allow malicious websites to use scripts in other sites *Document Object Model* (*DOM*). This would permit the adversary to observe and tamper with data over multiple origins, compromising any type of confidentiality.

7.4.1.2 Open Redirect and Address Bar Spoofing

Open redirects are unvalidated redirects and forwards. A combination of a redirect is to transmit the victim to a legitimate website – without validation – with changes made to the *DOM* with the verified website still appearing in the address bar.

This penetration testing requires code of *JavaScript* on a hosted website. The script function must trigger on a non-interactive function written in *HTML* or *PHP*.

Hooking the *HTML* webpage with a `<body onpageshow="redirectFunction()">` should execute *JavaScript* when the page has loaded. This script redirects the victim to a website with changes made to the *DOM* perpetrating as a legitimate website – with the address bar still indicating the genuine domain.

Unexpectedly, when the page had loaded, it redirected the internal user without user interaction or permission. However, when changes were made to the *DOM* of the legitimate website the browser emptied the address bar as protection from spoofing.

The unauthorized redirect was successful. However, it is unsure how an adversary would continue from here onwards. The victim could simply be lured into thinking the redirect was a legitimate one and proceed to enter credentials and send it to the adversary. As a further notice, the *popup* blocker was disabled by default which seemed like an implementational issue. This allowed `window.open()` to trigger without user interaction. An update or reset of the *Samsung* internet browser mitigated this matter. It is hard to consider this as a susceptibility if it has been patched or fixed, by *abductive reasoning* this might well have been a faulty packaging or implementation of software on delivery.

Even though this was successful, it seems farfetched that this would be allowed to occur once more. No reports of this issue have been conveyed when searching the internet. Final verdict would subside with the *abductive reasoning* and therefore the success rate is deemed low.

7.4.1.3 Further Penetration Testing with BeEF

The browser was hooked with the *Browser Exploitation Framework Project* (*BeEF*). By eavesdropping, the hook successfully connected with the refrigerator, which itself contradicted the *CSP* or *CSF* rule of not letting a *script source JavaScript* tag execute. However, executing different commands from the *BeEF* interface yielded no significant results.

Efforts were made to access the camera through *ActiveX* and *Adobe flash player*. These are third-party browser plug-ins that enables media play in browsers. However, these extensions were not implemented in the device's browser, and could not be installed.

Attempt to initialize *clickjacking* was a failure since the hook failed to create an invisible *iframe* in the *DOM* as the browser recognized this.

The hook managed to forge *DNS* requests to a domain resolver, which lagged the *UI* to a minimum but nothing more.

7.4.2 Abuse Open and Unsecured Ports

The target system passively engages in using different request toward domain name servers, webservers and discovery of devices in its vicinity. This section contains subdomains of which attempts of attacks have been carried out in perspective of their services.

7.4.2.1 Vulnerability Scan and Brute-force Attack with *Nikto*

Webservers such as the one revealed in the reconnaissance phase – `17654/tcp` – sent back the replies “404 Not Found” on *HTTP* requests. As this port was not yet identified, a multitude of requests of common services was assembled through *Nikto*. This was done through sending requests against the webserver in the hopes of receiving a response not related to the “404 Not Found” reply.

The brute-force attack contained a dictionary of well-known directories and authentication paradigms present in common webservers.

This was a runnable script using 7916 requests. While the *Nikto* script was running, *Wireshark* was used to monitor requests and replies.

```
# nikto -host 192.168.1.134:17654
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.134
+ Target Hostname:    192.168.1.134
+ Target Port:        17654
+ Start Time:         2020-02-25 09:31:01 (GMT-4)
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the
user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user
agent to render the content of the site in a different fashion to the
MIME type
+ No CGI Directories found
+ 7916 requests: 0 error(s) and 3 item(s) reported on remote host
+ End Time:           2020-02-25 09:33:23 (GMT-4) (142 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figure 10: Performing a webserver vulnerability scan by brute-force

As Figure 10 suggests, 7916 requests were made against the webserver with no replies – other than 7916 replies of "404 Not Found". The script also ran through *CGI* directories which executes commands on webservers, also with no replies. It did, however, detect the absence of the headers `X-Frame-Options`, `X-Content-Type-Options` and `X-XSS-Protection`.

Since this service still was not identified, no conclusions could be conceded about the eventual impact of a successful attack. Although, it is unorthodox having access to an unfiltered open port running a webserver. This service is bound to contain a file structure. However, this port may still need specified credentials in order to have access to its service.

The attack did disclose the nonappearance of critical headers. Nevertheless, in this case it does not indicate or justify the presence of an exploit, since the usage remain undetermined. These headers are specifically optional when visiting web *pages*. Since the lack of activity by this port and its service when browsing the web, it is deemed unlikely for the lack of headers to matter.

If this attack would have been successful, *Nikto* would have revealed the directory path, containing files created or maintained by the service. The adversary would then be able to explore this structure, compromising data. With enough time it could in theory, be successfully brute-forced.

7.4.2.2 TCP SYN Flood Attack

There were open *TCP* ports listening on the device whenever it was available, these could be susceptible to *TCP SYN flood* attacks or denial-of-service attacks.

When a *TCP* flood attack is initiated. The target machine's *CPU* is overwhelmed and overworked with maintaining connectivity of each *SYN* request, resulting in that legitimate requests become delayed or even refused.

A *TCP SYN* is an initiation of a handshake, and the receiving machine wants to respond with a *TCP SYN/ACK*. If the recipient does not have any form of *congestion control* or *pipelining*, it becomes overawed with requests it *must* respond to.

As the adversary was *flooding* the target machine – using *hping3*[38], an outside source tried to ping the victim – on an external and internal network – to observe the delay the *DoS* caused. Also, trying to browse the internet using the target machine's web browser to witness potential impact.

```
# hping3 --syn --rand-source --flood -p 17654 192.168.1.134
```

Figure 11: *hping3* command

As the *flooding* began, there was a *spike* in response time:

```
ping 192.168.1.134
PING 192.168.1.134 (192.168.1.134) 56(84) bytes of data.
64 bytes from 192.168.1.134: icmp_seq=33 ttl=64 time=3.62 ms
64 bytes from 192.168.1.134: icmp_seq=34 ttl=64 time=7.55 ms
64 bytes from 192.168.1.134: icmp_seq=35 ttl=64 time=33.4 ms
64 bytes from 192.168.1.134: icmp_seq=36 ttl=64 time=56.4 ms
64 bytes from 192.168.1.134: icmp_seq=37 ttl=64 time=79.4 ms
64 bytes from 192.168.1.134: icmp_seq=38 ttl=64 time=244 ms
...
64 bytes from 192.168.1.134: icmp_seq=39 ttl=64 time=2923 ms
64 bytes from 192.168.1.134: icmp_seq=40 ttl=64 time=3806 ms
64 bytes from 192.168.1.134: icmp_seq=41 ttl=64 time=3826 ms
64 bytes from 192.168.1.134: icmp_seq=42 ttl=64 time=4353 ms
```

[38] https://linuxhint.com/hping3/

```
64 bytes from 192.168.1.134: icmp_seq=43 ttl=64 time=13639 ms
64 bytes from 192.168.1.134: icmp_seq=44 ttl=64 time=9804 ms
```

Figure 12: *ICMP* response times of target device, pinging from internal network

Trying to use network dependent services on the target device yielded slow response times and timeouts, but nothing major – e.g. a freeze or reboot.

```
ping 130.237.6.49
PING 130.237.6.49 (130.237.6.49) 56(84) bytes of data.
64 bytes from 130.237.6.49: icmp_seq=1 ttl=54 time=1.78 ms
64 bytes from 130.237.6.49: icmp_seq=2 ttl=54 time=1.92 ms
64 bytes from 130.237.6.49: icmp_seq=3 ttl=54 time=2.5 ms
64 bytes from 130.237.6.49: icmp_seq=4 ttl=54 time=3.64 ms
...
64 bytes from 130.237.6.49: icmp_seq=8 ttl=54 time=65.2 ms
64 bytes from 130.237.6.49: icmp_seq=9 ttl=54 time=43.83 ms
64 bytes from 130.237.6.49: icmp_seq=10 ttl=54 time=32.7 ms
64 bytes from 130.237.6.49: icmp_seq=11 ttl=54 time=45.32 ms
```

Figure 13: *ICMP* response times of *WAN*, pinging from external network

The target machine did not seize to respond to outside requests. However, as mentioned, legitimate requests got delayed replies. As proven in Figure 13 the network – or router – did not suffer from the *flood*, rather the target machine itself.

The *TCP SYN flood* attack was borderline successful with a minor availability impact. With this *DoS* attack the network did not congest, which would be the intention and case with most *DoS* attacks. The target machine, however, struggled with responding to every *SYN* request and at the same time replying to genuine requests. Although, while this might not be worrisome, adding congestion control mechanisms – like rate limiting or usage of a firewall – would be considered mitigating alternatives.

7.4.2.3 CoAP Resource Discovery Tampering

The *Constrained application protocol* makes devices share their resources with each other via multicast in its locale. As this protocol is built on a *RESTful API* with actions such as *GET*, *POST*, *PUT* and *DELETE*, a malicious attacker might exploit these functions to perform unauthorized changes to devices.

Theory regarding the *CoAP* service and what underlying tools that could be used to send *CoAP* requests had to be researched.

By using the utility *cf-browser* by *Californium*, forged *CoAP* messages can be sent to the *CoAP* service existent on the target machine on port `5683/udp`. In order to perform a resource discovery, the message `GET coap://192.168.1.134:5683/.well-known/core` must be sent to the target machine.

In an attempt of a response revealing its resources, a *POST* request could be assembled containing a payload in the received resource format but with alterations to corresponding values.

As the resource discovery message was sent, an *UDP* discovery response was sent back to the host machine. However, this response did not contain resources of the target machine.

The same message was sent to `5684/udp` where `coaps` was present, enforcing encrypted communication. Nonetheless, this resulted in a handshake failure as there was no change of the *cipher spec protocol*.

Remarks were made to change the options and parameters of the requests. It is possible that the request was erroneously formatted. One of many altered requests:

`PUT coap://192.168.1.134:5683/.well-known/rd?ep=node1` with the *payload*: `VALUE = </sensors/temp-1>;rt="temperature-c";if="sensor"`.

Changing the requests were to no avail and resulted as previously stated attempts.

Relevant work on this protocol depicts a *simple* approach towards apprehending device resources. This was not the case with this system. By *abductive reasoning* this contemplates the idea of keying materials and access control lists implemented by the *CoAP* protocol. As this concluded the research made on this protocol, the assumption would be that there must be extensive implementations of methods and exploitation routes toward gaining access to the resources available in the device. Additionally, the usage of *CoAPS* with *DTLS 1.2* supports this theory.

7.4.3 Spoof External Services

As the refrigerator exchanges information with cloud-services and mobile applications, *spoofing* is an alternative towards deceiving and rerouting the internet traffic to and from a falsely trusted adversary.

7.4.3.1 Spoofing Against Cloud-services

The purpose of this attack was to validate the authorization of the cloud-services. An attacker would obtain access to the internal network, perpetrating as a legitimate user – by changing its *MAC* address into a trusted existing one. Then sending a request to the cloud-service to see whether it is authorized and can begin transaction of sensitive data and send traffic to a legitimate user (fake notifications or *DoS*). To conclude, the attack tries to impersonate the refrigerator.

The attack was performed by changing the *MAC* address of the adversary's device to the one present on the refrigerator with the tool *Macchanger*. Restarting the network router and shutting down the refrigerator yielded the router to believe that the adversary's device was indeed the refrigerator – as it also received the refrigerator's IP. However, when performing a *TCP handshake* against the cloud-services it fails.

Some services only have the IP source as an authoritative and validating circumstance to identify a *legitimate* user [30]. If the attack would have been successful, the cloud-service would have accepted the falsely acquired IP and replied to the legitimate IP. If the attacker then amasses more services that operates on the same terms, the legitimate system would be bombarded with replies from a multitude of services, resulting in an *DDoS* attack.

Also, the attack could have introduced the ability to perform *replay attacks* – where the attacker simulates (*replays*) legitimate requests – like sending fake notifications. But as the attack was not able to authorize against the cloud-servers, this was not possible.

7.4.3.2 Impersonating the Cloud-services

This attack required setting up a *man-in-the-middle* (*MITM*) between the refrigerator and cloud-servers. The purpose of performing this act was to fool or spoof the refrigerator into thinking it was in contact with legitimate servers, but in fact it was an adversary monitoring and redirecting sensitive traffic.

This attack was performed with creating a local *Apache* server with a *DNS* resolver, initially forwarding or routing all request to the server from the refrigerator. The refrigerator communicated via *SSL/TLS* so the server was equipped with a falsely created certificate authority signed with the *Samsung* site (much like the one exhibited in 7.2.1). The certificates were loaded onto the *Apache* server, which was aligned and configured to accept any cipher suite. However, the refrigerator did not accept the *SSL* handshake from the server which indicated that the tests conducted on the *SSL/TLS* protocol in 7.2.1 were accurate. If the refrigerator would have accepted the *SSL* handshake, an adversary would have the ability to intercept and gather packets between the cloud-services and refrigerator performing actions such as changing temperatures and uploading images on the device.

Samsung has rightfully implemented the certificate validation process. This does make it rather *impossible* to successfully edict *man-in-the-middle* attacks. An effective attack is therefore deemed low.

7.4.3.3 Decompiling and Analysing the Mobile Application

Samsung SmartThings is a mobile application as described in 5.1.2. This application was compressed into the *Android Package Kit* (*APK*) format which stores files needed to execute the application. The *SmartThings* application was decompiled with the *apktool* and *dex2jar*.

Prerequisites to investigate the source code of the application would be knowing the *Java* and *Kotlin* (based on the *Java Class Library*) programming language.

When reverse engineering the mobile application, no direct findings where proven to be associated with any vulnerabilities. However, it was *very* easy to blunder or get lost when examining the *Java* files, since they implemented many native libraries and archives and the application itself was composed of nearly 6300 different files.

Although, one could debug the application, with using *API* calls in the *Android* environment to print variables or useful information to the *UI*. However, this was never tested due to time restraints and relevance.

If an attacker would be exposed to negligent conduct of implementations in the application source code such as a *hardcoded* encryption key, it would have been devastating if the attacker would have known how to utilize it. Exposure to other vulnerabilities such as data leakage, *API* communication and insecure data storage could be present in an insufficient mobile application. As this was not the case when conducting the research, it is considered unlikely.

Chapter VIII
Result

This chapter introduces the threat analysis along with the results from the penetration testing, presenting the *threat traceability matrix* to enrich readability and interpretation. This holistic approach ensures an effective encirclement of threat agents, affected assets, the attacks, along with their surface, goal and impact.

8.1 Threat Traceability Matrix

Presenting the attacks relative to the documented attacks presented in section 7.3.3.1.

Threat agent	Asset	Attack	Attack surface	Attack goal	Attack impact	Controls	Related work	Attempted	Success probability	Success confidence
Authorized/Unauthorized external user	User credentials, application data	SOP Bypass	*Samsung* internet browser	R/W data over origins	Confidentiality and integrity	CSF/CSP	6.1	7.4.1.1	Medium	7.4.1.1
Unauthorized external user	User credentials	Open redirect and address bar spoofing	*Samsung* internet browser	Trusted website with changed *DOM* obtains credentials	Confidentiality and integrity	CSF/CSP	6.1	7.4.1.2	Low/Medium	7.4.1.2
Unauthorized external user	Application data, system data/logs	*Nikto* webserver vulnerability scan and brute-force	Port 17654/TCP	R/W configuration files	Confidentiality and integrity	Authorization	*N/A*	7.4.2.1	Medium/High	7.4.2.1
Unauthorized external user	User credentials, application data	*Mirroring* (passive information gathering)	Cloud-services	Sniff user and application assets	Confidentiality	TLS 1.2, certificates	*N/A*	7.1.1.2	Low	*N/A*
Unauthorized external user	Application data, system data	CoAP resource discovery tampering	Port 5683/UDP	Obtain or alter configurations	Integrity	DTLS 1.2 or very limited functions	6.1	7.4.2.3	Low	7.4.2.3
Unauthorized internal/external user	*DoS*	*TCP SYN flood*	Any listening TCP port	Device seizes to respond to legitimate requests	Availability	Rate limit, firewall	*N/A*	7.4.2.2	Very high	7.4.2.2
Authorized internal user	User credentials, application data, *DoS*	Spoofing against cloud-services	Cloud-services	Send and receive sensitive data as trusted user	Confidentiality and integrity	TLS 1.2, certificates	6.1	7.4.3.1	Low	7.4.3.1
Authorized internal user	User credentials, application data, system data	Impersonating cloud-services	Cloud-services	Send and receive sensitive data as trusted user	Confidentiality and integrity	TLS 1.2, certificates	6.1	7.4.3.2	Low	7.4.3.2
Unauthorized internal/external user	Application data, system data	Decompiling and analysing the mobile application	*Samsung SmartThings Android* mobile application	Find vulnerability in source code	Integrity	*N/A*	6.1	7.4.3.3	Medium	7.4.3.3

Table 8: Threat traceability matrix presenting the results

Chapter IX
Discussion

Debating the result with a summarization of reliability, validity and generalizability of findings. The oratory of the security of the system under consideration along with answering the questions of research stated in the objectives.

Cyber security is more important today than ever. The devastating effects that companies and other sectors may be exposed to, have both economic and ethical consequences. For example, a perpetrator may deny service, access bank accounts or personal information. As the cyber world grows, the demand and need of securing data are increasing. As mentioned earlier, there will be an incredible amount of IoTs that confirm the relevance of carrying out a project that investigates such.

9.1 Result

The results from the research of threat assessment and penetration testing has produced a solid indictment for security of the system under consideration. As demonstrated in this report, mitigating measures – such as the *Content Security Framework* – minimizes exposures to e.g. phishing which is one of the most common schemes a victim can be subjected to.

This report also explored the idea of faulty implementation of minor functions – such as the *popup* blocker. Although this was patched after an update, it proves that even the most reputational vendors *do* make mistakes.

Even though the *DoS* attack did not lead to a major availability impact, it still evidences the lack of mitigating implementations – like rate limits.

External services like the cloud-servers and mobile applications proved to be secure within the research conducted.

Overall, the results prove that *Tizen* and *Samsung* emphasises security, especially confidentiality. Encryption techniques and rightful security enactments of protocols and certificates strengthens the discretions of users and their data.

9.2 Sustainability and Ethics

This thesis has used the methodology of ethical hacking which is intended to work within law ramifications and with responsible disclosure. There have been no issues surrounding the ethical aspects when conducting this thesis. Even though findings might suggest the presence of vulnerabilities, these have been in the early stages of the attack vectors, which then have run to a halt by implementations of mitigating controls. In other words, there has been no demonstrations or findings of exploiting these vulnerabilities. Referring To section 5.1.1 where several vulnerabilities was found 2017, since then *Samsung* has increased the security of *Tizen* significantly. This is the way forward to prevent exploits that could lead to cyber-attacks that effect social, economic and environmental mentioned in section 2.2.1, 2.2.2 and 2.2.3.

9.3 Further Research

Due to time limit and delimitations, not all attack surfaces were explored. This thesis had the objective of external and internal network exploitation of the system under consideration. The advice for future work would be to investigate and evaluate the system up-close. This could be e.g. the *Bluetooth* protocol and physical interaction with the mainboard – extracting potential firmware.

Tizen needs to constantly be evaluated due to the amount of reliance of applications and third-party products. There are some security concerns regarding *Tizen*:

- *Apple WebKit*
  - *Tizen* uses the *WebKit* browser engine which is known to have uncovered a lot of security related bugs[39]. This will affect the security perception of the *Tizen* operating system and might propose exploitation.
- *SMACK*
  - *SMACK* implements a lot of rules. For a developer this might become a problem since it could be challenging to rightfully implement all relative policies. An application with incorrect rules could have unexpected consequences.

As this thesis did not explore the idea of *fuzzing*, future work should invest in it. *Fuzzing* might find bugs and vulnerabilities a human would miss.

Fully evaluate the *Bluetooth* protocol to make sure it does not contain vulnerabilities.

Get hold of the firmware and analyse it, as it was not attainable in this thesis.

[39] https://www.cvedetails.com/product/10007/Apple-Webkit.html?vendor_id=49

Chapter X
Conclusion

In this thesis, the security of the IoT-device: *Samsung Family Hub side-by-side*, has been scrutinised through ethical hacking.

The objectives of the thesis were to investigate and assess an IoT through proven methodologies and practises. The research demonstrates a resourceful and capable *security model* that has been implemented into the operating system of *Tizen*. Conversely, it is impartial to state that the device *does* have a vast and complex system architecture – that is challenging to traverse through one proposition, hence the prerequisite of delimitations. The general unbiased deduction of the probability of a devastating attack is deemed minor, because of the precautionary and encapsulating measures highlighted by the developers and *Samsung*. Furthermore, an enclosed system comparable to *Tizen* with a limited set of applications and services might be the answer to a more secure and sustainable environment of *Internet of Things* – ensuring IoT's operations to occur safely in all areas of society.

Bibliography

[1] "Internet Security Threat Report," Symantec Corporation, Mountain View, 2018.

[2] D. Eidenskog and F. Kamrani, "Internet of Things En IT-säkerhetsmässig mardröm." [Online]. Available: https://www.foi.se/rest-api/report/FOI%20Memo%206172

[3] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and Other Botnets," *Computer,* vol. 50, no. 7, pp. 80-84, July 7 2017.

[4] N. Vlajic and D. Zhou, "IoT as a Land of Opportunity for DDoS Hackers," *Computer,* vol. 51, no. 7, pp. 26-34, July 31 2018.

[5] D. Kushner, "The Real Story of Stuxnet," *IEEE Spectrum,* vol. 50, no. 3, pp. 48-53, 2013.

[6] T. M. Chen and S. Abu-Nimeh, "Lessons From Stuxnet," *Computer,* vol. 44, no. 4, pp. 91-93, 2011.

[7] J. Lewis, "Economic Impact of Cybercrime - No Slowing Down," McAfee, Santa Clara, 2018.

[8] D. Dean *et al.*, "The Internet Economy In The G-20," The Boston Consulting Group, Boston, 2012.

[9] S. Patil, A. Jangra, M. Bhale, A. Raina, and P. Kulkarni, "Ethical hacking: The need for cyber security," in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, Chennai, India, 2017: IEEE.

[10] P. Engebretson, *The Basics of Hacking and Penetration Testing*. Waltham: Elsevier Inc., 2011.

[11] K. Clark, C. Lee, S. Tyree, and J. Hale, "Guiding Threat Analysis with Threat Source Models," in *2007 IEEE SMC Information Assurance and Security Workshop*, West Point, NY, USA 2007.

[12] "Security analysis for the UK telecoms sector," January 2020. Accessed: 3 March 2020. [Online]. Available: https://www.ncsc.gov.uk/files/Summary%20of%20the%20NCSCs%20security%20analysis%20for%20the%20UK%20telecoms%20sector.pdf

[13] A. Shostack, *Threat Modeling: Designing for Security*. Indianapolis: John Wiley & Sons, Inc., 2014.

[14] A. Guzman and A. Gupta, *IoT Penetration Testing Cookbook: Identify vulnerabilities and secure your smart devices*. Birmingham, UK: Packt Publishing Ltd., 2017.

[15] B. A. Forouzan, *Data Communication And Networking*. New York: The McGraw-Hill Companies, Inc., 2013.

[16] R. Savjani. (2018, July 23) Understanding Firmware Updates: The Whats, Whys, and Hows. *Embedded Software*.

[17] M. Ljungdahl and M. Nordström, "Security Analysis of MachineMonitoring Sensor Communication," Chalmers University of Technology, Gothenburg, 2016.

[18] "Recommendations of the National Institute of Standards and Technology," January 9 2008. Accessed: 26 February 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf

[19] *Penetration Testing Procedures & Methodologies*. New York: Cengage Learning, 2020.

[20] K. Graves, *Certified Ethical Hacker Study Guide*. Indianapolis, Indiana, USA: Wiley Publishing, Inc, 2010.

[21] L. Ganapathy. (2012, May 10) Ettercap Tutorial: DNS Spoofing & ARP Poisoning Examples. *The Geek Stuff*.

[22] M. Kovatsch and J. Vermillard, "Hands-on with CoAP," in *EclipseCon*, Paris, 2014.

[23] M. T. Kyaw, Y. N. Soe, and N. S. M. Kham, "Security Analysis of Android Application by Using Reverse Engineering," WCSE, Yangon, 2019.

[24] I. Asrar, "Attack Surface Analysis of the Tizen OS," Intel Security Group, Santa Clara, 2014.

[25] A. Abraham, "Hacking Tizen: The OS of Everything," in *Hack In The Box Security Conference*, Amsterdam, 2015.

[26] "Getting Ready for the Next Wave of DDoS Attacks," Hong Kong Network Operators Group, Hong Kong, 2018.

[27] N. Vlajic, M. Chowdhury, and M. Litoiu, "IP Spoofing In and Out of the Public Cloud: From Policy to Practice," MDPI, Toronto, 2019.

[28] B. Belleville *et al.*, "Matasano And ISEC Interns Summer 2014 Internet of Things Security," NCC Group, Manchester, 2015.

[29] R. Lagerström and W. Xiong, "Threat modeling – A systematic literature review," *Computers & Security,* vol. 84, pp. 53-69, 2019, doi: https://doi.org/10.1016/j.cose.2019.03.010.

[30] M. Kaufman. (2019, November) IP Range Based Authentication. *ServiceNow Elite*.

TRITA-EECS-EX-2020:476