IoTsec

Automatic Profile-based Firewall for IoT Devices

Authors:
Daniel Amkær Sørensen
Nichlas Vanggaard

Supervisors:
Jens Myrup Pedersen

Aalborg University
Networks and Distributed Systems
Master's Thesis

8th June 2017

AALBORG UNIVERSITET

10. Semester
Networks and Distributed Systems
Fredrik Bajers vej 7A
9220 Aalborg Ø
http://www.sict.aau.dk

Title:
IoTsec: Automatic Profile-based Firewall for IoT Devices

Project:
Master's Thesis

Project period:
February 2017 - June 2017

Authors:
Daniel Amkær Sørensen
Nichlas Vanggaard

Supervisor:
Jens Myrup Pedersen

Abstract:

IoT botnets have been used to take down some of the biggest services on the Internet in the fall of 2016.
This project investigates methods used to protect IoT devices behind IGDs from IoT botnets like Mirai. This is done by filtering traffic to and from the devices, using the firewall on an IGD.
The solution is based on a system that generates profiles throughout a learning phase. These profiles provide information about the traffic to and from each IoT device connected to the IGD. The learning phase is automatically started when new devices are connected to the IGD.
When the learning phase has ended, a set of firewall rules is generated for the device and loaded into the IGD's firewall. In the final system evaluation, the results show that the firewall rules, generated for the IGD, prevent botnets from spreading and participating in attacks.

No. of pages: 112
Completed: 2017-06-08

PREFACE

This project is a Master's thesis, produced by two students in the Networks and Distributed Systems program on Aalborg University.

This project explores methods to secure IoT devices, given recent massive DDoS attacks have been made by botnets, consisting of IoT devices. This indicates that the security implementation on IoT devices is insufficient, hence there is a need for improvement.

The research in this project shows that the attack vector used to infect new IoT devices has been to brute force access to a Telnet server. Until 2017-04-06, where the botnet Amnesia was discovered, which instead used an exploit to gain access to devices. The design in this project was made before Amnesia was released.

In Chapter 2, an analysis of existing botnets and the vulnerabilities that they use are made, which leads to potential solutions. Chapter 3 provides some background information, which is used as reference material in the design and implementation. Chapters 4 and 5 describe the design and implementation of the solution in this project Finally, Chapter 6 provides an evaluation of the solution.

Appendix A provides some information about the IoT devices which have been available for this project. Appendix B provides information about a vulnerable Virtual Machine made to mimic an exploitable IoT device. Both appendices can be read as extra information to the project.

Appendices C and D provide some of the raw results to the system evaluation in Chapter 6, hence they are recommended to be read when they are referred.

The source code and the results are submitted along with the project report.

Daniel Amkær Sørensen

Nichlas Vanggaard

TABLE OF CONTENTS

ABBREVIATIONS

4-tuple Tuple of {Source Address, Source Port, Destination Address, Destination Port}.

ALPN Application-layer Protocol Negotiation.

AP Access Point.

API Application Programmable Interface.

ASCII American Standard Code for Information Interchange.

AV Anti Virus.

Bash Bourne Again Shell.

C&C Control and Command.

CE Conformité Européenne.

CGN Carrier Grade NAT.

CIDR Classless Inter Domain Routing.

CUPP Common User Passwords Profiler.

DDoS Distributed Denial of Service.

DHCP Dynamic Host Configuration Protocol.

DI Dependency Injection.

DNS Domain Name System.

DoS Denial of Service.

DPI Deep Packet Inspection.

DVR Digital Video Recorder.

GPU Graphical Processing Unit.

GRE Generic Routing Encapsulation.

HOTP Hash-based One Time Password.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

HULK HTTP Unbearable Load King.

IANA Internet Assigned Numbers Authority.

ICMP Internet Control Message Protocol.

IGD Internet Gateway Device.

IoT Internet of Things.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

ISP Internet Service Provider.

JSON Javascript Object Notation.

LAN Local Area Network.

LSN Large Scale NAT.

MAC Media Access Control.

MITM Man in the Middle.

MPD Multiple Purpose Device.

NAS Network Attached Storage.

NAT Network Address Translation.

NAT-PCP NAT Port Control Protocol.

NTP Network Time Protocol.

OAuth Open Authorization Framework.

OSI Open Systems Interconnection.

PoC Proof of Concept.

RCE Remote Code Execution.

RDP Remote Desktop Protocol.

RE Regular Expression.

SNI Server Name Indication.

SPOF Single Point of Failure.

SSH Secure Shell.

SSL Secure Sockets Layer.

TCP Transmission Control Protocol.

TOR The Onion Router.

TOTP Time-based One Time Password.

UDP User Datagram Protocol.

UPnP Universal Plug and Play.

URL Uniform Resource Locator.

VNC Virtual Network Computing.

VPN Virtual Private Network.

WAN Wide Area Network.

CHAPTER

1

INTRODUCTION

On 2016-10-21, two Distributed Denial of Service (DDoS) attacks were performed against a company called Dyn. The first attack started at 11:10 UTC and ended at 13:20 UTC. The second ran from 15:50 to 17:00 UTC.

Dyn is a Domain Name System (DNS) provider which is used by large companies, including Twitter, Amazon, Tumblr, Reddit, Spotify and Netflix [1]. The attack on Dyn meant that some users were unable to access the companies which use Dyn as their DNS provider. The DDoS attack was made by a botnet called Mirai. Mirai consists of Internet of Things (IoT) devices, where most are Digital Video Recorders (DVRs) and IP cameras.

Such IoT devices become more and more popular. It is estimated that 50 billion IoT devices are going to be connected in 2020 [2]. Even though people connect the devices, they do not necessarily know if the devices are secure or care about it.

Accordingly, this project analyses botnets to find the attack vectors which are used to infect IoT devices. When the analysis is completed, the project aims to find a solution which protects against these attack vectors. Hence prevent botnets like Mirai from existing. Since a lot of devices are already infected with malware (Mirai is estimated to control 100.000 device by Dyn [3]), the solution must also protect these devices from spreading malware to new devices.

CHAPTER

2

PROBLEM ANALYSIS

IoT refers to connecting everything to the Internet. Everything from light bulbs to smart electricity meters. These devices are often small, with a low power consumption.

These devices are cheap, hence there has not been spent much money on security. The lack of security makes the devices easy targets for attackers, malware and botnets.

This chapter aims to describe why such botnets of IoT devices are a problem to the society. What an IoT device is, why people use them, and where the vulnerabilities occur in IoT devices. This is followed by an initial problem statement, which summarises the problem. Afterwards, a number of suggestions are made for the manufacturers, which can increase the security of the IoT devices.

After these solutions are presented, a problem statement specifies exactly which problem the project aims to solve. When the problem is specified, a number of proposed solutions are presented to solve the problem. Finally, a delimitation is made, to ensure that the solution can be completed within the time frame.

2.1 Why Botnets Cause Problems

The first botnet has its beginning in 1988, and was created by a student that wanted to calculate the size of the Internet [4]. Since then, botnets have evolved and are often linked with malicious intentions. This section gives a brief review of why botnets have become a problem, and to whom they are a problem.

2.1.1 Activities by Botnets

Ordinary computer botnets use a number of different “attacks”.

Botnets are often created with the purpose to be sold or hired out to criminals. The following list shows the activities that are performed using botnets, ordered by the most

valuable [5]. However, an activity can only be used for some time, whereafter the botnet can proceed to the next activity on the list. In the end, when a botnet is old, and known to security researchers, it can only be used for activities such as sending spam e-mails or DDoS attacks.

1. Identity theft
2. Information stealing
3. Reputation theft
4. Botnet hosting services
5. Pay-per-install
6. Click fraud
7. Spam e-mail
8. DDoS attack

The list is from 2011, before IoT botnets began to be a problem. If a list for IoT botnets should be created, it probably looks a bit different. For example, it might be difficult to perform identity theft from a DVR. Until now, the only attacks seen from IoT botnets are DDoS attacks, and BrickerBot which destroys compromised devices.

Aside from the "activities" on the list, some botnets are also used for mining cryptocurrency, such as bitcoin. For example, an attacker called Folio was discovered around 2014-06-18 [6]. He managed to mine 500,000,000 dogecoin (a different cryptocurrency) in just two months. At the time, they were worth $620,496. He did it by creating a botnet, which were running on Network Attached Storage (NAS) devices from Synology [6].

2.1.2 Summary

In this section, the traditional activities of botnets have been described. In the following section, IoT botnets are analysed, in order to discover their activities and how they infect devices.

2.2 Analysis of Botnets

As mentioned in Section 2.1, a botnet consists of a number of computers that are infected with the botnet malware. These computers can be used for different things. In the past, they were used to spread spam e-mails, and they are likely still used for that purpose. Some botnets are also operated with worse intent, for example performing DDoS attacks.

This section analyses four IoT botnets, namely Mirai, BASHLITE, BrickerBot and Amnesia. Afterwards, a conventional computer botnet called Sasser is analysed.

Each section is divided into three subsections. The first subsection explains how the devices are compromised by a botnet. The second subsection explains the purpose of the malware, and the kind of attacks it is able to perform. Finally, the third subsection describes how to mitigate against the attack vector that the botnet uses.

2.2.1 Mirai

Mirai is a huge botnet which consists primarily of IP cameras and DVRs. A DVR is a device typically used to save video recordings from IP cameras. The botnet was first discovered in 2016-08, and it targets Linux machines running busybox. Busybox is a collection of tools, written in a minimalistic way, which makes them perfect for small devices with limited power. The botnet propagates by brute forcing access to a Telnet service, using a list of only 62 different usernames and passwords [7].

A reason why IoT devices are attacked instead of computers, could be that IoT devices do not run Anti Virus (AV), which makes it easier to infect the devices. Furthermore, the devices are available at all time, as people do not turn them off [8].

Attack Vector

The attack is performed by scanning randomly generated IP addresses, to check if port 23 is open (well-known port of Telnet). Though some of the source code seems to be prepared to work with SSH as well [9]. If the port is open, different credentials are tried, in a brute force manner. With a list of only 62 different usernames and passwords, Mirai was able to infect as many as 500,000 devices. When Mirai was peaking in infecting new bots, it reached 500 per second, according to the author of the Mirai [10].

Use case

The botnet was created to perform DDoS attacks in order to take down different services. Originally designed by Minecraft security firms, in order to take down competitors' Minecraft servers. The purpose of this is to get more users attracted to the attacker's servers, hence get a larger revenue. The first known attack was made against Krebs on Security on 2016-09-20 [11].

Mirai has some different attacks it can execute [9]:

- UDP Generic
- UDP VSE (Valve Source Engine - attack against specific game engine)
- UDP DNS
- UDP flood
- TCP SYN

- TCP ACK
- TCP STOMP
- GRE IP
- GRE eth
- HTTP flood

Mitigation

In order to prevent a botnet as Mirai, three simple things can be done. As it only gains access through Telnet, the Telnet port should not be exposed to the Internet. And if the Telnet service must be publicly available, the credentials should first be changed – by the user – in order to create more unpredictable passwords. But in general Telnet should never be used as it is an unencrypted protocol, instead Secure Shell (SSH) should be used. The most effective strategy would be to disable the Telnet service by default, and power users that require the feature can enable it themselves.

2.2.2 BASHLITE

BASHLITE is a botnet first launched in September 2014 [12]. It peaked at approximately 1 million infected devices. It is known under many aliases, such as Lizkebab, LizardStresser, Qbot, Torlus or Gafgyt [13]. It has been used to DDoS banks, telecommunication companies and government agencies in Brazil in 2016 [14], with bandwidths up to 400 Gbps.

Attack Vector

According to multiple sources [15, 16, 17, 18], BASHLITE exploits the Bourne Again Shell (Bash) ShellShock vulnerability that can be used for Remote Code Execution (RCE) [19].

However, from looking at the leaked source code of BASHLITE, published on GitHub, there are no identifiable ShellShock exploits in the code [20].

Instead, BASHLITE uses a Telnet scanner and a small set of usernames and passwords, and identifies BusyBox based systems upon successful login. The set of credentials include 6 usernames and 14 passwords.

Use case

After installation of the botnet software on the devices, the botmaster has access to a small range of commands, where some are attacks:

- `PING` – Determine if bot is alive.
- `GETLOCALIP` – Determine the bot's IP address.
- `SCANNER ON|OFF` – Use the Telnet scanner to infect other devices.
- `HOLD` – Attack by establishing connections and holding them open.
- `JUNK` – Attack by sending random strings of junk.
- `UDP` – Attack using UDP flooding.
- `TCP` – Attack using TCP flooding.
- `KILLATTK` – Stop ongoing attack.
- `LOLNOGTFO` – Stop the botnet software on the device.

Mitigation

Since BASHLITE propagates through Telnet scanning, disabling Telnet access from the Internet is a good mitigation. If remote shell access is required, it should be configured securely with strong passwords and use SSH.

Another mitigation point against BASHLITE is to patch against known vulnerabilities, as multiple sources claim that it also propagates using the ShellShock vulnerability. ShellShock was publicly announced 2014-09-24 [21].

2.2.3 BrickerBot

BrickerBot is a new botnet that "bricks" devices after they are compromised. "Bricking" a device implies that the device is unusable afterwards, essentially turning it into a brick. This is probably done to force people – and through extension, manufacturers – to start caring about security on their IoT devices [22]. Therefore, attackers created the botnet, which only purpose is to spread itself and brick the devices afterwards.

It was first seen on 2017-03-20. The first sample was named "BrickerBot.1" by researcher Pascal Geenen. A few hours later, he also found another very similar botnet called "BrickerBot.2". "BrickerBot.2" is a bit slower than "BrickerBot.1", but more thorough in the scanning and destruction of the devices.

One month later, Pascal Geenen discovered a third, "BrickerBot.3", which distributes faster than "BrickerBot.1" and is even more thorough than "BrickerBot.2" [23, 24].

Attack Vector

The attack vector is similar to both Mirai and BASHLITE, it uses Telnet, and it looks like it uses a default set of passwords. But unlike Mirai and BASHLITE, it does not

download a binary and distribute itself. The fact that it does not distribute itself, also means that it is distributed by a set of servers. Those servers are running on the The Onion Router (TOR) network, which makes it practically impossible to track the people behind it [23, 24]. The TOR network routes traffic encrypted through multiple nodes, such that the source address is masked.

Use Case

After the botnet has access to the device, it runs a series of commands:

```
1  busybox cat /dev/urandom > /dev/sda &
2  busybox cat /dev/urandom > /dev/sdb &
3  busybox cat /dev/urandom > /dev/mtdblock1 &
4  ...
5  fdisk -C 1 -H 1 -S 1 /dev/sda
6  ...
7  route del default; iproute del defaut; ip route del default; \
8  rm -rf /* 2>/dev/null &
9  sysctl -w net.ipv4.timestamps=0;sysctl -w kernel1.threads-max=1
10 halt -n -f
11 reboot
```

Listing 2.1: Some of the commands executed by BrickerBot, dots represent more similar commands.

As it can be seen on Listing 2.1, it first tries to write random data to the disk — line 1 – 3. Afterwards, it changes the partition table using invalid parameters, or at least non optimal parameters, on line 5. On line 7, it deletes the default route, after this it does not have access to the Internet. Line 8 removes all files on the system. Line 9 changes two kernel parameters, it disables TCP timestamps and changes the maximum number of threads to one (on a normal ARM processor it is set to $\approx$ 10,000). Finally it halts (power off) the system, and if it fails, it reboots. After this reboot, it is most likely not be able to boot again, due to all the damage. It might be nearly impossible to get the device to work again.

Mitigation

As this botnet also gains access through the Telnet service, the botnet can be mitigated in the same way as Mirai and BASHLITE. Specifically by blocking access to the Telnet server.

2.2.4 Computer Botnets

In Sections 2.2.1 to 2.2.3, it is apparent that IoT botnets mainly use a brute-force approach to infecting devices. Looking at the non-embedded device world, e.g. regular computers running Windows or similar, there are also a number of botnets using the same approach.

One example is the Morto worm [25], which connects to Windows systems using Microsoft's Remote Desktop Protocol (RDP). Another example, Ragebot, uses the Virtual Network Computing (VNC) protocol to brute-force its way into computer systems [26]. These two examples are not that different from the IoT kind described earlier.

However, there are many botnets for regular computers that rely on different approaches for compromising machines. The most popular approach, in the past few years, seems to be relying on human nature, by using phishing e-mails or similar, to trick the user into executing the malware [27, 28, 29]. This is however not the same automated approach to infecting users, as it requires the user to run the malware.

Sasser

Another method that can also be considered automated is by using a vulnerability. There have not been any well-known botnets of this kind lately, but going back ten years, there have been numerous. One example is the Sasser worm, discovered in April 2004, which spread to Microsoft Windows XP and Windows 2000 computer systems that had not been patched [30]. This was a big problem then, because to patch the system, it had to be connected to the Internet, but when connected, it would be vulnerable to Sasser. According to some security experts, Sasser is estimated to infected more than a million computers [31].

Discussion

In this section, some methods of exploitation have been described. Conferring [32], and not looking at malware which utilized social engineering (persuading a user into doing what you want), the trend is that 10 – 15 years ago, malware spread through vulnerabilities, and now it spreads through brute-forcing weak credentials. This development can possibly be attributed to that there were more exploitable vulnerabilities 10 years ago than there is now, and also that brute-forcing is faster now than then, as processors are faster now.

The reason for excluding social engineering in this section is that the solution is educating users, instead of relying on software to catch it all.

In the end, vulnerabilities are certainly going to be exploited, and exploits that are more difficult to patch than others, can lead to larger and more long-lasting botnets. Therefore, given a large pool of devices that use insecure credentials, such as mentioned in Sections 2.2.1 to 2.2.3, it is not difficult to imagine why the brute-force approach has been utilized in many IoT botnets already.

Actually, after the initial botnet analysis had been written, a new botnet was discovered, which is described in Section 2.2.5.

2.2.5 Amnesia

Amnesia is a new botnet, first discovered on 2017-04-06 by Paloalto. It uses an old vulnerability that was publicly disclosed on 2016-03-22, by a security researcher named Roten Kerner [33].

This attack method is the same as used in ordinary computer botnets, as mentioned in Section 2.2.4.

Amnesia is only known to infect DVRs, but up to 227,000 DVRs run a firmware which contains the bug [33].

Attack Vector

Amnesia is probably the first known IoT botnet, which does not attack by brute forcing the Telnet service. Instead it uses the Hypertext Transfer Protocol (HTTP) server, which is exposed on the DVR. There is a bug in the HTTP server software that allows an attacker to perform RCE, by crafting a malicious HTTP request.

The DVR only has one user account, which is the root user that has unrestricted access on the system.

Through the Uniform Resource Locator (URL), it is possible to setup a reverse shell using a program called `netcat`. By establishing a reverse shell, the attacker gains direct access to operating the device as the root user, and do not have to continue using the same bug [34].

After the device is infected, the first thing the bot does is to check if the device is a honeypot running in a virtual machine like `VirtualBox`, `VMware` or `QEMU`. A honeypot is a controlled environment that security researchers use to attract malware and analyse it.

This check is done by looking at `/sys/class/dmi/id/product_name` and `/sys/class/dmi/id/sys_vendor` If one of these files contains the name of one of the previously mentioned virtual machines, Amnesia instead tries to delete everything, by executing the commands in Listing 2.2.

```
1 rm -rf / --no-preserve-root > /dev/null 2> /dev/null &
2 rm -rf ~/ > /dev/null 2> /dev/null &
3 rm -rf ./ > /dev/null 2> /dev/null &
4 rm -rf / --no-preserve-root > /dev/null 2> /dev/null
5 rm -rf ~/ > /dev/null 2> /dev/null
6 rm -rf ./ > /dev/null 2> /dev/null
```

Listing 2.2: Amnesia deleting everything.

As it can be seen in Listing 2.2, it first starts removing everything in the background and in parallel (line 1 to 3 ends with an ampersand, which launches a sub-shell in the background). Afterwards, it tries to remove everything again, but by running the remove commands sequentially (no ampersand in the end of line 4 to 6). This manoeuvre is

probably made, to make it harder for security researchers to investigate the malware by using sandboxes.

If the virtual machine check passes, Amnesia creates some files in order to ensure, the devices are still infected after a reboot. Afterwards, Amnesia connects to a Control and Command (C&C) server, and waits for further commands [33].

Use Case

Just like Mirai and BASHLITE, Amnesia can be used to perform DDoS attacks:

- HTTP flooding
- User Datagram Protocol (UDP) pulsed flooding
- UDP flooding
- HTTP Unbearable Load King (HULK) doser
- NICK → change nickname of device
- BOTKILLER → kill other bot on the system
- IP → get the bots Internet Protocol (IP)
- GET → download a file from a server

Mitigation

To avoid a DVR being infected, the only thing the user can do is to close port 80, which is the default port for HTTP traffic. However, the web server might be used to control the operation of the DVR and view the camera feeds. If access to this feature is desired, then closing the port hinders that. The port is likely opened by the Universal Plug and Play (UPnP) service described in Section 2.4.1.

Otherwise the manufacturer has fix the bug in the firmware, but even though the bug is from 2016-03-22, Paloalto have not been able to find an update that fixes the bug.

2.2.6 Summary

As stated in this section botnets are using various techniques to gain access to computers. But in IoT botnets, the trend has been to use Telnet rather than exploiting bugs in firmware. However recently, the new botnet Amnesia changed that, when the botnet exploited a bug in the firmware. Table 2.1 shows a comparison of the IoT botnets in the analysis.

In order to understand why IoT devices are such an interesting target for botnets at the moment, the next section explains what an IoT device is, and why it is useful in botnets.

Botnet name:	Devices infected	Attacks
Mirai	≈ 500,000	DDoS
BASHLITE	1,000,000	DDoS
BrickerBot	N/A	Destroys the infected device
Amnesia	N/A but up to 270,000 vulnerable devices	DDoS

Table 2.1: Comparision of IoT botnets.

2.3 Introduction to Internet of Things (IoT)

People have a tendency to connect more devices to their home network – everything from thermostats and smoke detectors to egg counters. This concept is known as IoT.

In order to understand why IoT devices are popular targets for botnets, this section explains what an IoT device is. But also what makes it an interesting target.

2.3.1 What is IoT

The concept IoT refers to connecting everything to the Internet. This has a number of advantages, as the connected items can at be monitored at all time and controlled remotely. For example, Kamstrup has created smart meters, which replaces the old analog electricity meters. Such a smart meter falls under the category of IoT. With such a meter, the consumers do not have to manually report the amount of electricity they have used. And the electricity provider knows with certainty that they receive correct measurements. Another benefit of such a smart meter is that the consumer can monitor how much power is used in the household over time [35].

As it can be seen, there are a number of advantages of connecting different devices to the Internet. Other applications could be an IoT thermostat for the radiator that can turn off the radiator when nobody are home, and of course turn it on, when somebody is coming home. Such a thermostat can be seen on Figure 2.2. It can also be used, when the house is empty for a period, e.g. while on holiday. Such that the radiator is turned off, until the day you are returning home from holiday. Doing this can potentially save a lot of money in a household.

A smoke detector can also be connected to the Internet, this gives the opportunity that it can send an e-mail or even call you and the fire department in case of fire, example seen on Figure 2.3. With an ordinary smoke detector, it only makes a sound to wake you up if you are at home. But if nobody is home, no one notices that the house is on fire [36].

Figure 2.1: Kamstrup Omnipower smart meter.

Figure 2.2: IoT thermostat.

Figure 2.3: IoT smoke detector.

Figure 2.4: IoT egg holder.

You can even get an egg counter, which informs you, when there are only a few eggs left in the fridge (Figure 2.4) [37].

As it can be seen, an IoT device can be almost everything, but typically it is a small device. The devices are typically always on and connected to the Internet.

There are different sectors within IoT, e.g. consumer products and industrial products, while industrial products often follow stricter specifications, consumer products are generally not governed by any rules. This project focuses on consumer IoT devices, since they are the primary target of the known IoT botnets.

These devices generate different kinds of traffic, some of them only communicate with one specific server, while others host different services by themself. This is elaborated in Section 3.5.

2.3.2 Summary

As described in this section, IoT devices do not have malicious intention when manufactured. Nor can any device make any harm alone, first when they are infected by a botnet do they become dangerous.

The next section explains how IoT devices can get infected, but also why they are dangerous when infected.

2.4 How IoT Became Dangerous

All devices, not only IoT devices, which are connected to the Internet, are typically connected through an Internet Gateway Device (IGD). An IGD is a device that routes traffic between the Internet Service Provider (ISP) network and the home network. The IGD typically has features such as Wi-Fi and are often delivered as an all in one device by the ISPs. A typical consumer IGD has a built-in firewall, which offers protection for the devices connected to it.

Even though the IoT devices are protected to an IGD, they can still become a threat. This section explains how the devices that are protected by a firewall are able to circumvent it.

2.4.1 UPnP

As the IGD provides a firewall, it is not possible to access a service provided behind the firewall unless there is made a rule for this service.

A standard firewall on a consumer IGD does not (or at least should not) have any predefined rules. So no IoT devices should work before such a rules is created. As it is difficult to the average user, to make rules which allow services through the firewall, a set of protocols named UPnP exist.

With some of the protocols in UPnP it is possible for a devices connected to an IGD to send a request to the IGD and make a rules which allows the service. Another protocol which is capable of making such rules is called NAT Port Control Protocol (NAT-PCP).

These protocols are created to help users setup their firewalls to allow traffic to devices behind it. But unfortunately, if the manufacturers of the IoT devices do not secure it correctly, it may introduce the risk of attackers significant. If the developers use UPnP to grant access to unused services, such as Telnet or SSH, or other services which the average user does not need. Then an attacker may gain access to the device through these services. This could be done either by making an exhaustive key search described in Section 3.1, or by determining the default credentials.

2.4.2 Default Credentials

When a network connected device is sold, it often has factory defaults settings installed. Within these settings, a default password to administer the device can sometimes be found. This is often the case with consumer IGDs, and many other network devices, such as IP cameras and DVRs.

It may be too difficult for the average user to change the password, hence it is left as the default. If the user does not change the password, it remains the default password throughout the life of the device.

Leaving a default password unchanged is equivalent to no security at all, or worse, it may provide a false sense of security for ignorant users. An ignorant user, might believe that having a default password instead of no password is secure.

But unless the default password is randomly generated, and only printed on a sticker on the device, this is wrong. Because if the same password is used on multiple devices, there is a high risk that an attacker can guess the password, or even that the password is written in the manual. Hence attackers have easy access to the devices, because they know the password, and UPnP services have setup a rule in the firewall for the device.

It might be as simple as searching for "default password + brand + model" on Google. In many cases, this yields the correct set of credentials for the device.

If the password is the same across multiple devices, the attacker could also download the firmware and extract the password, as described in Section 3.2.

2.4.3 No Restrictions

As the IoT devices are connected to the IGD as any other computer, the devices have the same access as the computer does. That means, if an attacker succeeds in gaining access to an IoT device, he can generate any kind of traffic, because there are no limits on the IGD. Therefore, the attacker can infect other devices from the IoT device, or perform other attacks on the Internet.

However it is a necessity for the IGD to allow all outgoing traffic by default, otherwise an ordinary computer would not be able to access e.g. websites.

2.4.4 Firmware Bugs

Another problem with IoT devices is that it might be difficult to upgrade the firmware. For example, the user might have to manually download the firmware to a computer and then upload the firmware to the IoT device. This is a problem, as an IP camera could be running an old Linux distribution, where there are known security vulnerabilities. If multiple IP cameras use old firmwares, which contain bugs, attackers might have easy access to the device through the bugs.

Using bugs was often used in the past for computer botnets, as described in Section 2.2.4. But also recently seen in an IoT botnet – Amnesia – as described in Section 2.2.5.

2.4.5 Summary

This section has explained four weaknesses that can lead to compromised IoT devices, and cause them to be used to perform attacks on the Internet. The UPnP protocol can be used to allow incoming traffic to an IoT device, without the owner's knowledge. In combination with weak passwords or firmware bugs, this can lead to exploited devices.

Exploited devices can then be used to perform attacks and spread botnet malware to other devices.

However, the UPnP protocol has been created for the convenience of users, for example to help them setup remote access for their devices. While it is convenient for the user, it can pose a security risk. The compromise between security and usability is discussed in the next section.

2.5 Security vs. Usability

In this section, the compromise between security and usability is discussed. The basic assumption is that mechanisms that increase security usually comes with a compromise in usability, since more effort is required from the user's side.

2.5.1 Security

A point where security can often be improved is in authentication processes. An authentication process is the confirmation of the user's credentials.

Credentials

An evident place which can provide improved security, is the use of unique credentials. It has been described in Sections 2.2.1 and 2.2.2 that the Mirai and BASHLITE botnets were able to infiltrate devices through common credentials. Therefore, by using unique credentials, the aforementioned botnets could have been prevented.

Another point to credentials is that they should be hard to guess through brute force attacks, as explained in Section 3.1. In order to increase the difficulty of finding the correct password, passwords should be long and complex.

Another feature that can be implemented, in order to improve security, is to use two-factor authentication.

Two-factor Authentication

Section 3.3.3 describes how two-factor authentication works. Basically, two steps are needed, the first is to log in with a username and password, as before. A second step could then be to type in a code, which is sent to a phone after the first step has been verified.

This requires an attacker to both find the credentials and then have access to the user's phone as well.

A third place to improve security is to ensure that the firewall, in the IGD, is protecting the devices the outside.

Firewall

By default the firewall should prevent incoming connections, such that devices are not exposed to the Internet. However, some devices might actually need to receive such connections, and therefore the firewall must be configured to allow it.

2.5.2 Usability

The three aforementioned examples should each improve security, but they can also decrease usability.

The first example of using strong unique passwords, makes it practically impossible to remember the passwords. Therefore, many users choose to use weak passwords and even reuse the passwords for multiple services, since they are more convenient for the user to remember.

In the second example about two-factor authentication, if the second step is to receive a text message with random unique code to type in. Then it requires the users to always have their phones with them, when they need access to services.

The third example, the firewall, is an example where users prefer convenience over security. To avoid the need for the user to configure firewall rules manually, the UPnP and NAT-PCP protocols were created, such that the rules are automatically generated. Some manufacturers use this tool to allow incoming traffic to all the services on their device. For example, the IP camera analysed in Section 3.2, has a working Telnet server running. If a firewall rule is automatically made, such that the Telnet server is available from the Internet, the camera would be vulnerable to both Mirai and BASHLITE.

2.5.3 Summary

As it can be seen in this section, security can be improved, but it is often (if not always) more difficult or time consuming for the user. And in many cases, convenience is chosen over security, which is taken advantage of by attackers.

This project tries to deal with the compromise between security and usability, by creating a secure solution, which does not require any user interaction.

2.6 Initial Problem Statement

As described in Section 2.2, IoT botnets are an increasing problem [38]. Mainly due to the huge amount of IoT devices that use a set of default credentials that can easily be

brute forced. In Sections 2.2.1 to 2.2.3, it was mentioned that the problem can be solved by disabling the Telnet service.

However, more recent IoT botnets have been discovered that use vulnerabilities in the firmwares of IoT devices. This shows a progression of IoT botnets, where attackers find different ways to infiltrate devices. The use of vulnerabilities can be more challenging to protect against [39].

First of all, the vulnerabilities can exist in the intended functionality of a device, and thereby it cannot be blocked by turning off a service. Instead, it requires a patch of the security issue. Depending on the manufacturer, it can take a long time to patch a vulnerability, if it even gets done. Some IoT devices may simply not contain the necessary features to receive updates.

Therefore, this project aims to find a method of protecting IoT devices from being infiltrated. And even if a device has been infiltrated, a way to protect against it being used for malicious purposes.

To protect all the IoT devices in a household, the solution is going to be implemented on the IGD. By doing this, all the devices connected to the IGD are protected.

Therefore, an analysis is made of different aspects that should be considered when designing such a security system.

2.7 Securing IoT Devices

In Section 2.4, some of the reasons why IoT devices may be insecure are expressed. This section presents methods that the manufacturer could implement, in order to make their devices more secure. The methods presented are: limiting services, alternative authentication methods, whitelisting traffic and firmware upgrades.

2.7.1 Limit Services

If an IoT device is operating as a host, as explained in Section 3.5.1. For example, an IP camera which hosts an HTTP server where the user can log in and watch the video stream from the camera.

If that is the purpose of the device, there is no reason to have a Telnet server running as well. As written in Sections 2.2.1 and 2.2.2, this is exactly what has been exploited.

This does not only apply to Telnet services, it applies to all kinds of unused services. The manufacturers should try to limit the ways to connect the device, in order to minimise the possible attack vectors.

2.7.2 Alternative Authentication

If the IoT device is hosting a service, for example an HTTP service with a login page. The manufacturer can avoid simple passwords by using alternative authentication methods. For example, two-factor authentication can be used in addition to credentials. With two-factor authentication the user has something more than the password, e.g. e-mail account. When the user tries to log in to the device, the device sends an e-mail to the account with a PIN code that must be typed in, after the username and password. Two-factor authentication is explained further in Section 3.3.3.

A different method is public-key authentication. With public-key authentication, a pair of asymmetric keys are generated, a private key for the user, and a public key which can be distributed to services where the user logs in. The service can then authenticate the user, as they have access to the private key. Asymmetric keys are explained further in Section 3.3.1.

A third method is Open Authorization Framework (OAuth), which is a centralized authorization solution. This can be used to create a secure authentication server on the Internet, which can authorize access to individual devices. The OAuth method is explained further in Section 3.3.2.

2.7.3 Whitelist Traffic

In the event that a device has been compromised, the attacker has access to do anything on the device. At least that is what the analysis has learned.

However, the manufacturer could put an internal firewall in place, which would only allow outgoing traffic to specific host names, IP addresses and ports. Thereby, creating a whitelist of traffic that is permitted.

2.7.4 Firmware Upgrade

Firmware upgrades are a very critical feature that should be supported. If there exists a bug in firmware version, there is a high risk that it will be exploited by attackers. Using such a bug for creating botnets has been done multiple times on computers, as explained in Section 2.2.4. And recently, it has been seen in IoT botnets as described in Section 2.2.5.

When a vulnerability is present in the firmware, there is nothing the user can do to protect the device, except blocking all incoming traffic on the IGD. If all incoming traffic is blocked, then the user is not be able to access the device from anywhere, but his own local network. If the device is an IP camera, and the user wants to use it view a video stream remotely, then that is prevented by the firewall

2.7.5 Summary

This section has mentioned a few ideas that the manufacturers can use to improve security in their devices. However, it seems that not all of the manufacturers take security as important as they should. For example, white-label products from China are often found to be insecure [40].

One could argue that the manufacturer have a responsibility, as they are selling a lot of devices, which have a high risk of being infected by a botnet. While some manufacturers may take responsibility and improve, a solution is needed to improve the security in general. This is what this project aims to find, as explained in the next section.

2.8 Problem Statement

From the beginning of this chapter and up to this section, several IoT botnets have been explained (in Section 2.2). Some of the security issues (in IoT devices) have been described (in Section 2.4) and an explanation for their existence has been given (in Section 2.5). A number of solutions that manufacturers can apply have been described (in Section 2.7), but a solution that can protect insecure devices is necessary.

The problem statement of this project is:

> *How can a universal solution be created that protects IoT devices in a household?*

To answer this, the following questions should be answered:

- *How can IoT botnets be stopped?*
- *How can different IoT devices be protected universally?*
- *How can such a security solution work without user input?*
- *How can information about IoT devices be shared in order to improve the solution?*

In the next section, a number of proposals are described, which can solve the above questions.

2.9 Proposals

In this section, five proposals for the a software solution are proposed. The proposals can stand alone, but in principle they can also be combined.

After each proposal, the advantages and disadvantages are listed. In the end of this section, the proposals are summarised, and a solution is chosen to be developed.

2.9.1 Manufacturers Create Profiles

This proposal requires the manufacturer to create a profile of the device they are selling. This profile should contain services which are provided by the IoT device, e.g. Telnet, nginx (web-server) or other services. Furthermore, it should contain all the IP addresses and domain names that the device contacts, and which protocols are used in this communication. This profile should then be saved in a central repository. This repository should, ideally, contain profiles for all the IoT devices.

On the IGD, a piece of software should be running, and detect when a new device is connected. When a new device is connected, the software should download the profile for the device, and reconfigure the firewall to allow for the specified traffic. Such that the device only has limited access, both out- and ingoing traffic.

Pros

- Each device only has access to exactly what it needs, because the manufacturers know which services it provides and which domain names it accesses.
- The software executed on the IGD is lightweight.

Cons

- It may be difficult to persuade manufacturers to create the profiles.
- If a device has a Telnet server, should the manufacturer give access to the server or not. If the Telnet server is opened, the device is still exposed to the threat from existing botnets. But if it is closed, the people who actually need it, cannot get access to it.

2.9.2 Community Creates Profiles

In line with the proposal in Section 2.9.1, this proposal also relies on profiles. But instead of relying on the manufacturers to create profiles, a community of enthusiasts create profiles for the devices. The software on the IGD could in principal be much like the software from the proposal in Section 2.9.1.

But in this proposal, there is a risk of multiple profiles being created for the same device, therefore it must be able to, in some way, choose between the different profiles in the repository.

Pros

- Independent of the manufacturers.
- Possibility of more profiles per device, maybe a more advanced one (e.g. Telnet/SSH access) and a basic one (only access to the basic stuff).
- Could be used along with Section 2.9.1.

Cons

- New devices will not have a profile before an enthusiast has created one, and uploaded it to the repository.

2.9.3 Auto Generate Profile

The profile should contain the same informations as in Sections 2.9.1 and 2.9.2. Therefore, this proposal might be used along with either one or both of the previous ideas.

The idea behind auto generated profiles is that the software on the IGD automatically downloads a profile from the repository if it exists. But if there is no profile available, it starts recording traffic generated on the newly connected IoT device. Based on the traffic, it generates a new profile, which only allows the traffic which has actually been detected. By doing so, the device cannot – after the profile is generated – participate in a DDoS attack, because it only allows communication with the IP addresses or domain names, which the IGD recorded in the learning phase.

Pros

- It works for brand new devices without profiles.
- Independent of manufacturers.
- Independent of community.

Cons

- There is a risk that a device without a profile, can get infected before it has a generated secure profile.
- The profile might not cover the entire traffic pattern of the device, if it doesn't access all services during the learning phase.

2.9.4 Virtual Private Network (VPN)

A different method of securing IoT devices is to deny all connections to the services hosted on the IoT devices. Such that services like Telnet, and web servers cannot be accessed from the Internet, but only from the local computers.

To access the services, a VPN connection has to be established, e.g. from a phone to the IGD. If a phone is used, an application for the VPN should be made. The functionality of this application is to setup a VPN connection and then open another application – e.g. the application used for viewing a video feed. When this is done, the video application, and only the video application, is connected to the IGD through the VPN connection.

This method can also be used together with the previous mentioned methods. E.g. if a device has some restrictions, due to the profile assigned to the device – e.g. telnet), the user can still get access to it through the VPN connection.

Pros

- Does not depend on how the devices operate.
- The IGD only has to run a VPN server.

Cons

- It might be difficult for people to understand why they need an additional application, to view the video from IP camera.
- No data restrictions (destination IP/URL and type of traffic etc.), if a device is infected.

2.9.5 Auto Scanning of New Devices

A final proposal is to perform vulnerability scanning, on all new IoT devices connected to the IGD. Such a scan can be done, by using either OpenVAS, Nessus or a similar software.

When one of these programs has scanned the IoT device, any known vulnerabilities have been discovered. or if any of the services, provided by the IoT device, uses a weak password.

After the scan, it should automatically be determined whether or not the services should be allowed access from the Internet.

Pros

- Weak passwords can sometimes be found.
- All known vulnerabilities is discovered.

Cons

- Such vulnerability scanners are very heavy to run, therefore it might be impossible to host them on the IGD.
- Vulnerability scanner may trigger security implementations in devices that can lock out the user (throttling).

2.9.6 Summary

The first three proposals rely on a profile for each IoT device. When a device is connected and recognised, the profile for the device is downloaded, and a set of firewall rules is generated for the device. To generate the profile, either the manufacturer or a community has to create it, or in the third proposal, the profile is generated by the IGD, if it does not exists in the repository.

The fourth proposal works by utilising a VPN connection, when the services has to be accessed from the Internet.

The fifth proposal only exposes the device to the Internet, if the service is approved by a scanner, e.g. Nessus or OpenVAS.

In this project, the focus is to continue with the third proposal, which is going to be implemented as a program to run on the IGD. This program identifies new devices, when they are connected to the IGD. When the IGD has enough information about the device, it searches a repository for a profile that matches the device. If the repository does not have a matching profile for the device, the IGD can generate a profile automatically.

This solution has been chosen, because it gives the opportunity to include the first and second proposal in the future. Such that, both the manufacturer and a community can also create profiles for the devices. But if the manufacturers do not want to create profiles, the program still functions. Similarly the program does not rely on a community to create the profiles.

2.10 Delimitation

Due to the time frame of the project, there are a few limitations, which are listed below.

First of all, the software in this project is only a Proof of Concept (PoC). Therefore, the software is run on a regular computer, instead of an IGD. The computer has the same

functionality as an IGD, which includes a Dynamic Host Configuration Protocol (DHCP) server, Network Address Translation (NAT) and Wi-Fi. Besides that, the repository is omitted, and the focus is on generating profiles, which match the devices, and from the profiles generate firewall rules.

Furthermore, the recognition of different protocols (HTTP, Telnet etc.) is limited to a small number of protocols.

At last, there is an assumption that the devices connected to the IGD are only be IoT devices. So there are no computers or phones, which have non-deterministic traffic patterns. The last assumption is reasonable, as it can be beneficial for security reasons to divide a home into multiple network segments (segmentation). With this segmentation, the software should only analyse and profile the IoT device subnet.

2.11 Evaluation Metrics

In order to verify the software, a number of metrics are defined. The metrics are based on the analysis in the this chapter, and are the basis for the remaining part of the report. Each metric is described in a subsection, the number of the subsection is used for reference later in the report.

2.11.1 Prevent Spreading of Botnets

As mentioned in Section 2.2, botnets spread from IoT device to IoT device. Hence the software must prevent an infected device from spreading malware to other devices on the Internet. By preventing this, the software slows down the spread of botnets, and if the software is used on all IGDs in the world, it can effectively prevent botnets from spreading.

2.11.2 Prevent Denial of Service (DoS) attacks

In addition to preventing botnets from spreading, the software must also prevent already infected IoT devices from participating in DDoS attacks as mentioned in Sections 2.2.1 to 2.2.3.

2.11.3 Operate without User Interaction

Section 2.5 states that security measures often reduces usability, therefore people may deprioritise security. Accordingly, the software must not require, or at least require a minimum of, interaction from the user.

2.11.4 No Side Effect on IoT devices

In Section 2.3.1, it can be seen that an IoT device can be almost anything, and may not be easy to configure. As follows the software must work seamlessly with the IoT devices, such that all the intended functionality is still working after the profile is applied.

2.11.5 Auto Profile Time

To minimise the risk of the IoT device being infected while the IGD is in the learning phase, the profile must be fully generated after the IoT device has been set up by the user. This is important as some botnets spread quickly, as mentioned in Section 2.2.1. Mirai was able to infect up to 500 new devices per second, when it peaked.

2.11.6 Long Time Verification

For the software to be considered as working, it must not block any legitimate traffic to or from the device, as it might break the functionality of the device. Or worse, it might prevent the device from updating its firmware (Section 2.4.4), if it cannot communicate with server hosting the firmware upgrades. Therefore, there must be no dropped packages by the firewall for 24 hours after the device has been profiled.

2.11.7 Summary

A summary of the metrics is listed below.

2.11.1 Prevent botnets from spreading.
2.11.2 Prevent an infected IoT device from participating in DDoS attacks.
2.11.3 Require no interaction from users.
2.11.4 Work seamlessly with the IoT devices.
2.11.5 Profile must be generated, after an IoT device is connected and configured.
2.11.6 After the firewall rules are generated and applied, there must be no (non-malicious) dropped packets to or from the devices within 24 hours.

This section marks the end of the problem analysis chapter. In the next chapter, a technical analysis is presented, which describes the technologies that are used in the design chapter.

CHAPTER

3

TECHNICAL ANALYSIS

This chapter contains reference material which is used in the design and implementation chapters, along with more technical descriptions of topics referenced from the problem analysis.

3.1 Exhaustive Key Search

This section provides insight into the problem mentioned in Section 2.4.2, where it is described that an attacker can gain unauthorized access by guessing the password that is used [41, p. 23–30]. This attack also commonly known as a brute force attack.

In this section, there is differentiated betweens keys, which are fixed length bit sequences of ones and zeros, and passwords, which are variable length sequences of limited character sets.

3.1.1 Types of Brute Force Attacks

There are two types of brute force attacks: online and offline. The difference between the two is whether the attacker can verify the result locally (offline), or if it has to be done remotely (online). In order to know whether the correct key is found, the result must be verified.

Password Cracking

In an offline attack this is done by comparing the result with a known result. Listing 3.2 shows the hash of an input string, which could be a password. The output hash can then be compared with the known hash.

In online attacks, it is done by asking a remote service to verify the result. Listing 3.1 shows the attempt to log in using the password "12345". This means that offline attacks are often much faster, as there is no network communication necessary.

```
[root@kali ~] time sshpass -v -p 12345 ssh admin@203.0.113.23
SSHPASS searching for password prompt using match "assword"
SSHPASS read: admin@203.0.113.23's password:
SSHPASS detected prompt. Sending password.
SSHPASS read:

Permission denied, please try again.
SSHPASS read: admin@203.0.113.23's password:
SSHPASS detected prompt, again. Wrong password. Terminating.

real    0m2.300s
user    0m0.006s
sys     0m0.012s
```

Listing 3.1: Online verification of guess. The program `sshpass` is used to send the password, and `time` is used to measure how long it took.

```
[root@kali ~] time echo 12345 | openssl sha256 -r
f33ae3bc9a22cd7564990a7947899544099770139 66fb1a8f43c35776b833a95 *stdin

real    0m0.004s
user    0m0.004s
sys     0m0.000s
```

Listing 3.2: Offline verification of guess. The program `openssl` is used to hash the input `12345`.

There are some technical challenges involved in both types of attack. For example, in order to perform an offline attack, the attacker must acquire data that can be used to verify whether a guess is correct. To brute force a password, this typically means acquiring a hash of the password. In simple terms, a hash is an irreversible computation of an input that always yields a fixed length output.

Encryption Key Cracking

To brute force an encryption key, such as one used in SSL, the attacker needs to know what the plaintext corresponding to the ciphertext looks like. An exact plaintext may not be necessary, depending on the protocol that is encrypted. In certain cases it can be sufficient to determine that all the characters are printable (ASCII codes between 32 and 127), e.g. if the protocol is text based, such as HTTP.

Online Attack Prevention

In online attacks, other measures can be taken to prevent brute force attacks, e.g. throttling, by limiting the number of attempts that can be made in a time span per IP or account. However, with throttling there is a possibility that an attacker can instead perform denial of service, by locking out login attempts to a certain account or from a certain IP. This is one of the risks when using throttling.

On Linux, the package fail2ban (https://www.fail2ban.org) is an example of a way to throttle login attempts. This package works by analyzing log files for login attempts, and then blocking the source IP address from connecting after multiple wrong attempts.

3.1.2 Entropy

The difficulty of guessing the correct key (or password) depends on the entropy of the key. Entropy is a measure for the number of different possibilities that can be the correct solution.

For a key, which is a fixed length bit sequence, the entropy is equal to the amount of bits in the sequence. However, this is only correct if the sequence is randomly generated using a cryptographically strong random number generator.

For a password that is randomly generated (using a good random number generator), the entropy can be calculated using Equation (3.1). However, passwords that are human generated, or follows any rules quickly diminishes the amount of entropy, as the number of symbols at each position is reduced. This is especially true, if the password is generated using natural language words.

$$H = L \cdot \log_2 N \tag{3.1}$$

Where H is the entropy in bits, and L is the password length, and N is the number of symbols.

3.1.3 Special Cases with Passwords

In addition to using natural language words in a password, there are other mistakes that reduce the difficulty of brute forcing passwords. One example is using a password that is included in a known password lists, such as `rockyou.txt`, which contains 14,344,391 passwords [42].

Another issue is using personal information in a password, e.g. birth date, names, pets' names and so on. Attackers can use tools such as Common User Passwords Profiler (CUPP) that can generate a password list based on many combinations of known information about a user [43].

However, this is not something that can be used in automated botnets such as Mirai, as it requires personal information, which can be difficult to gather automatically, without already being on the inside.

3.1.4 Summary

This section has described a weakness that can be exploited by attackers, which exist in many devices and configurations, due to negligence from manufacturers and users. In

the next section, a means of utilising brute force attacks to retrieve the default password of a device is described, by extracting a firmware image of a device. Brute force attacks are then used to crack the hash used in the image.

In Section 3.3, different authentication methods are described, which can be used in place of passwords, or together with, to increase security.

3.2 Firmware Disassembly

This section describes a method for gaining insight into the software stack that is running on IoT devices. This method, however, only relates to devices that are running a Linux operating system or similar. Devices that run embedded applications require different approaches.

In this section, the firmware image of an IP camera is used and reverse engineered. The specific firmware image is of a NEXUS FW series camera. The firmware image is downloaded from:

`http://www.cinema-shop.dk/nexus-235fw.html`.

The downloaded file is a RAR file, this file is extracted. The RAR file contains a ROM file.

The ROM file is a binary file which contains the entire firmware for the IP camera. This ROM is scanned using a Linux program called `binwalk`. The program `binwalk` can scan binary files, looking for known file signatures, and then extract files within the binary file.

The `binwalk` scan discovers that the ROM contains a JFFS2 image. This image is extracted and mounted, it contains a complete Linux filesystem with everything included, see Listing 3.3.

```
drwxr-xr-x  2 1000 1000    0 Jul  8  2015 bin
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 boot
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 dev
drwxrwxrwx  6    0    0    0 Aug  8  2014 etc
drwxrwxrwx  2    0    0    0 Sep 29  2011 font
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 home
lrwxrwxrwx  1 1000 1000    9 Jul  8  2015 init -> sbin/init
drwxr-xr-x  3    0    0    0 Dec 15  2014 jb_config
lrwxrwxrwx  1    0    0   10 Jul  8  2015 komod -> /tmp/komod
drwxrwxrwx  2    0    0    0 Jul  8  2015 lib
lrwxrwxrwx  1 1000 1000   11 Jul  8  2015 linuxrc -> bin/busybox
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 lost+found
-rw-r--r--  1 1000 1000 1341 Apr 21  2011 mkimg.rootfs
-rw-r--r--  1 1000 1000  431 Apr 21  2011 mknod_console
drwxr-xr-x 11 1000 1000    0 Dec 15  2014 mnt
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 nfsroot
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 opt
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 proc
drwxr-xr-x  2 1000 1000    0 Jan 22  2015 root
drwxrwxrwx  2    0    0    0 Jul  8  2015 sbin
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 share
-rwxr--r--  1    0    0 3781 Feb 25  2016 start.sh
```

```
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 sys
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 tmp
drwxr-xr-x  7 1000 1000    0 Dec 15  2014 usr
drwxr-xr-x  2 1000 1000    0 Jul  9  2013 var
```

Listing 3.3: Contents of JFFS2 image.

In a file named `/etc/init.d/S10mpp`, which is part of the Linux init system, it is determined that it runs a Telnet server. The Telnet server is unnecessary, and the user manual does not mention this service anywhere, neither is the password shown in the user manual.

The password has been determined using a program called "John the ripper". "John the ripper" is a tool for determining the plaintext behind a hash value, by exhaustive key search, as explained in Section 3.1. The hash of the root user password is found in `/etc/passwd`, and its value is:

`$1$.fYIOcXG$YXooBDAWHtN6IKFXD3yj5/`

The prefix `$1$` specifies that it is the `MD5crypt` algorithm that is used to hash the password. The following `.fYIOcXG$` is the salt, which is used to prevent rainbow table attacks.

The password has been brute forced using a machine with 64 CPU cores (AMD Opteron 6272) in about 48 hours, and the password is `anni2013`. Here the password has been cracked relatively quickly, and without the use of Graphical Processing Units (GPUs), which can make the process significantly faster.

The quick discovery of the plaintext password is also related to the used hashing algorithm MD5crypt, which is obsolete. Instead a modern and secure hashing algorithm should be used, such as `bcrypt`. The `bcrypt` algorithm is around 16 times slow to compute than `MD5crypt` [44].

3.2.1 Summary

This section has described a method for finding weaknesses in IoT devices on the market, by analysing the firmware that is released on the Internet. The process that is followed is fairly simple, as soon as the firmware image has been extracted, the entire Linux system can be inspected.

Since Linux is a standardised in the way that the file system is structured, it should be possible to write scripts that determine various information about firmware images. Information that can be useful for attackers, for finding weaknesses that may be exposed.

3.3 Alternative Authentication Methods

In Sections 3.1 and 3.2, it has been established that passwords can be weaknesses in computer systems. Therefore, this section describes alternative methods for authentication

that can also apply to IoT systems.

The methods that are analysed are: public-key authentication, OAuth and two-factor authentication, where either Hash-based One Time Password (HOTP) or Time-based One Time Password (TOTP) are commonly utilised.

3.3.1 Public-key Authentication

Public-key authentication is often used by system administrators to manage Linux servers, since it allows login without entering a password. Instead authentication is performed by cryptographically signing a message by using public-key cryptography [45]. If an entity is able to sign a message such that it can be validated using a specific public key, then it proves that the entity has access to corresponding private key.

The SSH protocol implements support for public-key authentication (example seen on Listing 3.4), but there is also support for it in applications that rely on Secure Sockets Layer (SSL). In SSL, the server can be configured to request a client certificate, which is essentially a public key that is verified by another party. By proving that the client has access to the corresponding private key, it can utilise the certificate to authenticate itself during a SSL handshake. The browser typically prompts the user to select a matching certificate, this is seen on Figure 3.1.

A benefit of using public-key authentication over e.g. a symmetric key approach, is that when using public keys, no secret information is stored on the server side. A symmetric key approach would indicate that the same key is stored on both the server and the client, which means that if the server is compromised, it might be possible to reuse keys recovered to compromise other servers. This is not the case with public-key authentication, as the public key does not unlock access to the server.

```
1  OpenSSH_7.4p1, LibreSSL 2.5.0
2  debug1: Reading configuration data /etc/ssh/ssh_config
3  debug1: Connecting to github.com [192.30.253.112] port 22.
4  debug1: Connection established.
5  debug1: identity file /Users/user/.ssh/id_rsa type 1
6  (...snip...)
7  debug1: Server host key: ssh-rsa
8      SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8
9  debug1: Host 'github.com' is known and matches the RSA host key.
10 debug1: Found key in /Users/user/.ssh/known_hosts:55
11 (...snip...)
12 debug1: Authentications that can continue: publickey
13 debug1: Next authentication method: publickey
14 debug1: Offering RSA public key: /Users/user/.ssh/id_rsa
15 debug1: Server accepts key: pkalg ssh-rsa blen 279
16 debug1: Authentication succeeded (publickey).
17 Authenticated to github.com ([192.30.253.112]:22).
18 (...snip...)
```

Listing 3.4: SSH handshake that uses public-key authentication.

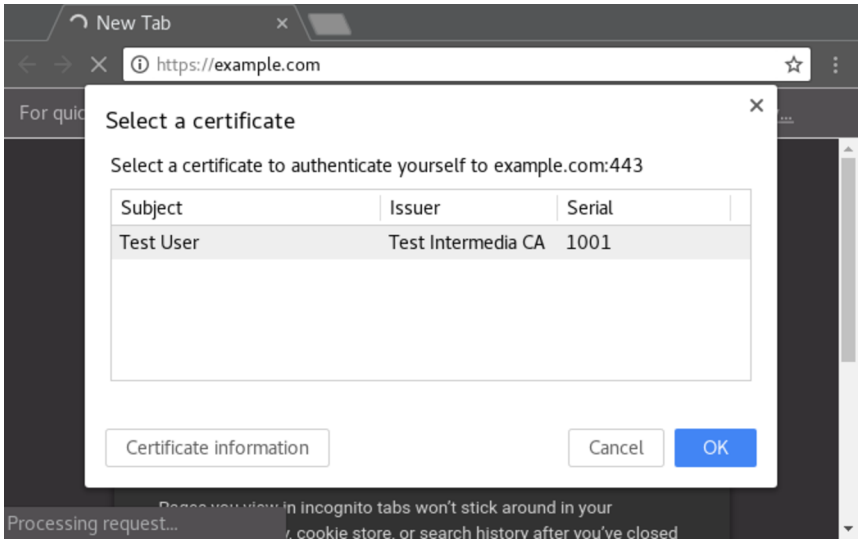

Figure 3.1: Browser prompting for a client certificate during SSL handshake.

3.3.2 Open Authorization Framework (OAuth)

The OAuth is an authentication and authorization framework that can be used to grant third parties access to resources, without the third party needing to have access to sensitive user credentials [46].

In the OAuth standard there are four entities, which act according to Figure 3.2.

The Client represents the entity that wants to access data that is stored on the Resource Server.

The Resource Owner represents the user, which owns the data. It is the Resource Owner that grants access to the data on a Resource Server to a Client.

The Authorization Server issues access tokens that can be used by the client to access data on the Resource Server. This is only done after it has been authorized by the Resource Owner.

Using OAuth for IoT Devices

To use the OAuth protocol to implement authentication and authorization in IoT devices, the different entities would then be:

The Client is the user-agent or application that wants to access the IoT device.

The IoT device is the Resource Server.

The Resource Owner is the user, which can grant access to the Client to access the Resource Server.

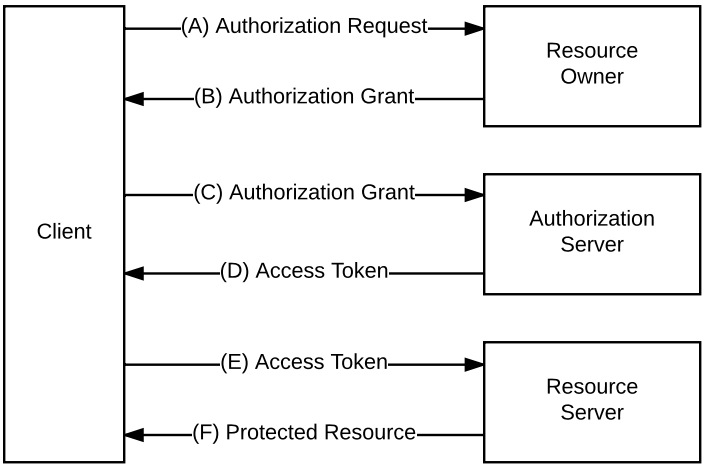

Figure 3.2: OAuth authentication flow.

The Authorization Server is a service hosted by the manufacturer, which is responsible for accepting the user's credentials, and allowing password recovery and similar services. The Authorization Server can also implement other means to increase security, such as identifying where the user connects from, sending notifications about suspicious activity or implementing two-factor authentication.

Considerations

When using the OAuth protocol to authenticate user access to IoT devices, a new Single Point of Failure (SPOF) is created, the Authorization Server. If the Authorization Server is taken offline, it becomes impossible to access the device, without an independent means of authentication.

The Authorization Server must also be secured, as it contains the keys that can grant access to many IoT devices at the same time. However, considering that there is practically unlimited resources compared to an IoT device, and it is easier and faster to upgrade a single service, compared to many thousands of IoT devices, it may still be able to increase security in the devices. The Authorization Server should be secured in the same way that other web services are secured.

3.3.3 Two-factor Authentication

Two-factor authentication indicates that multiple factors participate in authentication attempts. Typically, a password is one of the factors, and then a second factor is added to overcome the weaknesses that exist by using password authentication. A common schema is to require "something that you now" with "something that you have", where a password is "something that you know" and a physical device, smartphone or similar

is "something that you have" [41, Chap. 7.6]. The benefit is that if the technique is implemented securely, only the person fulfilling both requirements can be authenticated.

RSA SecurID

For the physical factor, there are multiple possibilities. Some of the earliest digital solutions is the RSA SecurID, which is a physical device with a display that can generate a time limited code. This code must then be used together with a password to authenticate. The RSA SecurID is seen on Figure 3.3 [47].

Figure 3.3: A RSA SecurID code generator.

Smart Phone Application

Another solution that has received a lot of attention in this decade is using a smartphone application, such as the Google Authenticator, FreeOTP or similar. These applications can generate HOTPs or TOTPs, which are used for authentication, similar to the RSA SecurID [48]. A benefit of smartphone applications over a separate physical device, is that a smartphone is something that is typically brought with you.

TOTP is a method for generating a code that is dependent on the current time, and thereby ensuring that it is time limited. This requires that the time is synchronized between the TOTP generator and validator.

The other method, HOTP, generates codes based on a counter. Whenever a code has been used, the counter is incremented, and a new code is valid the next time. This ensures that each code can only be used a single time, but it is not time limited as TOTP is.

However, this solution relies on the security of the smartphone to keep the secret information needed for the codes safe. If the smartphone has been compromised, it can also compromise the security gained by two-factor authentication. Therefore, this solution is considered less secure than the RSA SecurID.

SMS or Phone Call

A third solution is either sending a text message to the user, or calling the user's phone to confirm the login. This is a more universal solution, since it does not require a smartphone for implementation. However, if the phone has been compromised, text

messages may still be forwarded without the knowledge of the user, and as such the method is less secure than the RSA SecurID.

3.3.4 Summary

This section has described alternative authentication methods that can be used instead of accepting username and password as credentials for IoT devices. The methods that have been described should each increase the difficulty in gaining unauthorized access to IoT devices.

In the remaining part of this chapter, network related techniques are described.

3.4 Network Address Translation

This section describes a technique that is widely used on the Internet, namely NAT. NAT was invented as a way to connect more computers to the Internet, than there were IP addresses for. With NAT it is possible to extend the

$$2^{32} = 4,294,967,296 \text{ addresses}$$

to connect practically unlimited numbers of computers, as NAT can be used in a nested fashion, by implementing multiple layers of NAT. Multiple layers of NAT have been adopted by many newer ISPs, as they are assigned only 1024 IP addresses for their customers. This is known as Large Scale NAT (LSN) or in some cases Carrier Grade NAT (CGN).

NAT works by translating the IP address of a Local Area Network (LAN) computer, when it accesses the Internet. Then by recording this translation in an internal connection table, the NAT gateway reverses the translation when a response is received from the Internet server.

3.4.1 Private IP Addresses

To facilitate this method, LAN computers must be assigned IP addresses that are not in use on the Internet, or the consequence is that some parts of the Internet are inaccessible. This is handled by the three private IP address assignments in [49], which specifies that the networks — specified using Classless Inter Domain Routing (CIDR) notation — cannot exist on the Internet:

10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16.

CIDR notation is a used to express a network including its size. The number after the slash indicates the number of one bits in the subnet mask. For example, 192.168.0.0/16 is equal to network 192.168.0.0 and subnet mask 255.255.0.0, since 255.255.0.0 has 16 one bits set (from the most significant side).

The three private IP address spaces contain

$$2^{32-8} + 2^{32-12} + 2^{32-16} = 17,891,328 \text{ addresses}$$

that can be locally administered, without impacting any Internet related activity.

On top of this assignment, an address space for LSN has been reserved, which can be used by ISPs. This address space for LSN is not routable from the Internet, and must only exist within ISPs' networks. The network for LSN in CIDR notation is:

$$100.64.0.0/10$$

which again allows for

$$2^{32-10} = 4,194,304 \text{ addresses}$$

and an ISP can theoretically reuse this space for every public IP address they own.

3.4.2 Implicit Security

Even though NAT breaks the host-to-host connection principle of the Network layer, it implicitly denies ingoing traffic to devices that are behind NAT. While this decreases the attack surface significantly, because no server applications can be accessed from the Internet that problem could have been solved without NAT by using a firewall.

3.4.3 Considerations Regarding NAT

A consideration regarding NAT, and especially in the context of LSN, is the increased computational requirements. When using NAT, every single packet that flows through the gateway, must be rewritten by the NAT code, and the translations must be remembered for as long as the connection is active. This increases the requirements to the equipment that ISPs are using for their access network.

3.4.4 Summary

In this section, NAT has been described, and the widespread use of it. It should be noted that with the implementation of Internet Protocol version 6 (IPv6), NAT is no longer required, as there is going to be enough addresses for all devices.

In the next section, different types of IoT devices are described, and how they communicate with Internet based services.

3.5 Types of IoT Equipment

This section describes some of the different ways IoT devices communicate with e.g. cloud services or simply the corresponding mobile applications. By determining any patterns in traffic from IoT devices, it is easier to create profiles of them.

There are multiple ways that IoT devices can operate, to be controlled and communicate with other services. A basic scheme of Internet services is the client-server model, where a client contacts a server that is hosting a service. This scheme can be utilized in two ways, either the IoT device is the client, which is contacting a remote service. Or the IoT device can be the server, and accepting connections from clients.

3.5.1 IoT Device acting as a Server

In the case where the IoT device is acting as a server, a server program is running on the IoT device. There are two possibilities for clients, it can either be a LAN device, which connects directly without being routed by the IGD. Or it can be a remote Internet device, which is communicating through the IGD.

In the first case, it is possible to eavesdrop on the traffic if it passed through the IGD on the data-link layer, e.g. if the client and server are bridged through the IGD. However this may not always be the case, if two devices are connected using a switch that is separate from the IGD. In this situation, the traffic between the client and the server is not going to flow through the IGD.

The second case is complicated by the appearance of one or more NAT gateways, and it may not be possible to connect to the device remotely. If the IGD is behind a LSN, it may simply not be possible to accept incoming connections from the Internet. If it is not, then incoming connections require configuration in the IGD, in order to forward new connections to the IoT device.

When the IoT device is hosting a server program, it can become the target of a vulnerability. There exist many vulnerabilities that rely exploiting a weakness in a service, by crafting a malicious request. Some of the exploits that are well known are e.g. Shell-Shock, Heartbleed and various SMB exploits in the Windows operating system. If the IoT device is then accessible from the public Internet, it can be compromised by attackers, if they have found a vulnerability to exploit.

3.5.2 IoT Device acting as a Client

On the other hand, if the IoT device is acting as the client, which connects to the Internet, it does not need to accept incoming request. Instead it connects to another service, and acts as a client. This situation can bypass the restrictions that are imposed by NAT gateways.

When the IoT device is the client, it is more difficult to attack, as the attacker has to position himself between the IoT device and the service it is contacting. Thereby performing a Man in the Middle (MITM) attack. If an attacker manages to position himself, he can then serve crafted responses to the client. While this is considerably more difficult, due to the necessity of the MITM position, it can also be used to compromise the device.

First of all, a DoS attack can be performed, by simply dropping the traffic between the IoT device and the remote service. Another attack could be an IoT device that repeatedly checks for new firmware updates on the Internet. In the MITM position, it can be possible to serve a malicious firmware update to the device.

3.5.3 Summary

In this section, two different models for how IoT devices are connected to Internet based services have been described.

Of course, IoT devices can be a combination of both. Even if the device operates as a server, it can still use clients programs to perform other activities, e.g. synchronizing time using Network Time Protocol (NTP) or checking for firmware upgrades, and in combination with either, it could also be performing DNS lookups. In the next section, the behaviour of IoT devices is be determined, by using Deep Packet Inspection (DPI).

3.6 Deep Packet Inspection

In this section, a technique known as DPI is described. This technique can be used to identify traffic on the Internet. In DPI, the payload of network traffic is inspected to determine which protocol is being used. This is opposed to only inspecting simple numbers, such as source and destination ports, which can only tell limited information about the communication. For example, traffic going to port 80 would be identified as HTTP traffic, due to the port number registration with the Internet Assigned Numbers Authority (IANA). However, this may not be truthful, as there are no limitations in place to prevent the port for being used for other protocols. And a service can also run on a different port than the IANA assigned.

This is where DPI enters the picture. By looking at the payload of the communication, instead of only e.g. port numbers, it can be determined more precisely what protocol is being used. Following the HTTP example, only if the first payload coming from the client contains something like “METHOD url HTTP/1.x” is it HTTP traffic. Similarly the response from the server also have to follow certain rules about the what it sends. By looking into these payload, it becomes possible to identify traffic, regardless of which port numbers it is communicating on. And by identification, it also becomes possible to block activity.

By using DPI, it also becomes possible to generate more precise profiles of devices, as with the HTTP example, the client sends a User Agent string — containing the software version it is running — and the server often includes a software version as well.

However, DPI can be very resource intensive, as it requires checking the payloads with different matchers, until the correct protocol is identified. The definition of a matcher here, is something that is looking for a specific protocol in a payload. The more protocols that can be matched, the more processing time is required to inspect individual payloads.

3.6.1 Summary

In this section, DPI has been described, which can be used to identify network traffic more precisely than by relying on port numbers and IP protocols. DPI is impaired by the use of data encryption, this is further described in the next section. Encrypted data is more difficult to identify using DPI, since only the outer unencrypted layer can be analysed.

3.7 Data Encryption

This section describes some of the different ways that data is encrypted on the Internet, and also some ways of identifying the application, even though the data is encrypted. Data encryption is becoming more widely used everywhere, many websites have begun to offer Hypertext Transfer Protocol Secure (HTTPS) instead of HTTP, and certain websites are now only accessible using HTTPS. When HTTPS is used, it becomes impossible to determine, what is being transmitted between the server and the client, as SSL is being used [50].

The SSL protocol, is an accepted practice whereby to establish a secure communication channel, over an insecure medium — e.g. the Internet [41, Chap. 2]. On the inside of the secure communication channel, an application protocol can exchange messages without the risk of being tampered with or eavesdropped on.

When data encryption is used, it becomes difficult to do DPI, as is mentioned in the previous section, Section 3.6. However, there can still be gathered certain information, which is exchanged in plaintext, when the secure channel is established. When using SSL for example, the protocol consists of multiple messages, where information can extracted, the handshake protocol is shown in Figure 3.4.

3.7.1 Messages Exchanged by the SSL Handshake Protocol

ClientHello

In the first message, which is always sent by the client of an SSL session, it is possible to determine the supported Cipher Suites, Compression Methods and Extensions. A

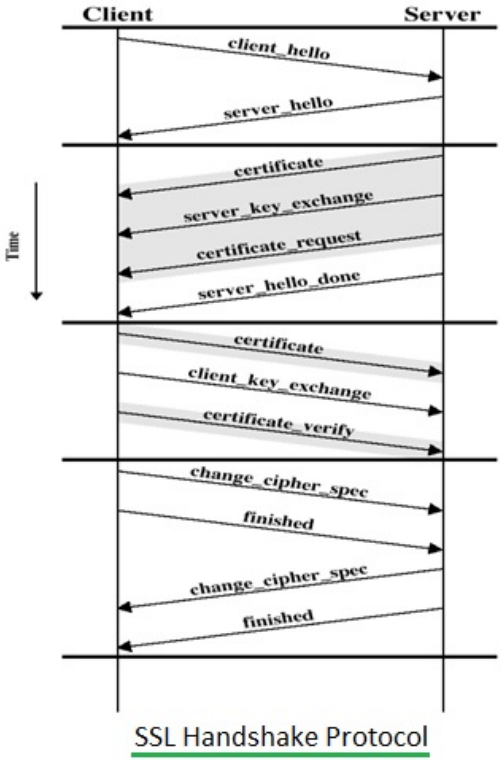

Figure 3.4: Messages exchanged in the SSL Handshake Protocol.

Cipher Suite is a combination of a Key Exchange algorithm, Authentication algorithm, and Data Encryption algorithm. In low resource environments, such as smaller IoT devices, the number of Cipher Suites that are supported may be limited to save resources, and the same goes for Compression Methods and Extensions. The supported Cipher Suites, Compression Methods and Extensions can then be used to coarsely divide devices between multiple groups.

Two of the useful Extensions that can be used to get more information about the connections are Server Name Indication (SNI) and Application-layer Protocol Negotiation (ALPN). The SNI extension is used when a server hosts multiple virtual servers on the same IP address, in order to determine the correct certificate to present to the client. The value of the SNI extension is the domain name that was resolved to find the IP address of the server. The other extension, ALPN, is used to negotiate the application-layer protocol to be used inside the encrypted channel.

Both of these extensions can contain information that can be used to distinguish between different SSL connections.

ServerHello

In the ServerHello message, which follows immediately after the ClientHello message, the server responds with the selected Cipher Suite and Compression Method. This does not grant much information, as the server only selects between those sent in the ClientHello.

ServerCertificate

Immediately after sending the ServerHello message, the server follows up with the ServerCertificate message. In this message, it is possible to gather information about the remote server that the client connected to. For example, the server certificate contains information such as Valid From, Valid To, Common Name (Domain Name), information about the certificate issuer, and information about the server's public key, see Figure 3.5. All of this information is something that persists for a longer period of time, but is not completely static.

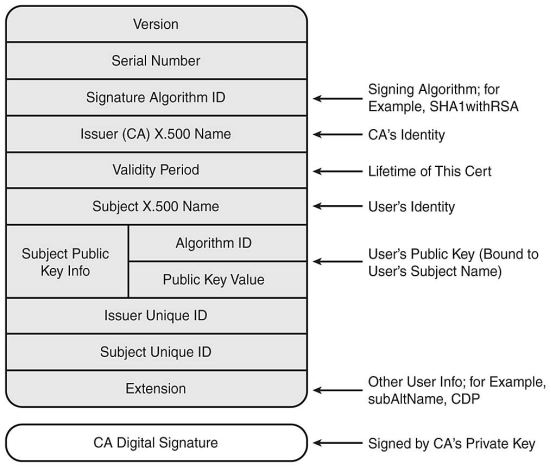

Figure 3.5: Example of an X.509 certificate, which is used in an SSL handshake.

The other SSL handshake messages are related to the key exchange procedure, and setting up the negotiated cipher suite, and as such do not contain useful information. Therefore they are omitted in this section.

3.7.2 Summary

This section has briefly described the SSL protocol, which is used for most encrypted communication between client and server programs. There are, of course, other encryption protocols in use, but SSL is the most prevalent one, since basicly any traffic can be transmitted inside the encrypted channel.

The next section describes some of the Linux related tools that can be used to set up firewalling, routing, virtual network namespaces and more.

3.8 Linux Router

This section describes the procedure that is used to create a router on the Linux operating system. The Linux kernel is capable of forwarding traffic between interfaces, and performing filtering and NAT in the process.

3.8.1 Forwarding

By default, forwarding is disabled and must be enabled in the system to create a router. Forwarding can be enabled in the kernel, by setting the kernel parameter `net.ipv4.ip_forward` to 1. Listing 3.5 shows how to enable forwarding.

```
[root@router ~] sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

Listing 3.5: Enabling forwarding in the kernel.

When forwarding has been enabled, the Linux kernel is permitted to forward received packets, where the destination address does not belong to the system. The packet is forwarded to the interface that the routing table has selected for the destination address.

Listing 3.6 shows an example of a routing table in Linux. In the routing table, it can be seen that the two networks 192.51.100.0/24 and 192.168.0.0/24 are locally connected, in other words, the system has interfaces with addresses in those networks.

The route which begins with "default" is selected for outgoing traffic that is not matched by other routes. This is the default gateway.

```
[root@router ~] ip route
default via 198.51.100.1 dev eth0
192.168.0.0/24 dev eth1 proto kernel scope link src 192.168.0.1
198.51.100.0/24 dev eth0 proto kernel scope link src 198.51.100.91
```

Listing 3.6: Linux routing table example.

This setup is sufficient that devices on the 192.168.0.0/24 network can transmit packets to the Internet, through the Linux router. However, other hosts on the Internet cannot reply to 192.168.0.0/24, since it is a private IP address space, as mentioned in Section 3.4.1. Therefore, the Linux router must perform NAT. In the Linux kernel, NAT happens in the netfilter framework.

3.8.2 Netfilter

Netfilter is a framework in the Linux kernel that offers the features required for NATing, firewalling and connection tracking. It does so by providing a set of hooks, where traffic can be manipulated when it moves through the kernel. Two hooks are registered by the iptables tables `nat` and `filter`, where NAT and firewalling occur respectively. Each table contains chains, which are a set of rules that are processed in the order they occur.

NAT

In the `nat` table, NAT can be performed. All the traffic that is being sent from the Linux router go through the `POSTROUTING` chain in the `nat` table. In this table, the source address can be manipulated through a rule.

Listing 3.7 shows a rule being added. The rule specifies that all traffic with source address in 192.168.0.0/24 being transmitted on the `eth0` network interface, should have the source address translated (masqueraded) to the current IP address of the `eth0` interface.

```
[root@router ~] iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth0 -j
    ↪ MASQUERADE
[root@router ~] iptables -t nat -nvL POSTROUTING
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target      prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      eth0    192.168.0.0/24       0.0.0.0/0
```

Listing 3.7: Inserting a rule to translate the source address.

Through this rule, outgoing traffic is NAT'ed and LAN devices can communicate with Internet devices. Netfilter handles the reverse translation automatically.

Firewall

In order to create a firewall in the Linux router, the `filter` table is used. As the name implies, it is where packets are filtered. Packets that are forwarded in the Linux router pass through the `FORWARD` chain of the `filter` table. In this chain, it is possible to permit or deny specific connections.

An example of a firewall policy is shown in Listing 3.8.

```
[root@router ~] iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j
    ↪ ACCEPT
[root@router ~] iptables -A FORWARD -s 192.168.0.0/24 -o eth0 -j ACCEPT
[root@router ~] iptables -A FORWARD -j DROP
[root@router ~] iptables -nvL FORWARD --line
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source
    ↪ destination
1        0     0 ACCEPT     all  --  *      *       0.0.0.0/0
    ↪ 0.0.0.0/0           state RELATED,ESTABLISHED
2        0     0 ACCEPT     all  --  *      eth0    192.168.0.0/24
    ↪ 0.0.0.0/0
3        0     0 DROP       all  --  *      *       0.0.0.0/0
    ↪ 0.0.0.0/0
```

Listing 3.8: Example of a firewall policy in a Linux router.

In the firewall policy shown in Listing 3.8, rule 1 permits packets belonging to established connections, rule 2 accept traffic that arrives from the LAN network and is routed to the Internet. Rule 3 specifies that all packets should be dropped. However, the last rule does not affect the rules before it, since when a packet has been ACCEPT'ed it stops progressing through the chain.

3.8.3 Summary

In this section, a description of how to create a Linux router, NAT gateway and firewall has been made. The instructions have been basic, compared to the possibilities in `iptables`. This section is the end of the technical analysis, where techniques that are used in the following chapters have been described. In the next chapter, the design of the solution is described.

CHAPTER

4

SYSTEM DESIGN

This chapter presents the design of the solution proposed in Section 2.9.3. The chapter starts with a system overview to present the big picture of the solution. After the overview, the system is divided into parts, starting with the profiles, which define the IoT devices. Followed by the architecture of the software, and then the plugins used in the software.

4.1 System Overview

On Figure 4.1 a diagram of a typical home network is seen. As it can be seen both Multiple Purpose Devices (MPDs) and IoT devices are connected to the IGD. To protect all devices behind the IGD, the solution is a program that is hosted on the IGD, as traffic to all devices – IoT devices and MPDs – goes through the IGD. Though, this project only focuses on the IoT devices, as mentioned in Section 2.10.

The solution is a system that uses profiles to filter traffic for connected devices. The profiles are created by the system itself, when a new device is connected. This is done by analysing traffic to and from the device, and using DPI to learn specific protocols used by the device.

A new profile is started whenever a device is connected and an IP address is assigned. It is assumed that a device retains the same IP address, if it is disconnected and then reconnected. This can be done by creating static DHCP allocations. When the profile has been completed, it can be converted to firewall rules and applied to the IP address of the device, in the IGD firewall.

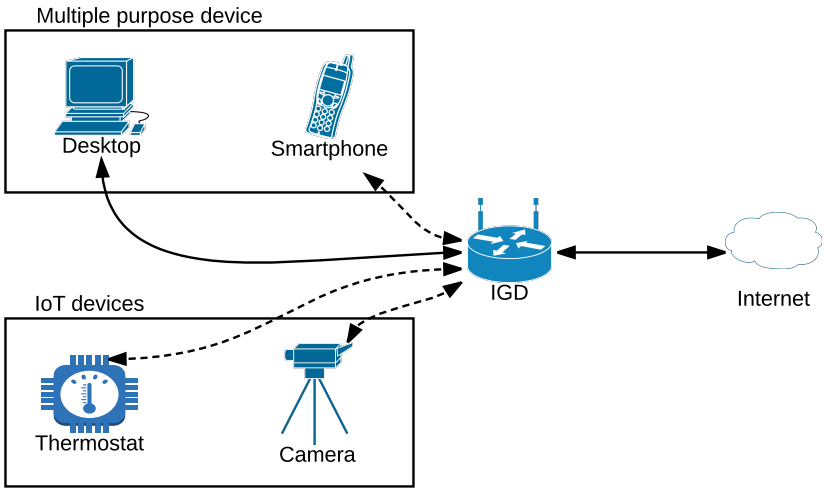

Figure 4.1: Top view diagram of a typical local area network.

4.2 Profiles

A profile defines a device, and includes two components. The first component is information about what traffic is accepted to and from the IoT device. The other component is information that can be used to identify the device, thereby a unique fingerprint.

4.2.1 Traffic Information

In order to describe the traffic that should be accepted, the profile must include information about what servers are running on the device, and what it connects to on the Internet. Since DNS is used to resolve the IP endpoints on the Internet, DNS queries must also be included in the profile. These three elements should describe the Internet traffic that a device produces.

4.2.2 Fingerprint Information

To create a fingerprint of a device, the above information about Internet traffic can be useful. For example, two devices that contact the same domain name are more likely to be related. While true in some cases, there are also more generic Internet services that different devices can connect to, for example NTP servers.

By using DPI, it is also possible to determine specific protocols, and information such as protocol version or software versions. This information can be included in the fingerprints as well.

Along with traffic information, which flows to the Internet, there is also traffic that only reaches the LAN. For example, the DHCP protocol and Media Access Control (MAC) addresses never reach the Internet. From the DHCP protocol, the device can provide

a host name, which can be included in the fingerprint. MAC addresses are defined as two 24-bit values, where the first value is a vendor prefix, and the remaining part is a number assigned by the vendor. The vendor prefix can be included in the fingerprint, since it indicates which vendor produces a device.

The information to be included in the profile is listed here:

- Host name of device.
- Vendor prefix of MAC address.
- DNS queries made by device.
- Services running on device.
- Services contacted by device (device acts as client).

4.2.3 DHCP Request

The host name and vendor prefix of MAC address of the device can be found in the DHCP request. The DHCP request is essentially the first message that the IoT device sends, since it used to configure the device with an IP address.

From the DHCP request, the host name of the device can be learned and the MAC address can be recorded. The host name and the vendor prefix of MAC address are used in the fingerprint of the device.

An example of a DHCP acknowledgement can be seen in Listing 4.1.

```
Bootstrap Protocol (Request)
...
    Client MAC address: Tado_10:47:e1 (ec:e5:12:10:47:e1)
    Magic cookie: DHCP
...
    Option: (50) Requested IP Address
        Length: 4
        Requested IP Address: 192.168.1.93
    Option: (12) Host Name
        Length: 4
        Host Name: tado
...
```

Listing 4.1: Excerpt of a DHCP acknowledgement from a Tado bridge. Content produced by Wireshark.

4.2.4 IoT Device as Server

If a device hosts any services which should be reported, this is done by analysing the incoming traffic to the device. If a new connection is made to the device, the protocol it uses is analysed and what port it is running on. The connection is then tracked, and the response from the device, tells which software that the host is running. E.g. a web server can both be hosted using Apache or nginx.

On Figure 4.2, a flowchart explaining the process of a client connection to a IoT device is shown.

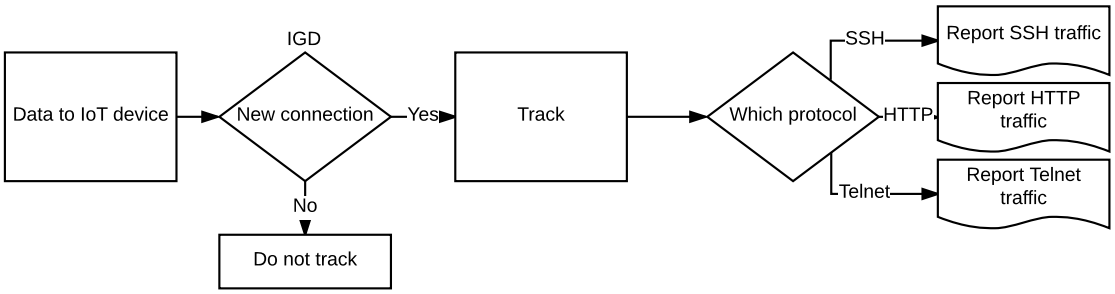

Figure 4.2: Flow chart of detection of services hosted on devices.

4.2.5 IoT Device as Client

In addition to Section 4.2.4, when a device contacts remote hosted services, it should also be reported.

If the client contacts other remote services, both the protocol is reported, but also the destination domain name or IP, depending on whether it has made a DNS query or not.

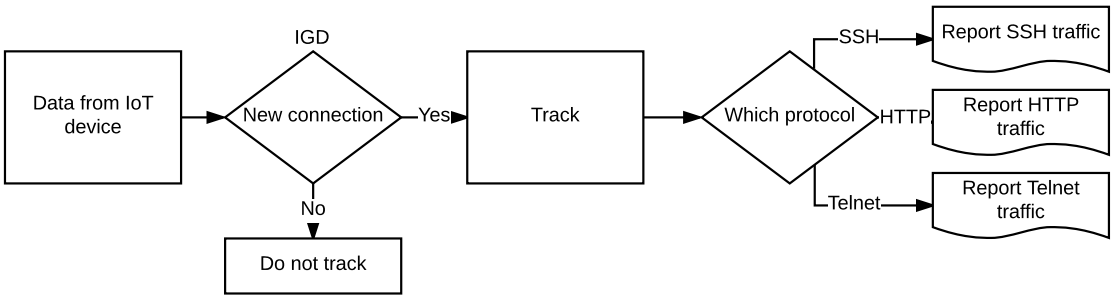

Figure 4.3: Flow chart of IoT device contacting server.

4.2.6 DNS Queries

If the IoT devices contact remote services, it might also make a DNS query. If such a query is made, it is reported, and both the domain name and resolved IP is reported. An excerpt of a DNS response is seen in Listing 4.2. In this listing, it can be seen that a device asks for the IP address of the domain name `i.my.tado.com`, in the response there are two IPs addresses, both should be reported.

```
Domain Name System (response)
...
    Questions: 1
...
    Answer RRs: 2
...
        i.my.tado.com: type A, class IN, addr 54.171.136.152
        i.my.tado.com: type A, class IN, addr 52.48.170.120
...
```

Listing 4.2: An excerpt of a DNS response from a DNS query on my.tado.com. Content produced by Wireshark.

Based on a query like the one seen on Listing 4.2, a mapping between the domain name and IP address is saved in the profile. An example of a report is shown on Figure 4.4.

DNS Query Report	
IP	192.168.1.110
Query	i.my.tado.com
Response	54.171.136.152 52.48.170.120

Figure 4.4: DNS query report.

4.2.7 Summary

The parameters for the profile have been chosen, the chosen parameters are: “Host name”, “vendor prefix of MAC”, “services on device”, “services contacted by device” and “DNS queries made by device”. The next section explains how the software for the IGD is designed in order to create these profiles, along with how the profiles should be shared such an IGD can download them instead of creating them.

4.3 Software Design

This section describes the general design of the software that creates profiles and handles how the profiles are shared. From a global perspective, there should be a central place to exchange profiles, where the IGD can lookup an existing profile for a device. This should benefit, in the long term that the devices can be identified more quickly and more reliably.

Other than the centralized repository of profiles, software has to be designed to run on the IGD. The software on the IGD handles analysing packet traffic and building profiles from the data.

Therefore, two applications are designed in the following sections. The two applications behave according to the client-server model, where the software on the IGD, can be considered a client that requests resources that are on the server, namely the centralized

repository. In the following, the two applications are named: Client Application and Server Application. Figure 4.5 shows the one-to-many relationship between the server and clients.

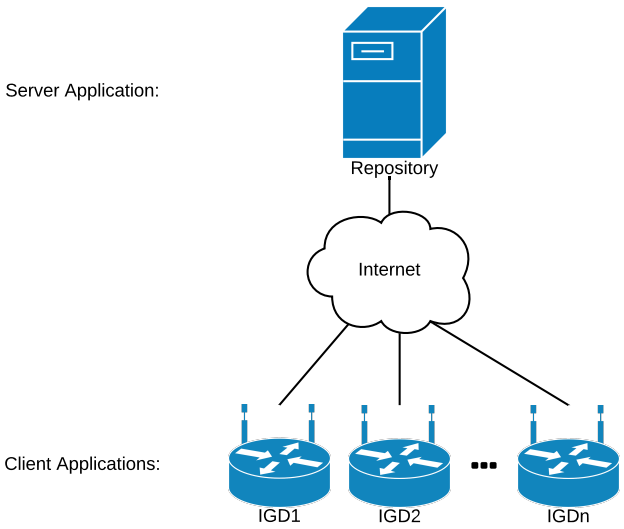

Figure 4.5: Client-server model nature of the applications.

4.4 Server Application Design

The Server Application is used by the Client Application, and must provide certain features to it. The Client Application uses the Server Application to query existing profiles, download specific profiles and upload profile information related to a device.

Since these features are only used by the Client Application, when there is a new device connected to it, it only needs to access the Server Application at that time. Therefore, there is no need for the Client Application to maintain a persistent connection with the Server Application at all times.

Whenever the Client Application needs to access the Server Application, it can do so in a session context, by sending a request and receiving a response. This approach is well supported by a RESTful Web Service, where the Client Application can use HTTP requests to access resources on the Server Application. Each request can then be processed and finished, and the session can be ended, until the Client Application needs to send the next request.

In Section 4.4.1, the functionality provided by the Server Application is described.

4.4.1 Functions

Tables 4.1 to 4.3 described the functions that are defined in the Server Application.

Function Name:	Query Existing Profiles
Input:	A full or partial Profile is accepted as input.
Output:	If found, any existing Profiles are returned.
Procedure:	1. Search in Profile database using input Profile. 2. If found, return the Profile that has most identical features. 3. If not found, send an appropriate response.

Table 4.1: Function: Query Existing Profiles.

Function Name:	Download Profile
Input:	A Profile identifier
Output:	An existing Profile
Procedure:	1. Search in Profile database using identifier. 2. If found, return the Profile. 3. If not found, send an appropriate response.

Table 4.2: Function: Download Profile.

Function Name:	Upload Profile
Input:	A Profile
Output:	Status message
Procedure:	1. Look for relevant existing Profiles. 2. If found, merge the incoming Profile with the existing. 3. If not found, create a new Profile in the repository using the input.

Table 4.3: Function: Upload Profile.

4.4.2 Repository Operation

The repository must store profiles in a kind of database. The underlying database type is decided by the implementation. In order to merge profiles, and thereby increase the confidence that each profile provides, an algorithm must determine the similarity of two profiles. This should be done based on the fingerprint information that is included in the profile.

4.4.3 Security Issues

Since the Server Application hosts device profiles that the Client Application trusts, precaution should be taken against malicious profiles. Namely, a way to prevent an attacker from uploading a profile which allows any traffic, thereby opening a hole in the solution.

The issue arises from the fact that anything that the Client Application sends, can be forged by an attacker. Therefore, an approach to determine whether a profile is legitimate must be found. A number of approaches are described below.

Consensus

One approach is to first consider a profile legitimate, when a number of similar profiles have been uploaded, and a consensus has been found. By doing this, a single uploaded profile cannot be spread to all Client Applications.

However, if an attacker controls a botnet, they can then upload multiple similar profiles and succeed anyway. Therefore, this approach is not secure enough on its own.

Verified Account

Another approach could be to require users running the Client Application to register and get verified. Verification could be through linking social media accounts, or verifying phone numbers etc. A verified account would then be required to upload profiles.

This increases the difficulty for an attacker that wants to upload a malicious profile. But it also decreases the usability from the user's perspective, who now has to register to use the system, and become verified.

Digital Signature

A third approach could be to make the Client Application sign the profiles before uploading them. However, since the Client Application would contain the key material to sign the profile, an attacker can still use the key to generate a malicious profile.

If this approach is combined with previously described Consensus approach, it would require an attacker to acquire multiple keys. The keys would need to be distributed with the IGD, in order be able to trust them. Therefore, an attacker needs to acquire many IGDs.

This approach has no usability impact on the user, since it would be handled from the IGD manufacturer's side.

4.4.4 Summary

In this section, the design of the Server Application has been described. Three approaches have been described to overcome a security issue that an attacker can generate fake profiles. The chosen solution is to use consensus and digital signature to prevent forgery of profiles. In the next section, the design of the Client Application is described.

4.5 Client Application Design

In the previous Section 4.4, the Server Application design has been described. In this section, the Client Application design is described.

The primary objective of the Client Application is to generate profiles from network traffic, in order to identify devices and create a model of their behaviour. The Client Application communicates with the Server Application, to retrieve existing information about a device. Information that is automatically learned from other IGDs, or provided by communities or manufacturers.

4.5.1 Traffic Inspection

In order to generate a profile about a device, the Client Application is required to inspect network traffic. Because the software runs on an IGD, it has complete access to all the traffic that are routed between networks. Effectively, it is in a man-in-the-middle position.

Traffic that is not routed — in other words, traffic between two devices on the same network — is not necessarily captured by the IGD. However, if a home network is segmented into a MPD network and an IoT device network, then that requires routing, and traffic must go through the IGD.

4.5.2 Profile Generation

When a new device gets physically connected to the network, it starts with a clean slate. Then as it starts to communicate, e.g. configuring an IP address using DHCP or checking for firmware upgrades, further information can be gathered about the device.

The information that is gathered can then be used to generate a profile that characterizes a device's behaviour. Whenever the Client Application encounters a type of traffic that the profile does not contain, it is added to the profile. This is done until the profile generation is stopped.

4.5.3 Profile Management

When a profile is generated, it should be sent to the Server Application, in order to share it with other users. Likewise, the Server Application should also be queried when a new device is connected, to retrieve an existing profile. These two tasks should happen automatically.

When a device is connected, and a preliminary fingerprint of it has been generated, the Server Application can be queried. A preliminary fingerprint could be just the MAC vendor prefix, host name, and the first DNS query that the device makes. Using this information, it can be determined if there is a device that resembles it, or if it is an unrecognised new device.

Similarly, when a profile has been generated for an unrecognised device, it should be sent to the Server Application.

4.5.4 Subsystems

In Sections 4.5.1 to 4.5.3 three tasks have been described. The Client Application is the combination of these three tasks. There are two ways to design the application, either all three tasks live in the same application, or they can be separate applications.

By creating one application, it can be simpler to communicate between the three tasks, as it can be done without encoding and decoding information to transfer it between tasks. Similarly, it might be able to run faster than three separate applications, since its memory is shared between all three tasks, and data can be accessed more directly.

However, by creating three separate applications, it is possible to move an application to a more powerful device if it is necessary. It is also possible to impose different restrictions on the three applications, since they do not require the same privileges necessarily.

The Client Application is divided into three applications, since the security benefits of privilege separation outweighs the limited performance gain from a single application.

The three subsystems are shown on Figure 4.6, and perform the following tasks:

Sniffer

- Receive packet traffic, either on the wire (network interface) or from a file (pcap).
- Track related packets, such as used by Transmission Control Protocol (TCP) handshakes, or requests and responses in a UDP based protocol, for example DNS.
- Decode application-layer protocols by using DPI techniques.
- Output information in the format of reports, which contain information about the host, and the traffic detected.

Profiler

- Receive information in the format of reports.
- Track services that devices interact with as a client, and services that devices host as servers.
- Track DNS queries, in order to substitute IP addresses in reports with domain names.
- Provide a real-time picture of the devices on the network.

Management

- Get notifications from the Profiler when a new device is connected.
- When preliminary information about a device has been collected, query the Server Application.
- When a profile is completed by the Profiler, upload it to the Server Application.
- Manage which profiles are active, and thereby manage the firewall rules on the IGD.

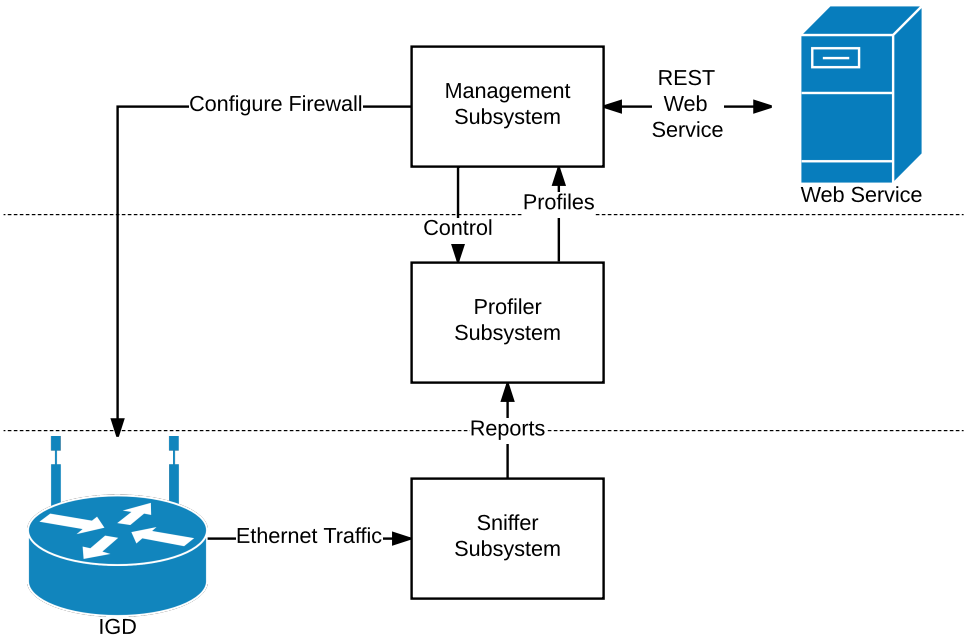

Figure 4.6: Overview of subsystems in the Client Application.

There are more benefits to three applications rather than one. For example, the sniffer has to handle a lot of data, because every packet must be inspected to determine if it

starts a new session or connection. However, the profiler only has to handle data coming from the sniffer, which only generates reports when certain events occur.

Therefore, those two subsystems can be optimised for different properties, where the sniffer should be optimised to process data quickly, the profiler can be more flexible, since it does not have the same speed requirement.

The management subsystem only interacts with the profiler, and controls when to stop a profile generation process. This removes some of the complexity of the profiler, since it should only focus on generating the profiles. The management subsystem is then a dedicated system to manage the Client Application on the IGD.

4.5.5 Summary

In this section, the overall design goals of the Client Application have been described. The design specifies three subsystems that should exchange data by producing reports in the sniffer, which are consumed in the profiler. The profiler and the management subsystem then exchange data in the format of profiles. In the next sections, Sections 4.6 to 4.8, the designs of the sniffer, profiler and management subsystem are elaborated.

4.6 Sniffer

In this section, the design choices of the Sniffer are elaborated. The Sniffer handles analysing packet traffic, and sending reports of detected traffic to the Traffic Profiler.

The Sniffer is designed in layers, according to the Open Systems Interconnection (OSI) model. This is done as protocols are nested, and therefore the lower layer protocols should be decoded first.

The lowest layer in the Sniffer handles receiving Ethernet packets from a network interface or from a PCAP file. These packets are then passed to the next layer, which decodes the IP headers, for Internet Protocol version 4 (IPv4) and IPv6 respectively. Thereafter, both TCP and UDP can be decoded and in the end, the packet is passed to the application layer decoders.

Whenever a layer detects what it is looking for, it can report that it has detected something. Following the overall architecture, this is the output of the Sniffer. A complete flowchart for the Sniffer is shown on Figure 4.7.

The following subsections describe the layers that the Sniffer must contain.

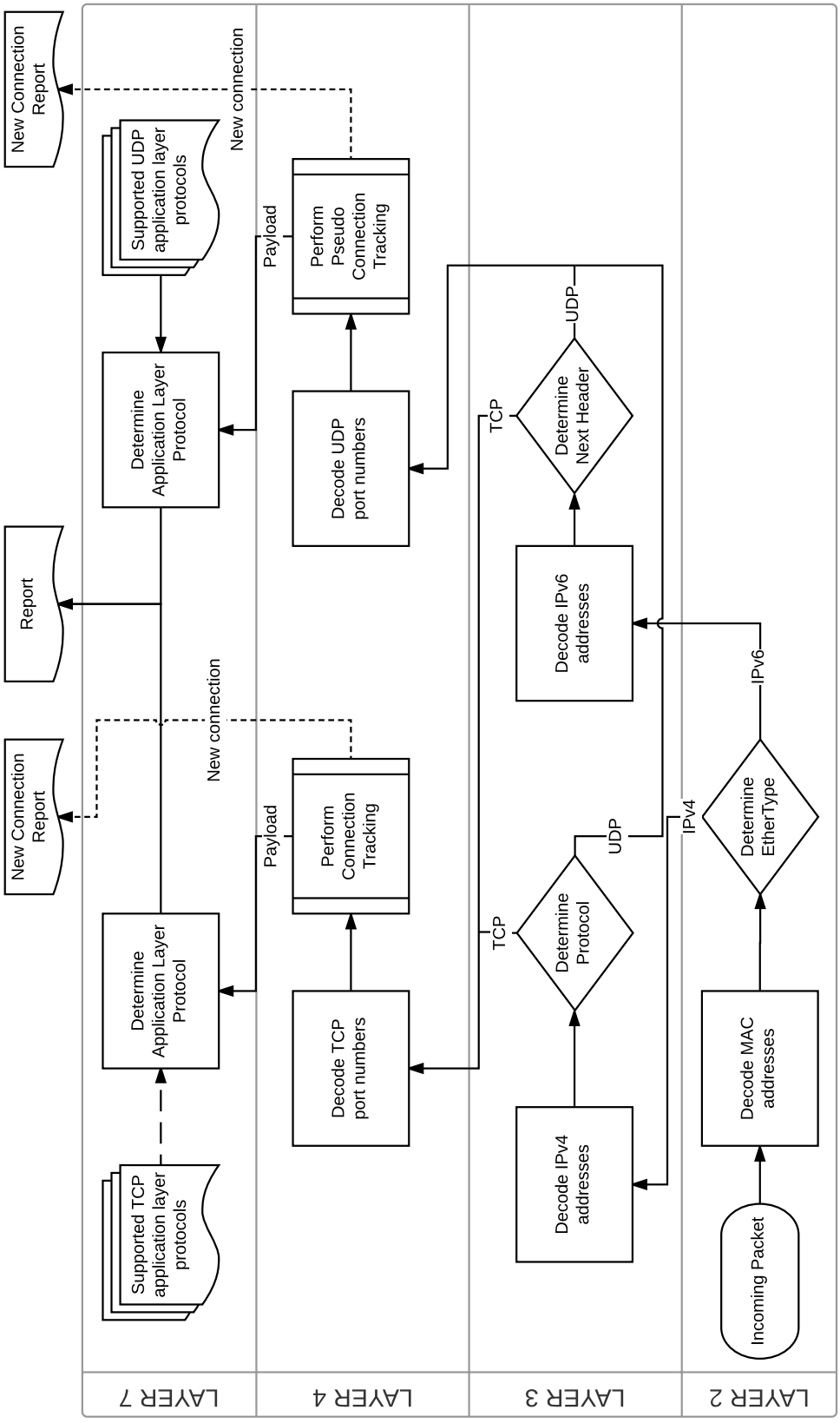

Figure 4.7: Flowchart of sniffer design.

4.6.1 Layer 2

On layer 2, a module must handle incoming packet traffic, and decode MAC addresses and EtherType from the Ethernet header (Figure 4.8). EtherType is used to determine which protocol the payload is using, e.g. IPv4 or IPv6.

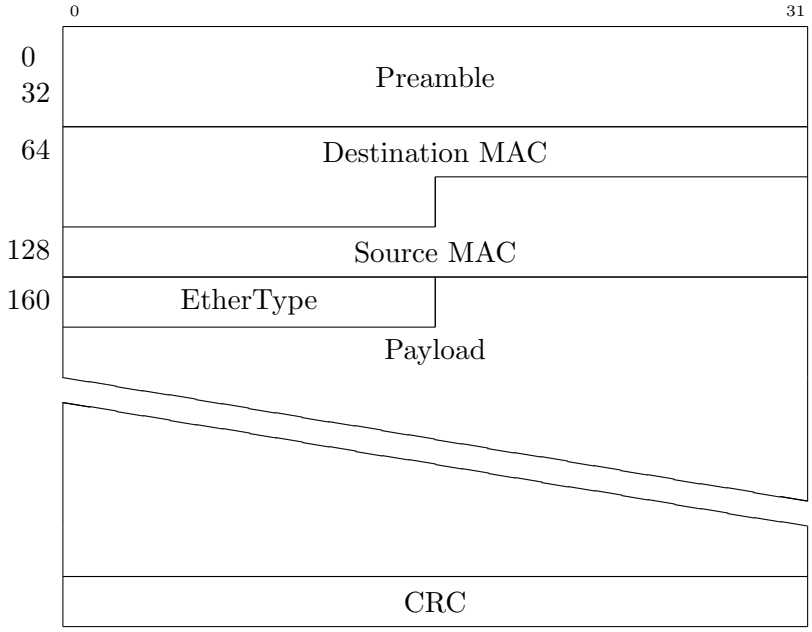

Figure 4.8: The Ethernet header.

The notable EtherType values to support are listed below:

- `0x0800` – IPv4.
- `0x86DD` – IPv6.

4.6.2 Layer 3

On layer 3, there are two protocols that should be supported, IPv4 and IPv6. From this layer, IP addresses should be decoded, and determined what the next layer protocol is. The IPv4 and IPv6 headers are shown on Figures 4.9 and 4.10 respectively.

Where the IPv4 header has room for options inside the header, it has been moved outside of the header in IPv6, and the header has a fixed length. In order to add IP related options in IPv6, the Next Header field is used, and a chain of headers is constructed. However, the Next Header field is also used to specify the transport layer protocol. This is a difference between IPv4 and IPv6 that is necessary to handle in the application, since the transport layer protocol is always specified at the same position in IPv4, but it might not be in IPv6. Therefore, in the IPv6 decoder, headers should be recursively decoded until a transport layer protocol Next Header value is found.

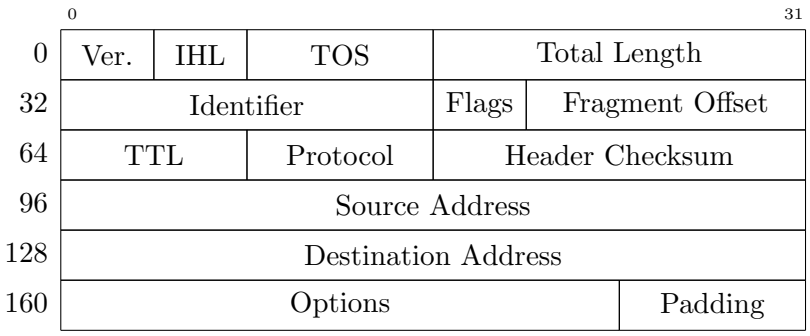

Figure 4.9: The IPv4 header.

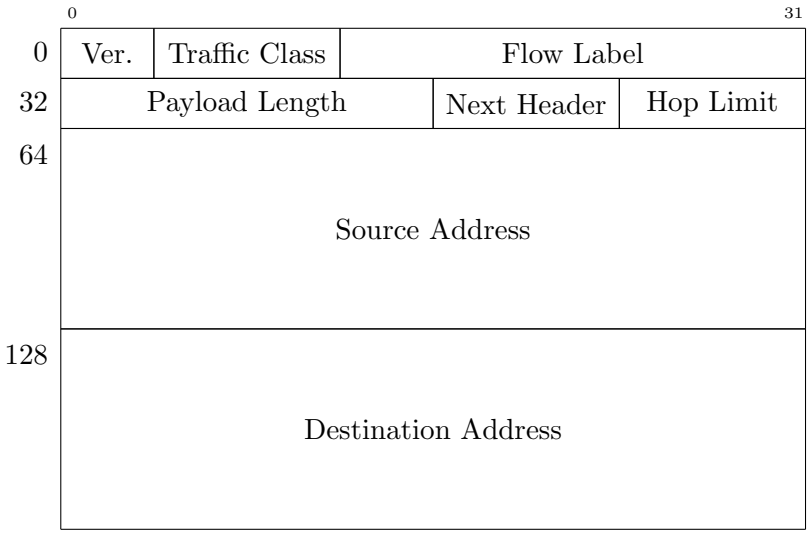

Figure 4.10: The IPv6 header.

4.6.3 Layer 4

On layer 4, there are two main protocols in use, TCP and UDP. Both TCP and UDP use port numbers to allow multiple connections to and from a host. The most notable differences between the two protocols are that TCP is stream-oriented, reliable and has built in algorithms to determine available bandwidth, where UDP is message-oriented, unreliable and does not use any algorithms to determine bandwidth.

TCP

The stream-oriented nature of TCP uses three-way handshake for establishing connections, and a pair of two-way handshakes to close them. These handshakes can be tracked and used to determine the duration of a connection, amount of traffic etc. By tracking connections, it is also possible to accurately determine the first (and subsequent ones) message that is sent, which can be analysed with DPI. The TCP header is shown on

Figure 4.11, where port numbers, sequence and acknowledge numbers and flags are used to perform connection tracking.

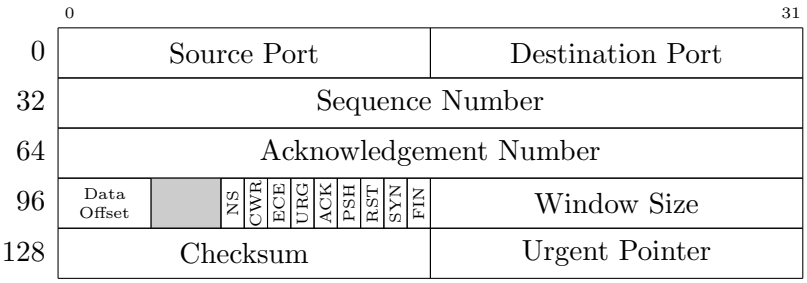

Figure 4.11: The TCP header.

UDP

In the message-oriented nature of UDP, it can be difficult to determine which message is the first, since there is no concept of connections in the protocol. Instead, UDP passes datagrams (messages) between hosts, to transmit application layer data. There is no reliable way to track "connections", since they do not exist, but it is possible to relate a message to another message, by looking at the Tuple of {Source Address, Source Port, Destination Address, Destination Port} (4-tuple).

This way, a pseudo connection can be tracked, whenever a message arrives on a 4-tuple, but there is no method to identify when the connection is closed. The method that is commonly used in stateful firewalls, is to mark the connection as closed after a timeout period without activity. This is the most that can be done, without looking into the payload.

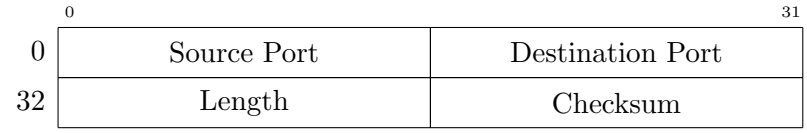

Figure 4.12: The UDP header.

ICMP

Internet Control Message Protocol (ICMP) is a protocol used to deliver diagnostic or control messages in IP networks. It is used to perform the `ping` command, which sends/receives ICMP Echo Request/Reply.

However, it is also used to deliver error notifications, such as Destination Unreachable, which should also be handled by the application, in order to untrack filtered connections, such as when a UDP packet is sent to a port number, where nothing is listening. Listing 4.3 shows an ICMP error notification, when a UDP packet is delivered to port

10000, and nothing is listening for it. The listing also shows that the entire IP packet, including the UDP header and payload was returned to the sender.

```
Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 3 (Port unreachable)
    Checksum: 0x394c [correct]
    Unused: 00000000
    Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
        (...Truncated...)
        Protocol: UDP (17)
        Header checksum: 0x78e2 [validation disabled]
        Source: 127.0.0.1
        Destination: 127.0.0.1
    User Datagram Protocol, Src Port: 53163, Dst Port: 10000
        Source Port: 53163
        Destination Port: 10000
        Length: 13
        Checksum: 0xfe20 [unverified]
    Data (5 bytes)
        Data: 616263640a
        [Length: 5]
```

Listing 4.3: ICMP Port Unreachable message.

4.6.4 Layer 7

On layer 7, the Sniffer handles decoding of application layer data. By inspecting the payloads of TCP or UDP packets, it is possible to distinguish between protocols that are supported by the applications.

In the application, it should be possible to extend this layer with numerous protocols, since new application layer protocols are created every once in a while. On the lower layers, new protocols do not suddenly emerge, since they often require operating system level support, which is not the case with application layer protocols.

The application should support two general cases, text-based protocols and binary protocols. The definition of a text-based protocol, is one that uses only printable American Standard Code for Information Interchange (ASCII) characters for normal operations. Binary protocols on the other hand, do not consist of only printable characters, but may include any byte value from 0–255.

Text-based Protocols

In general, text-based protocols should be easier to distinguish, since there is more information transmitted, and therefore less ambiguity. Examples of text-based and binary protocols are shown in Listing 4.4 and Figure 4.13.

In the HTTP example, the application can look for the “GET / HTTP/1.1” line, followed by a list of key-value pairs (separated by colon), until it finds a double line feed.

```
GET / HTTP/1.1
Host: example.com
User-Agent: curl/7.54.0
Accept: */*
```

Listing 4.4: An example HTTP request.

Binary Protocols

However, in the DNS example, the DNS header (Figure 4.13) is shown, which is followed by the questions for the DNS server. In that case, there is virtually no consistent information to look for (as with "GET / HTTP/1.1"), the first two bytes is an ID that can contain any value between 0–65535.

The ID is virtually useless, except it is used to match the response packet with, but it does not provide a definitive clue that it is a DNS query. After the ID, certain flags are sent, depending on the query, which provides some basis for matching, and then four 16-bit numbers are sent, which indicate the number of domain names to look up. The application can reasonably assume that these numbers are close to zero, since a query often looks up a single name at a time.

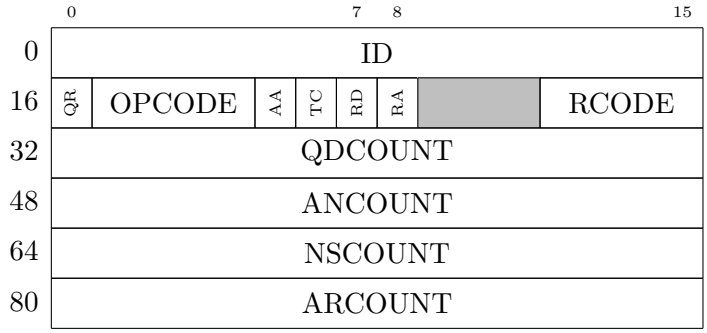

Figure 4.13: The DNS header.

4.6.5 Reports

The previous 4 subsections, Sections 4.6.1 to 4.6.4, have described how traffic should be analysed on the different layers of the OSI model. As stated initially in this section, whenever a layer detects that which it is looking for, it can produce a report that is used by the Traffic Profiler.

In Section 4.2, a list of information that a profile must include is shown. To create a profile containing that information, the following reports are necessary:

DHCP Acknowledgement Report

Whenever a device has been powered on, and it is configured to retrieve an IP address automatically (through DHCP), it sends out a DHCP request, in order to configure its own IP address. The DHCP server on the network responds to the request, and provide an IP address to the device, which sends the final message in the DHCP protocol, the acknowledgement. This acknowledgement contains the MAC address of the device, the given IP address, and a host name, if the device is configured to provide it.

This report covers the first three points on the list in Section 4.2.

DNS Query Report

When a client wants to connect to server, through a domain name, it must first resolve the IP address pointed to by the domain name. This is done through the DNS protocol. The DNS query, and the response to it, contain the domain name of the query and the resolved IP.

This report covers the 4th point on the list in Section 4.2.

New Connection Report

This report should be generated whenever a new TCP connection is established, or a new packet has arrived on an untracked TCP 4-tuple.

This report covers the 5th and 6th point on the list in Section 4.2, but only to the level that IP addresses and port numbers are known. In order to determine which service is running, DPI can be used to generate reports from the traffic after the connections are established. This is described in the next report.

Application Layer Protocol Report

After a connection has been established, only the involved IP addresses and ports are known. In order to determine which protocol is used in a connection, it is necessary to use DPI to look at the payloads.

By looking for specific patterns in payloads, the application layer protocols can be determined. When the protocol has been determined, a report should be generated. In certain cases, it is possible to gain information about the software running on either end. If this can be determined, it should also be included in the report.

4.6.6 Summary

This section has described the design choices regarding the Sniffer part of the application. Sections 4.6.1 to 4.6.4, have described how traffic is analysed on different layers, and Section 4.6.5 has described what reports are necessary to generate the profiles that are described in Section 4.2.

In the next section, the Profiler subsystem is described, which uses the reports that are produced by the Sniffer subsystem.

4.7 Traffic Profiler

The previous section, Section 4.6, describes the subsystem that analyses network traffic, and generates reports that can be used to build device profiles. This section describes the design of the Traffic Profiler subsystem, which combines multiple reports of the same device into a profile. The end goal is to produce profiles, such as specified in Section 4.2, for each device that is connected to the network.

The Traffic Profiler subscribes to reports generated by the Sniffer. Each time a report is received, the Traffic Profiler should check if the device is currently being profiled. If it is, the information of the report is added to the existing profile. If it is not, a new profile generation is started for the new device.

Internally, the Traffic Profiler must treat each device independently. Reports that are received regarding one device, must not have any side effects on other developing profiles. The rationale is that the devices are independent, and the profiles must accurately represent one device.

The flowchart on Figure 4.14 shows the process of the Traffic Profiler subsystem. The horizontal lines represent the boundaries of the different subsystems, where to Management subsystem is on the top, the Sniffer on the bottom, and the Traffic Profiler is in the middle. The Sniffer generates reports, which are processed by the Traffic Profiler. The data of the reports are stored into a profile storage, which contains developing profiles.

The Management subsystem accesses the profile storage, in order to extract profiles and share them with the Server Application, and perform other management features of the Client Application. The connection to the profile storage should be in a real time manner such that the Management subsystem can get notified about profile changes.

On Figure 4.15, an example is shown where a profile is updated due to incoming reports. The first report “(1)” is the result of a DNS query, where the Tado device is resolving “i.my.tado.com”, and the response is an A record, containing the two IPs “54.171.136.152” and “52.48.170.120”.

The second report “(2)” shows the TCP connection initiated to one of those two IPs on port 443, where the profiler should substitute the IP for domain name, since it has learned that from the first report.

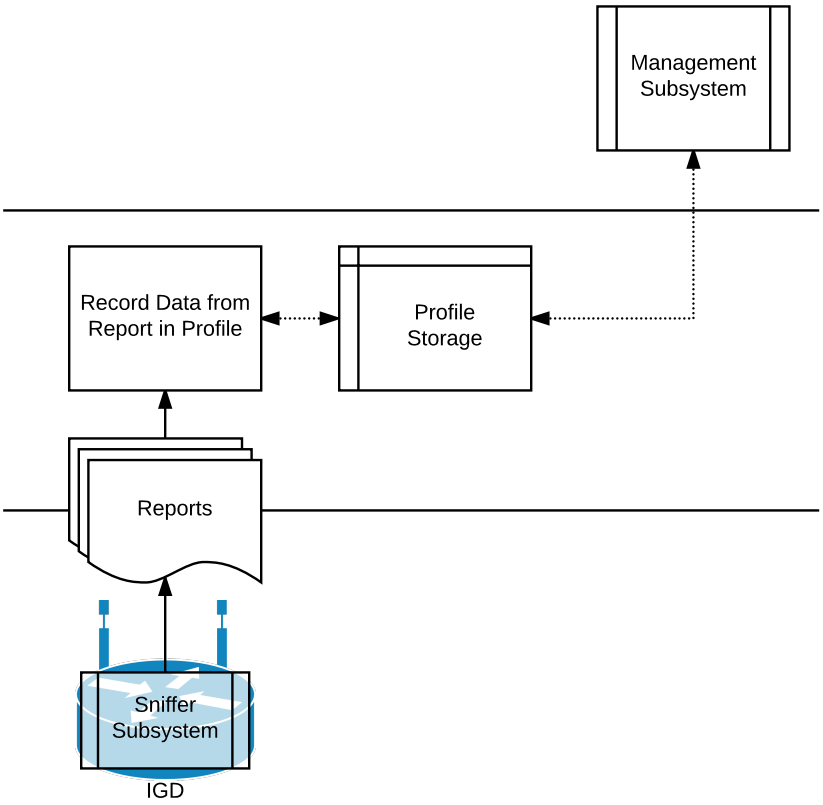

Figure 4.14: Flow chart of profiler design.

4.8 Management Subsystem

The Management subsystem is responsible for managing the process of generating profiles. It is also the part of the Client Application, which communicates with the Server Application. The communication to the Server Application is already established to be a RESTful Web Service.

Since the subsystem is responsible for managing profile generation, it must have access to the profiles that are in generation. This should happen through a real-time connection, where changes in profiles are pushed to the Management subsystem, instead of repeatedly retrieving all the profiles. This behaviour is more efficient, and the Management subsystem can react faster than if it is polling.

Figures 4.16 and 4.17 show the interaction between these two subsystems.

Following the sequence on Figure 4.17, the Profiler automatically starts profiling new devices that are connected. Every time a change is made in a profile, the Management subsystem is notified, and it can decide how to proceed. When a device is connected for the first time, the Management system queries the Server Application, after receiving initial data about the device. Such as MAC address and host name.

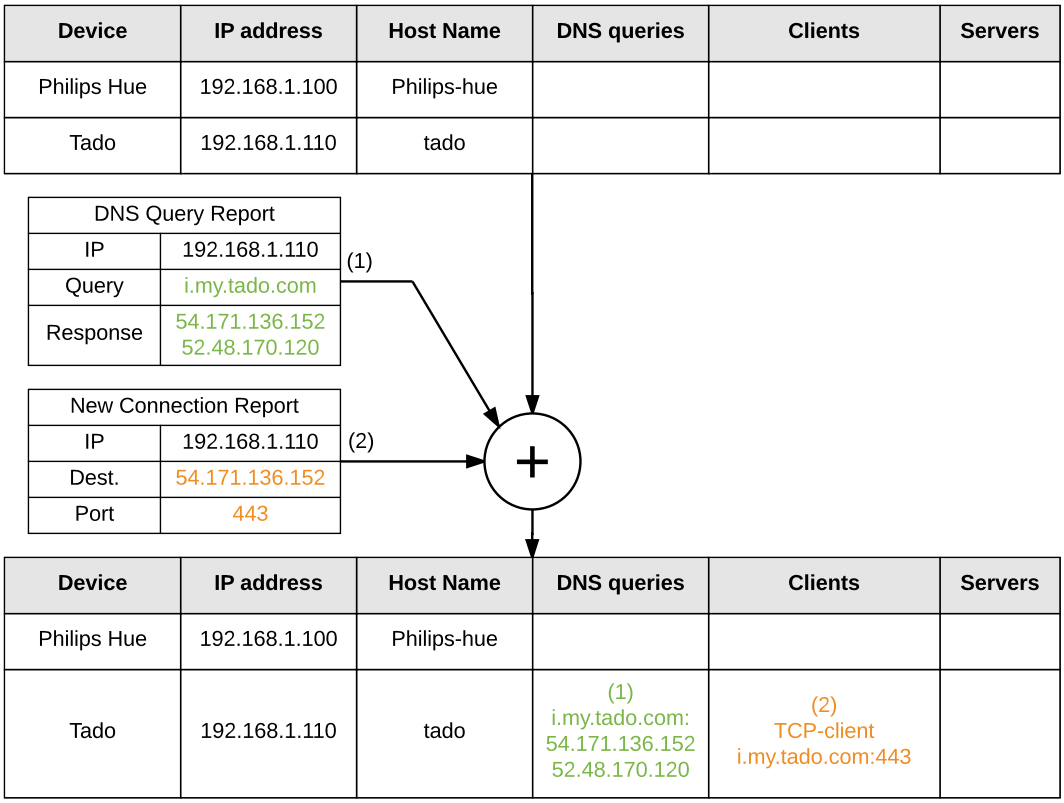

Figure 4.15: Example of how a profile is updated when reports are processed.

If a profile is found on the Server Application, it is downloaded and the Management subsystem configures the IGD according to the profile. Profile generation is also stopped on the Profiler, as the only traffic that is detected, is already permitted by the profile.

If a profile is not found, the Profiler continues building a profile, as explained in Section 4.7. When the profile is considered complete, the Management uploads the profile to the Server Application, and configures the IGD's firewall according to the generated profile. Determining when a profile is complete, e.g. represents a device completely, can be difficult, as there could be periodic traffic, which does not occur while training the profile. But a simple way is to consider the profile complete after a period of time where there has been no changes in the profile.

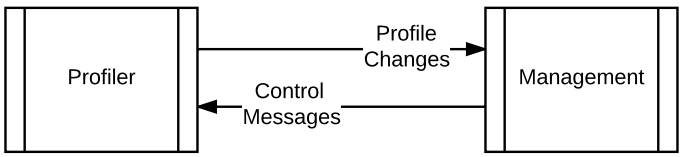

Figure 4.16: The interaction between the Profiler and the Management subsystem.

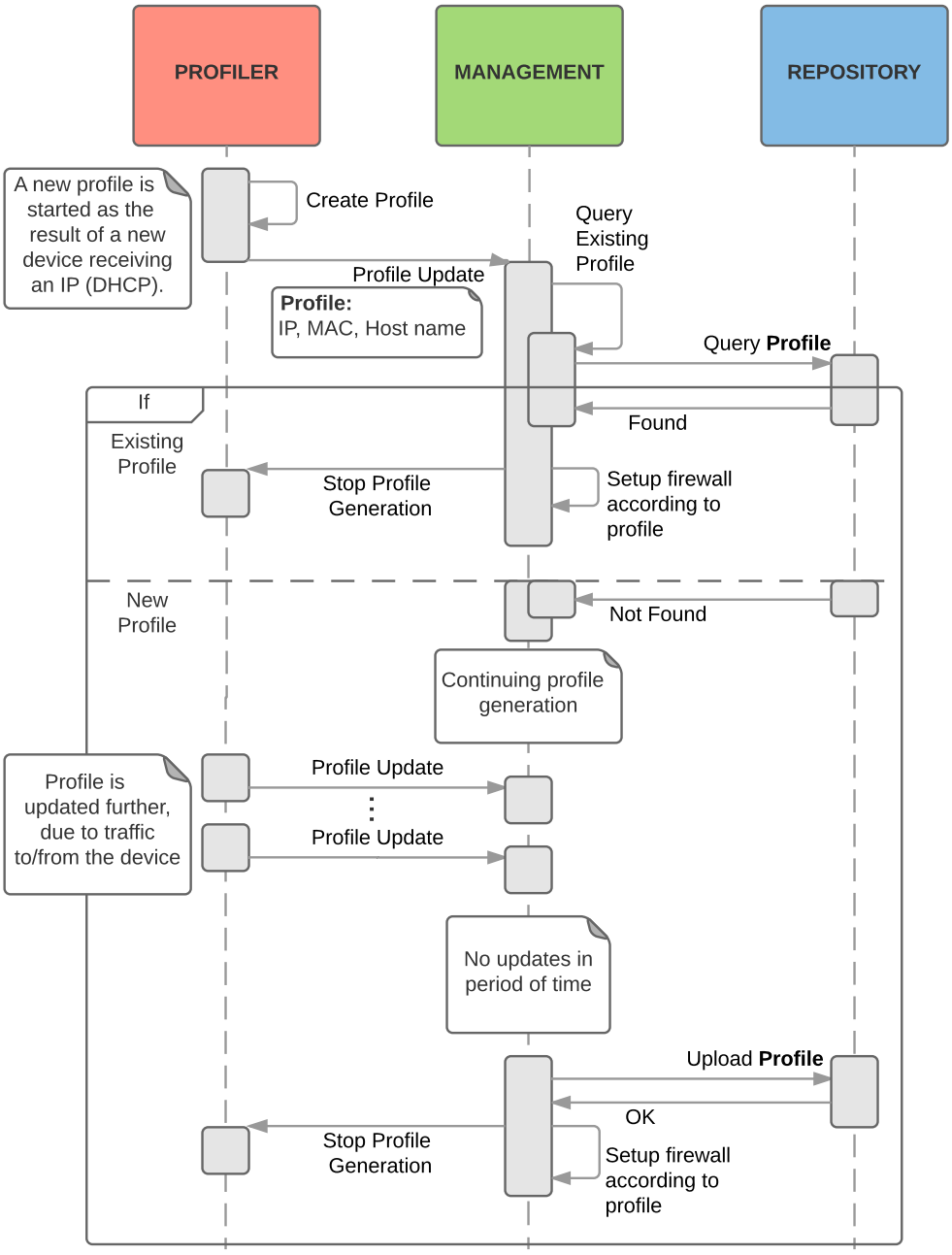

Figure 4.17: Sequence diagram showing how the Management subsystem and Profiler cooperate.

When a profile is completed, or downloaded, it should be added to the firewall on the IGD. In this case, the profile should be transformed into firewall rules that are applicable on the IGD. The transformation is dependent on the firewall software, which depends on the operating system of the IGD. Therefore, the process of transforming a profile to firewall rules, has to been done on an implementational level, and there is not a specific design to it.

4.8.1 Summary

This section has described the design of the Management subsystem in the Client Application. The subsystem manages the Profiler application, and makes decision about when a profile is complete. It also interacts with the Server Application, in order to find an existing profile or upload a profile that has been generated. Finally, the subsystem handles configuration of the IGD firewall, when a profile should be added to the firewall configuration.

CHAPTER

5

IMPLEMENTATION

This chapter describes the implementation of the design in Chapter 4. As the software is only a PoC, it implements a small amount of protocols. The implementation is programmed in Python, as it has the required functionality to build the application, and there are existing libraries that can be used for packet sniffing.

The implementation is, as specified in the design, split into three subsystems. Some of these subsystems use some common software patterns, namely dependency injection and event busses. These patterns are initially described in Sections 5.1 and 5.2. Then the subsystems are described individually.

The first subsystem is the PySniffer module. This module captures network traffic and analyses it. Based on this analysis, reports are generated and dispatched to the next subsystem, PyProfiler.

PyProfiler generates a profile for each device based on the information in the reports it receives. Finally, based on the generated profiles, a set of firewall rules is generated by PyProfiler. The names PySniffer and PyProfiler represent the traditional Python way to name libraries, e.g. `py*`.

5.1 Dependency Injection (DI)

In Section 4.5.4, the Client Application is separated into three subsystems. The three subsystems can then be verified independently, before they are integrated into the complete system. In order to verify the subsystems, Dependency Injection (DI) is used to separate components within the subsystems.

DI is a technique, where a component supplies a dependency to another component, by injecting it into the dependent component. This is opposed to the dependent component having a tightly coupled connection to the dependency. DI is shown on Figure 5.1, and the tightly coupled case is shown on Figure 5.2. On both figures, the `Car`

class has a method called `setVehicleSpeed`, which in this example, should call the `setRotationSpeed` on the `engine`, by multiplying the speed in km/h by a constant to determine the engine speed in RPM.

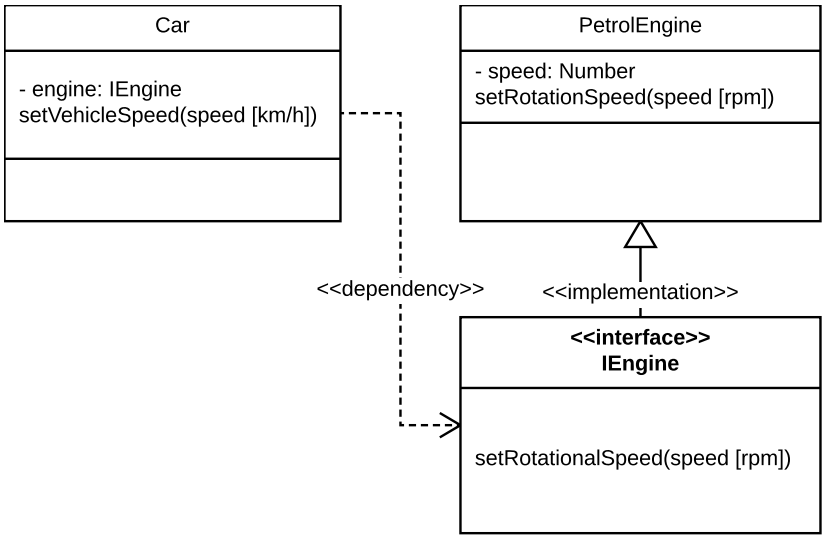

Figure 5.1: Car example using Dependency Injection (DI).

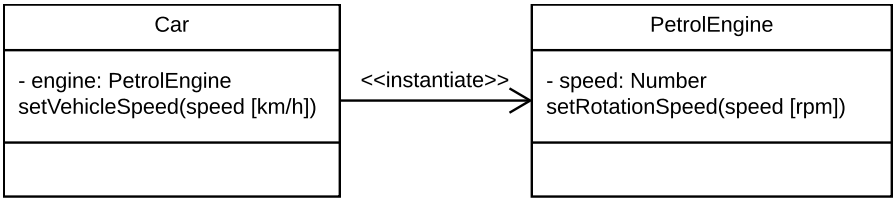

Figure 5.2: Car example without Dependency Injection (DI).

On Figure 5.2, the `Car` object autonomously instantiates a `PetrolEngine`. In order to write a unit test, which verifies that setting the vehicle speed accurately sets the rotation speed of the engine, the test must create a `Car` object. Creating a `Car` object, results in the instantiation of a `PetrolEngine` object. The unit test then has to measure the output of the `PetrolEngine`, and the unit test also fails, if it is the engine that is broken, but not the car itself.

On Figure 5.1, the `Car` instead depends on any type of engine (`IEngine`). Therefore, a unit test can create a fake engine, which follows the interface, and pass the fake engine to the `Car`. By this abstraction, the unit test is isolated to the `Car` class, which can be verified without requiring a functioning `PetrolEngine` class.

5.1.1 Summary

In this section, the DI pattern has been described. By writing the application code following this pattern, the components become more loosely coupled. This is beneficial, since it is simpler to verify that the components are working as intended.

Another benefit is that, in order to follow the pattern, the code must have a higher level of abstraction. Since the components may only depend on the availability of a component following the specification. This, in turn, should make it simpler to exchange one implementation of a component with another. For example, to select between two components that recognise the same protocol.

In the next section, another concept is explained, which also assists in separating the different components.

5.2 Events

Along with DI, another technique is used to separate the different components in the subsystems, the "publish–subscribe" pattern. This pattern specifies that the interactions between components are published by the sender component, and received by any component that subscribes to them. There can then be multiple receiver or none at all.

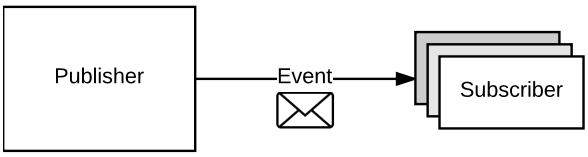

Figure 5.3: Publisher sending event to multiple subscribers.

By performing this inversion of control, the components become loosely coupled, as they interact through generic messages, instead of function calls. It also simplifies the way of creating one-to-many relationships between components, where one sender interacts with many receivers. Since the sender can be oblivious to who receives the messages that it sends. By extension, this makes it simpler to create multiple components that receive messages from the same sender.

This pattern is useful in the Sniffer subsystem, where there is a layered architecture. For example, the IPv4 module subscribes to packets from the Ethernet module, but the Ethernet module does not know about the IPv4 module. Similarly, the IPv6 module can also subscribe to packets from the Ethernet module.

Listing 5.1 shows a simple implementation of the pattern, where a publisher can emit an event to multiple subscribers. The `subscribe` method should be passed a callback function, which is subscribed to the event. When the publisher wants to send a message to the subscribers, the message is passed to the `publish` method.

```
class Event:
    def __init__(self):
        self.subscribers = set()

    def subscribe(self, callback):
        self.subscribers.append(callback)

    def publish(self, message):
        for subscriber in self.subscribers:
            subscriber(message)
```

Listing 5.1: Event emitter.

5.2.1 Summary

In this section, the "publish-subscribe" pattern has been described. This pattern simplifies the process of passing data from one component to multiple components, without a predefined list of receivers. It can used to extend the subsystem with new components that support new protocols.

5.3 PySniffer

PySniffer is the module that captures all the traffic passing through the IGD. It performs DPI on the captured packets, in order to generate reports, describing which protocols are used, and the endpoints. To do so, a Python library called Scapy is used, the usage of Scapy is explained in Section 5.3.1.

The analysis of the Internet packets is done using a plugin system, where plugins can emit events, as described in Section 5.2, when they find packets following their matching criteria. These plugins operate on different layers in the OSI model, specifically layers 3, 4 and 7.

5.3.1 Scapy

Scapy is a Python library that can analyse and manipulate network traffic. It can both send and receive packets. In this project, it is used for sniffing packets on the IGD. After a packet is received, Scapy decodes the packet. For example, a DNS header contains many values, which are at a specific offset in the UDP payload, see Figure 4.13.

Scapy has mappings of the offsets, such that each value can be retrieved as shown below:

```
ancount = packet['DNS'].ancount
```

This returns the `ancount` value from a DNS packet, which is the number of answers in a response. If a value is required from a different layer, it can be accessed similarly, by exchanging "DNS" with e.g. "IP" or "TCP".

Without Scapy, these offsets must be specified manually, in order to decode the packets. An example hereof is found below:

```
ancount = packet[x + y + z + q]
```

In this example, `x`, `y` and `z` are the lengths of Ethernet, IP and UDP headers respectively and `q` is the offset of `ancount` in the DNS header.

5.3.2 Reports

Reports are used as an interface between PySniffer and PyProfiler. PyProfiler reads the reports and uses them to generate a profile for each device, based on these reports. The reports are generated at layer 4 and layer 7, in the PySniffer software.

Layer 4 reports describe which transport protocols are used for the device, hence it contains source and destination IP and ports, along with the protocol, either TCP or UDP.

Layer 7 reports contain information about which application layer protocol is in use, and what software and version it is using. This could be an IoT device connecting to a HTTP server, using curl version 7.54.0. The report would then contain the IPs of the client and server, along with curl 7.54.0, since it is the HTTP client and version, used by the IoT device. An example of a report is found on Figure 5.4.

HTTP Report	
Client IP	192.168.1.110
Server IP	54.76.229.21
User-Agent	curl/7.54.0

Figure 5.4: HTTP report, with user agent “curl”.

If, on the other hand, it was a DNS report, it would contain the source and destination IPs and the domain name queried, together with the DNS responses. Such a report is found on Figure 4.4.

5.3.3 Layer 3

The plugins on layer 3 divide the Ethernet traffic into IPv4 and IPv6 traffic. When using Scapy, it can be done by checking if the packet contains an `IP` or `IPv6` layer.

If a packet is either an IPv4 or an IPv6 packet, an event is emitted, and the plugins on the higher layers receive it if they are subscribing to the event.

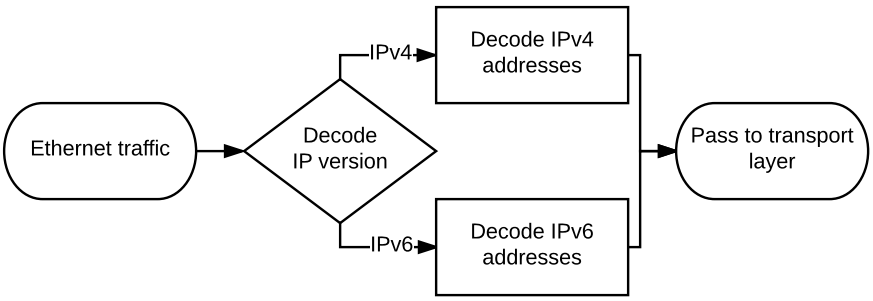

Figure 5.5: Flowchart of layer 3 plugin separation.

5.3.4 Layer 4

On the transport layer, two plugins are created, to perform TCP and UDP connection tracking.

UDP

This plugin tracks all UDP connections, a flowchart is shown on Figure 5.6. The plugin subscribes to packets, from both the IPv4 module and the IPv6 module.

When a packet is received, the protocol number is decoded. If the protocol is UDP, the source and destination IPs and ports are decoded. If this 4-tuple is tracked, the packet is recorded on the existing tracked connection. On the other hand, if it is not tracked, the 4-tuple is saved to a list with tracked connections. A connection is tracked, if the 4-tuple is already saved, and the 4-tuple has been used within the last 300 seconds which is the default UDP tracking time for Linux.

If the received packet instead uses the ICMP protocol, then the ICMP type of the packet is decoded. If the ICMP type is “Destination Unreachable”, a report is created stating that a device has tried to access a closed port.

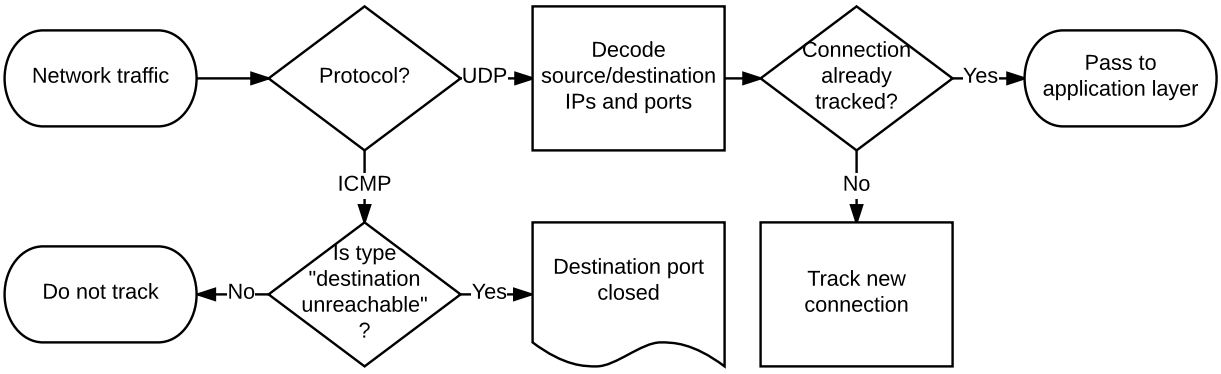

Figure 5.6: Flowchart of the UDP plugin.

TCP

This plugin tracks TCP connections. If a new TCP connection is found, this layer keeps track of the connection, using the sequence and acknowledge numbers defined in the TCP protocol. A new tracked connection generates an event for other plugins on higher layers, which they can subscribe to, and receive the payload without keeping track of the specifics in the TCP protocol, such as the Three-way handshake, acknowledgement and retransmission.

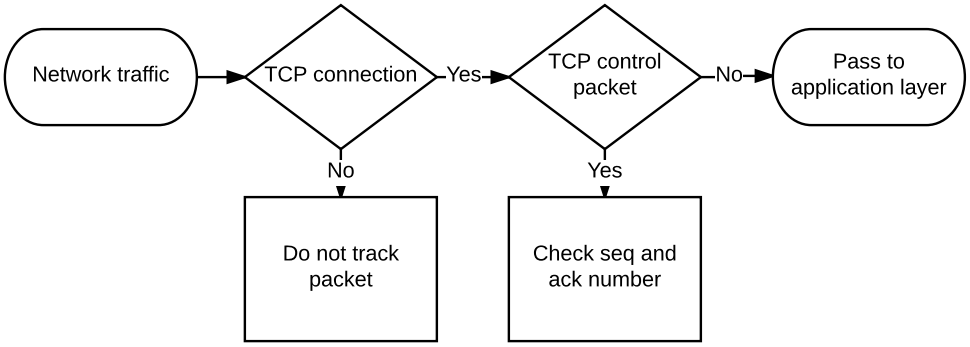

Figure 5.7: Flowchart of the TCP plugin.

5.3.5 Layer 7

On the application layer, packets are already tracked on a connection level, meaning that each plugin can subscribe to new connections being established.

When a new connection is established, the plugins can subscribe to receive traffic on the connections. Thereby the plugins can operate using a state machine, where a plugin can look for multiple packets arriving in a certain order.

If a packet does not match the current state of the state machine, the plugin then unsubscribes from the connection, as it has determined that the application layer protocol is not the one that is detected by this plugin. Similarly, if the state machine reaches its final state, the plugin can generate a report, which contains information about the protocol in use.

Several plugins have been implemented, and these plugins are described below.

DHCP

The DHCP plugin subscribes to all new UDP connections, if a DHCP request packet is found, it waits for the DHCP ACK packet.

A report is generated based on the DHCP ACK packet, with the IP address, host name and MAC address of the IoT device which sent the DHCP request.

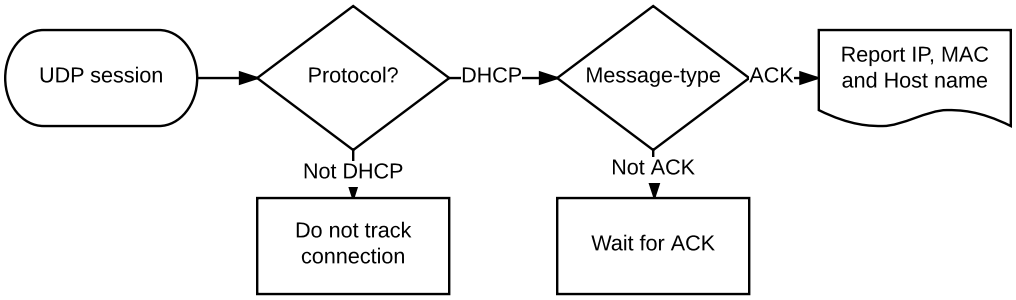

Figure 5.8: Flowchart of the DHCP plugin.

DNS

The DNS plugin subscribes to the UDP plugin, to receive all new UDP connections.

Each DNS session has an ID, which is random generated, this ID is decoded by the plugin when the first DNS packet is received, the query. After the DNS query is received and decoded, the plugin waits for the DNS response. When the response is received a report is generated, with the domain name, IP of IoT device, and the IP in the response. If the response has multiple IPs, they are all included in the report.

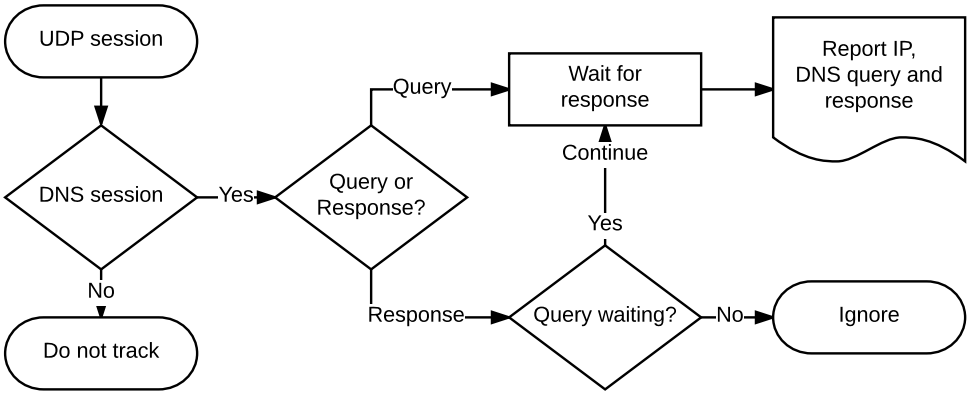

Figure 5.9: Flowchart of the DNS plugin.

SSL

This plugin subscribes to a TCP connection. In order to determine if a connection uses SSL, the payloads are examined for messages of the SSL protocol. The SSL handshake protocol begins messages with the byte `0x16`, which can be used to track SSL messages. When an SSL session is found from the analysis, a report is generated.

By extending the SSL module, it is possible to extract the server certificate, and the cipher suite that is used for the connection. This can then be included in the report.

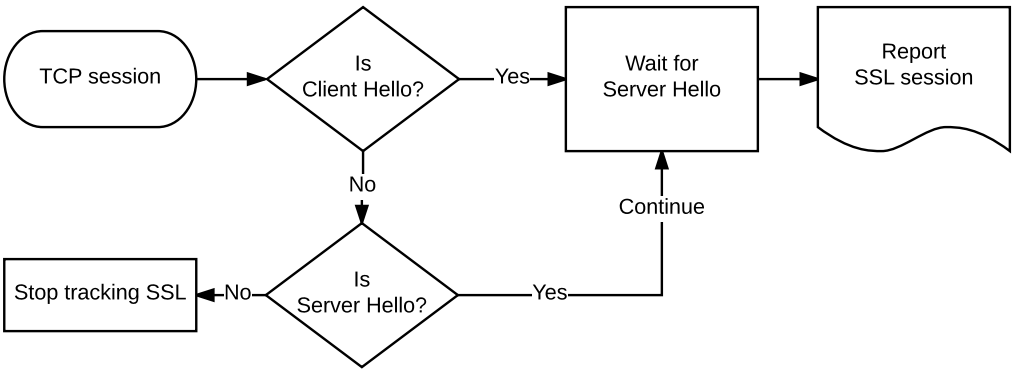

Figure 5.10: SSL plugin flowchart.

Telnet

Telnet subscribes to the TCP plugin. When a new TCP connection is established, the Telnet plugin looks for Telnet commands. The Telnet commands are structured as follows:

```
Command start byte + Command byte + Value byte
```

The `Command start byte` is an identifier, which always has the value 255, and the `Command byte` is the actual command, which has a value between 240 and 255.

Multiple Telnet commands can be sent in succession.

If this criteria is met, the plugin waits for a response from the server, to verify that a server is running, and a report is generated.

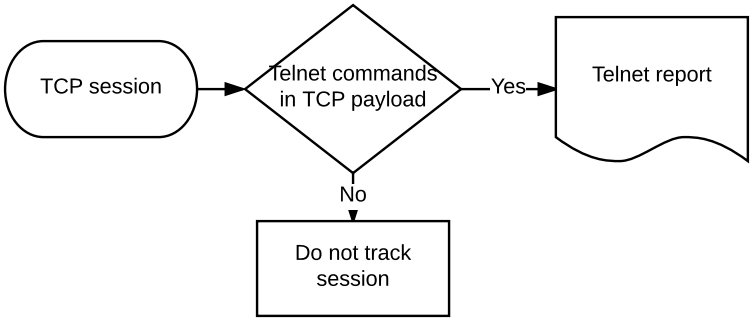

Figure 5.11: Telnet plugin flowchart.

Regular Expression (RE)

The Regular Expression (RE) plugin checks protocols that are text-based and use patterns which can be matched using RE. In this case, it is used for SSH and HTTP.

For example, in order to identify the SSH protocol. The string the RE plugin matches against could look like `SSH-2.0-OpenSSH_7.5`. The RE used to match that string looks like:

```
^SSH-(?P<software>.+?)\r?$
```

This RE searches for "SSH-" in the beginning of a line. It captures the following text until it meets a "Carriage Return" (`\r`), and returns it in a variable called "software".

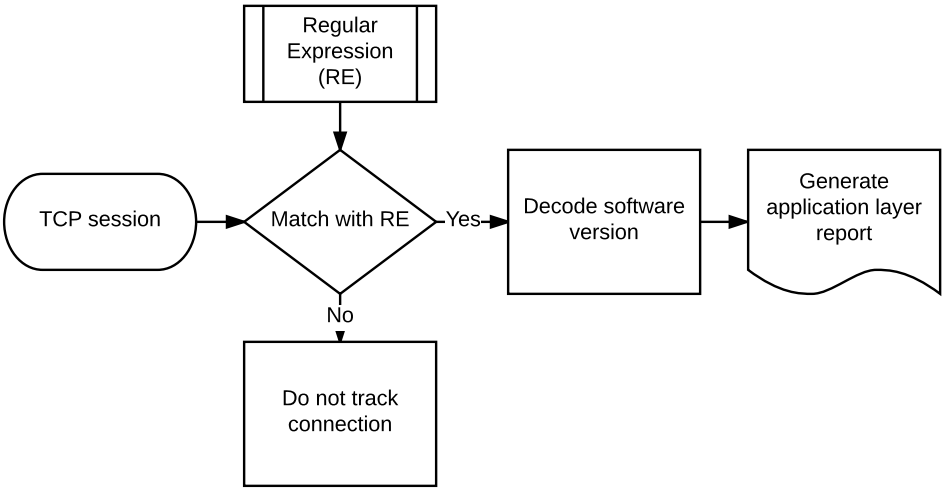

Figure 5.12: TextMatch plugin flowchart.

5.3.6 Summary

In this section, the implementation of PySniffer has been explained. It is described how the implementation captures and analyses packets, and how the reports are generated. The next section is going to describe the implementation of PyProfiler, which uses the reports that are generated by PySniffer.

5.4 PyProfiler

This submodule, PyProfiler, processes reports generated in PySniffer. Based on these reports, a profile is generated for each IoT device. From this profile, a set of firewall rules is generated. Though the "management subsystem" is not implemented, due to limited time. Instead a script is made, which takes a profile as input and returns a set of firewall rules for the device, which should manually be loaded into the firewall. This script serves as a PoC of the ability to generate firewall rules from the profiles that are generated.

5.4.1 Profiles

Based on reports generated by PySniffer, described in Section 5.3.2, a profile is generated. This profile contains everything reported about each IoT device.

Table 5.1 shows a representation of an example profile. The profile contains a DNS query to resolve *updates.example.com*, where firmware updates could be retrieved. The DNS query leads to an HTTP request, which is shown in the clients section. The device could then be managed through an HTTPS Application Programmable Interface (API) on port 443, shown in the servers section.

Profile				
Name	ExampleDevice			
Servers	**Protocol**	**Port**	**Service**	
	TCP	443	SSL	
Clients	**Protocol**	**Destination**	**Port**	**Service**
	TCP	203.0.113.65	80	HTTP
DNS queries	**Query**		**Response**	
	updates.example.com		203.0.113.65	
			203.0.113.66	

Table 5.1: Example of a profile.

5.4.2 Web Server

While developing the application, it is necessary to look into the generated profiles in real time. Therefore a simple web page has been created to provide a real time overview of the profiles. This web page uses the real time connection that the Management subsystem is anticipated to use.

The real time connection sends out notifications when there is a change to the internal data structure that contains the profiles in PyProfiler. By encoding these notifications using Javascript Object Notation (JSON) and sending them over a WebSocket connection, they can be interpreted by an HTML and Javascript web page. A WebSocket connection is substituted for a regular socket, since it can be opened directly from a web page, therefore receiving the notifications directly from PyProfiler.

The web page is shown on Figure 5.13.

5.4.3 Firewall Rules

After a profile has been generated, it can be converted to firewall rules that can protect the device. In this project, the firewall is running on Linux and therefore using the netfilter framework, which is configured through the iptables tool.

Figure 5.13: Screenshot of the web page that shows real time profiles.

The profiles contain three sections as seen in Table 5.1. From the servers section, incoming traffic can be allowed on the recorded ports. For outgoing traffic, listed in the clients section, traffic can be allowed to the specific IPs and ports. However, since outgoing traffic uses DNS to resolve the IP address, the domain name should be taken into consideration. Since the IP addresses that the DNS queries resolve to can change over time.

Incoming Traffic

Incoming traffic can be allowed through a single iptables rule:

```
iptables -A FORWARD -p tcp -d 192.168.0.100 --dport 443 -j ACCEPT
```

Where `tcp` can be exchanged for `udp`, if that is the case. And `192.168.0.100` and `443` represent the device's IP address and service port.

Outgoing Traffic

Outgoing traffic that is going directly to an IP address, which is not resolved by DNS, can be allowed with a single rule:

```
iptables -A FORWARD -p tcp -s 192.168.0.100 -d 203.0.113.44 --dport 80
                                -j ACCEPT
```

Where `tcp` can be exchanged for `udp`, if that is the case. And `192.168.0.100`, `203.0.113.44` and `80` represent the device's IP address, the Internet server's IP address and service port respectively.

However, if the traffic is going to a domain name, which has been recorded in the profile. Then the rule should show that. Iptables does not support domain names dynamically, they are resolved statically when the rule is added. Therefore, the two tools dnsmasq and ipset can be used together to provide a dynamic list of IP addresses that the rule matches.

Dnsmasq is a DNS resolver and ipset is an in-memory dynamic list of IP addresses, which can be referenced by iptables. Dnsmasq has an option to add resolved IP addresses

of specific domain names to an ipset set. This can be accomplished by the following configuration line in dnsmasq:

```
ipset=/updates.example.com/updates_example_com
```

Every time dnsmasq resolves `updates.example.com`, the corresponding IP address is added to the ipset set named `updates_example_com`.

Subsequently, traffic to `updates.example.com` can be allowed by the following rule:

```
iptables -A FORWARD -p tcp -s 192.168.0.100 -m set --match-set
         updates_example_com dst --dport 80 -j ACCEPT
```

Where the `--match-set` option specifies that the destination IP address should be looked up in the `updates_example_com` ipset set.

5.4.4 Summary

In this section, the PyProfiler subsystem has been explained. This subsystem interprets the reports that are generated by the PySniffer subsystem, and use them to create profiles. These profiles can be viewed in real time through a web page, and after the learning phase, they can be used to generate firewall rules.

These firewall rules can then be installed in the IGD, where they only permits traffic that matches the profile.

In the next chapter, the application is tested, in order to verify if the metrics in Section 2.11 are fulfilled.

CHAPTER

6

SYSTEM EVALUATION

This chapter documents the evaluation of the metrics described in Section 2.11.

Initially the test setup is described, then the tests are described.

Profile generation is evaluated in the first test. The second test evaluates the behaviour of the devices after the firewall rules are added. The final test is a port scan, which shows if any of the learned ports are closed, after the firewall rules are applied.

After each test, the results are presented and evaluated against the metrics in Section 2.11.

6.1 Test Setup

The test setup consists of a computer which acts as DHCP and NAT. This computer is connected to the Internet, and to two Access Points (APs). One AP for IoT devices, and one for MPD devices. Each AP is connected to different subnets, the IoT AP is connected to subnet `192.168.1.0/24`, and the MPD AP is connected to subnet `192.168.0.0/24`. The complete setup is found on Figure 6.1.

Every IoT device gets an IP addresses in the 192.168.1.0/24 subnet and the MPDs get IP addresses in the 192.168.0.0/24 subnet.

The following devices are used in the test.

- Philips Hue.
- Tado thermostats.
- DEVELCO Squid.link.
- Virtual machine with exploit.

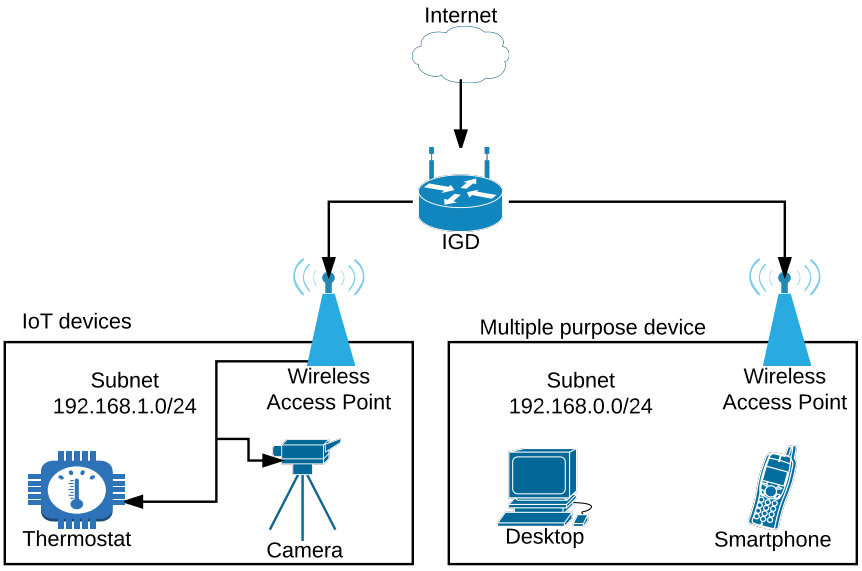

Figure 6.1: The system setup in the test.

A short description of Philips Hue, Tado, and DEVELCO Squid.link is found in Appendix A. The Virtual Machine is described in Appendix B. The Virtual Machine mimics an IoT device, this is running a service which is exploitable, the exploit is used after the firewall rules are generated, and it is tested if the Virtual Machine is able to make an outgoing connection, in order to spread malware to other devices, or generate a DoS attack. The Virtual Machine is only used in the profile learning phase and in the exploit test.

With the setup in place, the profile generation is explained in the next section.

6.2 Profile generation

This section describes the test of the profile generation.

Initially the procedure is described, followed by the generated profiles which are presented and discussed.

After the profiles are presented, the firewall rules are shown and discussed, and finally a port scan of the devices, before and after the firewall rules are applied.

6.2.1 Procedure

First off, the computer is connected to the Internet, and the two APs are connected to the computer. After the computer is connected to the Internet, it configures the two

subnets, and starts the PySniffer and PyProfiler. When the software is running, the computer is connected to the two APs.

Afterwards, a phone and a second computer are connected to the MPD AP, these are used to configure the IoT devices.

When the phone and the computer are connected to the MPD subnet, the IoT devices are connected the IoT AP. Each IoT device is configured from the factory settings. When all the devices are fully configured, the profiling is stopped and the firewall rules are generated.

6.2.2 Execution

All the devices were connected, and the profiler generated the profiles. A profile from the web interface for Tado can be seen on Figure 6.2. The total time to make the profiles was 17 minutes, that included registering profiles for Philips Hue and Tado. When the setup of all the devices were done, the profiler was turned off, and the firewall rules were generated.

6.2.3 Generated Profiles

On Figure 6.2, a profile from the web interface can be seen. Similarly in Appendix C, the profiles for Philips Hue, DEVELCO Squid.link and the virtual machine can be seen. From these profiles, it is seen which services are used by the IoT devices, and which services the IoT devices hosts. On Figure 6.2 it can be seen that Tado does not host any services, and it only uses two remote services. One of the remote services is the DNS server hosted on the IGD. The other service is a management service hosted by the company Tado.

Figure 6.2: The profile shown in the web interface.

The firewall rules generated from this profile are shown below on Listings 6.1 and 6.2.

```
Chain IN_192.168.1.100 (1 references)
target prot source    Destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0 state RELATED,ESTABLISHED
ACCEPT icmp 0.0.0.0/0 0.0.0.0/0
LOG    all  0.0.0.0/0 0.0.0.0/0 LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing 6.1: Ingoing firewall rules for Tado.

From the rules on Listing 6.1, it can be seen that it accepts ingoing connections, which are established by the device. Furthermore, all ICMP traffic is allowed, this choice is made as some users might find it useful to ping their device to check if the device is online.

Listing 6.2 shows the outgoing traffic rules.

```
Chain OUT_192.168.1.100 (1 references)
target prot source    destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0   state RELATED,ESTABLISHED
ACCEPT udp  0.0.0.0/0 192.168.1.1 udp dpt:53
ACCEPT tcp  0.0.0.0/0 0.0.0.0/0   match-set set_bd1123a0 dst tcp dpt:443
LOG    all  0.0.0.0/0 0.0.0.0/0   LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing 6.2: Firewall rules for Tado outgoing traffic.

On Listing 6.2, related connections are allowed similarly to the ingoing rules. The second rule permits traffic to the IGD's DNS server, however this rule has no effect, since traffic going to the IGD does not pass through the FORWARD chain. The third rules uses an ipset set, `match-set set_bd1123a0`, which is a set of IPs addresses. These addresses come from the response of a DNS lookup. The ipset set is seen on Listing 6.3.

```
Name: set_bd1123a0
...
52.212.221.252
54.72.3.36
```

Listing 6.3: Ipset set for domain name i.my.tado.com.

In this case, only two IPs addresses are permitted by the rule.

This tests concludes that metric Section 2.11.3 can be fulfilled, but the user manually has to run a script, it still requires a little user interaction.

With the firewall in place, the next section describes the tests of the devices with the firewall rules implemented.

6.3 Firewall

This section describes the test of the devices after the firewall is setup. This includes 3 separate tests. In all the tests, the firewall is logging if any packets are dropped according to the firewall rules.

In the first test, the IoT devices are used, and all the different functions are tested. The behaviour of the devices are examined and it is checked if they behave as expected. Furthermore, the firewall log is checked if any packets have been dropped. It is expected that no packets are dropped, and the devices function properly.

In the second test, the Virtual Machine is exploited, and a reverse shell is sought to be established to the attacker's machine. In this test, it is expected that the firewall drops the traffic and the packets should be logged by the firewall.

The third test is a test running for 24 hours. After the test, it is be investigated if any packets are dropped by the firewall. If the system works as expected, no packets should be dropped.

6.3.1 Functionality Test

The functionality test is made on the 3 IoT devices, and the Virtual Machine, to make sure they respond as intended. Throughout the test, it was realised that there were some dropped packets on the DEVELCO Squid.link device, not that it had any noticeable effect on the device. The device tried to contact `2.pool.ntp.org`, which is a NTP server for time synchronisation. For some reason, the DEVELCO Squid.link device did not contact it throughout the profile learning phase, hence it is not allowed by the firewall.

Because of the dropped packet, it was decided to start the profiling again, and leave it running for 30 minutes. After 30 minutes, the profiles were generated and the firewall was configured again. After the firewall configured, the devices were rebooted, in order to make sure that new connections were established.

6.3.2 Results

These results are from the second profile generated. As it turned out that the first set of profiles was incomplete. The firewall logs are shown in the 24 hour test Section 6.3.5.

Tado

The temperature on a thermostat is changed from the app, and the actuator moved in the thermostat device. In this test, it is seen that no packets were dropped in the firewall log. The firewall log is shown in Listings 6.4 and 6.5 in Section 6.3.5.

Philips Hue

Controlling the light which includes, turning it on and changing colour, logging into the application and controlling it both locally and remotely. All functions worked as they should, and from the log of the firewall, it is seen that no packets were dropped. The firewall log is shown in Listings D.1 and D.2 in Appendix D.

DEVELCO Squid.link

The HTTP server is accessed and it works as expected. From the firewall, it can also be seen that there are no dropped packets. The firewall log is shown in Listings D.3 and D.4 in Appendix D.

6.3.3 Exploit Test

The Virtual Machine described in Appendix B is running a HTTP server, which had an exploit that made it possible to perform RCE through the URL. After the firewall is enabled, the exploit is used and it is attempted to create a reserve shell, as explained in Appendix B.2.

6.3.4 Results

After the firewall is enabled the HTTP server is still accessible, and the vulnerability in the HTTP server could still be used. Though, after the firewall is enabled, it is not possible for the machine to create an outgoing connection to the attacker's IP address.

6.3.5 24 Hours Test

In the 24 hours test, the devices are turned on for 24 hours in order to investigate if they try to connect to anything which is not allowed by the firewall. The 24 hour period is included in the functionality test in Section 6.3.1.

This test is made to ensure that the devices do not try to connect to anything which is not included in the firewall. With a correct profile, this should only happen if an attacker tries to gain access to the device. Or if the device is already compromised, and it is trying to establish a connection to other devices (e.g. trying to infect other devices).

6.3.6 Results

This section presents the results of the four devices, namely Tado, Philips Hue, DEVELCO Squid.link and the Virtual Machine. The main thing, which is investigated in these results is the number of packets transmitted, and the number of dropped packets. The firewall log output for Philips Hue, DEVELCO Squid.link and Virtual Machine are found in Appendix D.

```
1 Chain IN_192.168.1.100
2  pkts bytes target prot
3  2715  205K ACCEPT all
4     0     0 ACCEPT icmp
5     0     0 LOG    all
6     0     0 DROP   all
```

Listing 6.4: Iptables rules for Tado, number of packets sent and dropped ingoing traffic.

```
1 Chain OUT_192.168.1.100
2  pkts bytes target prot
3  5284  309K ACCEPT all
4     9     9 ACCEPT udp
5     3   132 ACCEPT tcp
6     0     0 LOG    all
7     0     0 DROP   all
```

Listing 6.5: Iptables rules for Tado, number of packets sent and dropped outgoing traffic.

By looking at Listing D.2 in Appendix D, it can be seen that one of the rules in the firewall for Philips Hue only matches one packet throughout the 24 hours.

As it is seen on Table 6.1, the Virtual Machine has some dropped packets because it is exploited, and it tried to establish an outgoing connection which is not allowed by the firewall rules.

	Total number of accepted packets		Total number of dropped packet	
Devices	Ingoing	Outgoing	Ingoing	Outgoing
Tado	2958	5756	0	0
Philips Hue	6748	7106	0	0
DEVELCO Squid.link	1347	1702	0	0
Virtual Machine	229	232	0	24

Table 6.1: Summary of results from firewall rules. The first three devices do not have any dropped packets, which indicates that the firewall rules work. However, the Virtual Machine has some dropped packets, which indicates that it cannot communicate with the IP address it is trying to.

According to Table 6.1, the three IoT devices do not have any dropped packets. Thus it can be concluded that the firewall does not have any impact on the IoT devices, hence metric Section 2.11.4 is fulfilled. Besides that, there are no dropped packets after 24 hours which was metric Section 2.11.6.

Since the Virtual Machine has some dropped packets in Table 6.1, it demonstrates that it cannot establish a connection to the IP which it is trying. Thereby it cannot participate in a DDoS attack, furthermore it also prevents it from spreading the malware to other device. This fulfils the metrics Sections 2.11.1 and 2.11.2.

6.4 Port Scan

This section compares a port scan on the Wide Area Network (WAN) side of the IGD, before and after the firewall rules are applied on the IGD.

Tado

The Tado device does not have any ports open on the device, hence the firewall rules do not change this result.

```
1 All 1000 scanned ports on 192.168.1.100 are closed (639)\
2                  or filtered (361)
3 Nmap done: 1 IP address (1 host up) scanned in 13.88 seconds
```

Listing 6.6: Port scan of Tado, without firewall rules.

Philips Hue

From Listings 6.7 and 6.8 it can be seen that before the firewall rules are applied, it had both port 80 and 8080 open. But after the firewall rules are applied, port 80 is available, and since no packets are dropped, port 8080 has not been used.

```
PORT     STATE SERVICE
  80/tcp open  http
8080/tcp open  http-proxy
```

Listing 6.7: Port scan of Philips Hue, without firewall rules.

```
PORT   STATE SERVICE
80/tcp open  http
```

Listing 6.8: Port scan of Philips Hue, with firewall rules.

DEVELCO Squid.link

DEVELCO Squid.link has 3 ports open, namely 22, 80 and 10000. After the firewall rules are applied, it is only port 80 which is allowed, and again no packets are dropped, hence port 22 and 10000 are unused.

```
PORT      STATE SERVICE
   22/tcp    open  ssh
   80/tcp    open  http
10000/tcp    open
```

Listing 6.9: Port scan of DEVELCO Squid.link, without firewall rules.

```
PORT   STATE SERVICE
80/tcp open  http
```

Listing 6.10: Port scan of DEVELCO Squid.link, with firewall rules.

6.4.1 Summary

Below, a table with the results is found, it can be seen that on both Philips Hue and DEVELCO Squid.link, there are ports which are not used – in this test at least. Tado, on the other hand does not have any ports open, since it is not hosting any services.

	Open ports	
Device	Before profile	After profile
Tado	None	None
Philips Hue	80, 8080	80
DEVELCO Squid.link	22, 80, 10000	80

6.5 Results Interpretation

From the results in Table 6.1 in Section 6.3, it can be seen that no packets to the devices are dropped. This means the devices functions as intended.

But on Listing D.2 in Appendix D, it can be seen that only one packet matches a rules in the Philips Hue outgoing chain. That packet could easily have been neglected in the learning phase, if that had happened, packets would have been dropped, hence the functionality of the devices would not have been complete.

In the evaluation, the learning phase is only 30 minutes, but in the final version it should automatically be determined when profile is complete. However, in the current version this is not implemented, which is a feature that should be implemented.

A complete overview of the results are found on Table 6.2.

Metric:		Fulfilled	Comment
2.11.1	Prevent botnets from spreading	✓	From Section 6.3.3, it is seen that the firewall prevents unknown connections.
2.11.2	Prevent DoS attacks	✓	From Section 6.3.3, it is seen that the firewall prevents unknown connections.
2.11.3	Operate without user interaction	✗	As a script has to be executed, it requires some user interaction.
2.11.4	No side effect on IoT device	✓	Since there are no dropped packets after 24 hours, it does not effect the IoT devices, as shown in Section 6.3.5.
2.11.5	Auto profile time	✗	This is not fulfilled, as the time had to be increased to 30 minutes for the learning phase.
2.11.6	Long time verification	✓	According to Section 6.3.5, this is passed.

Table 6.2: A table showing a summary of the results, and the metrics which are fulfilled.

CHAPTER

7

CLOSURE

This chapter concludes the project "IoTsec: Automatic Profile-based Firewall for IoT Devices". First, conclusions on the project are presented, which summaries the answers of the problem statement. Afterwards, a discussion is made, which contains suggestions for improving the solution, further work and alternative solutions. Finally, an attacker's perspective is provided, which illustrates some system weaknesses that may be exploitable.

7.1 Conclusion

On 2016-10-21, Twitter, Amazon, Tumblr, Reddit, Spotify, Netflix and more were unavailable due to the DNS provider Dyn being DDoS attacked. This attack was performed by the botnet Mirai, which consisted of 100,000 IoT devices that have been compromised due to inadequate security. Due to a Telnet server on the devices, Mirai was able to gain access to them by using a credential list of 62 usernames and passwords.

This attack has inspired this project, where an analysis of IoT botnets have been performed. In this analysis, the weak points of IoT devices have been examined, in order to bring recommendations on how to improve security on IoT devices. During the initial analysis, the only attack vector that was used is to brute force credentials on a Telnet server, however on 2017-04-06 the Amnesia botnet was discovered. Amnesia uses a vulnerability in the software on specific IoT devices, which can be used for RCE and by doing so compromise the device.

The correct way to solve security issues in Internet-connected devices, is for the manufacturers to make the devices secure by design. For instance, the average user does not need a Telnet server on their IP camera. Since only manufacturers can improve the security on the devices, there is a demand for a universal solution that protects insecure devices. The purpose of this project is to design and implement such a universal solution.

This solution is made to run on IGDs, where it is possible to analyse and influence the Internet traffic to and from the IoT devices. From this analysis, a profile can be

generated for each IoT device, which is done automatically based on traffic that is seen in a learning phase. After the learning phase has ended, the device has been profiled, and from this profile firewall rules can be generated that can be added to the IGD firewall. This way, different restrictions can be applied to different devices at the same time. By restricting the outgoing traffic that IoT devices can send, botnets are prevented from trying to infect other devices.

Since the profiles are generated automatically from observed traffic, user input is not required, this benefits the average user who expects things to just work. Additionally, by sharing generated profiles through a repository, users can receive existing profiles for devices that have just been connected, and prevent infection during the learning phase.

Profiles that are shared through the repository are identified by a fingerprint of the observed traffic. If the fingerprint already exists, the confidence of a profile can be improved by adding a locally generated profile. This can also be used to prevent against malicious profiles being uploaded, by requiring a certain number of independent sources before accepting it as a legitimate profile.

From the system evaluation, the capability of the solution has been assessed. It is shown that the generated profiles support the traffic that flows to and from the tested IoT devices. After the firewall has been configured with the devices' profiles, the functionality of each device still works perfectly. During the 24 hour period after profiling, the devices do not send or receive any traffic that was not specified in the profile. Finally, when the virtual machine was exploited, any outgoing connections were successfully stopped by the solution.

This concludes that a universal solution has been created that successfully can protect IoT devices in the household.

7.2 Discussion

This project has proven that an automated security solution can be made that hinders the spread of IoT botnets. However, the solution still requires more work before it is ready for consumer use. The repository application is only conceptually designed in this project, and it should be completed to facilitate sharing between IGDs.

Besides that, the learning phase ran for fixed amount of time, which is 30 minutes. While this works with the three devices tested in the project, specifically Tado, Philips Hue and DEVELCO Squid.link, there is no guarantee that it works for all IoT devices. A different device could contact a server after 31 minutes, which would be excluded from the profile, and therefore not be able to communicate with it. Therefore, a solution to when the learning phase should end must be found before the system can be released.

Another aspect, which is not taken into consideration is firmware upgrades, since after the learning phase is completed, the profile cannot be changed. This means that, if e.g. Philips Hue decides to add another service, which is hosted on the Philips Hue bridge, it is not be permitted in the firewall. Or similarly if Philips decides to change the NTP

servers used by Philips Hue, it would also be blocked by the firewall, since it is not included in the profile, and it is currently not possible to change the profile. On the same note, if Philips decides to stop using e.g. a NTP server, it is still included in the profile, and hence the firewall.

Furthermore, it should be decided how the solution is distributed. In Section 4.4.3, it has been described that profiles must be signed by the application, to inhibit malicious profiles from being shared. This requires a unique key for each instance of application, in order to count the number of unique profiles made for the same device. These keys must be managed by the repository, and therefore be distributed with the solution. One way of doing this, is to distribute the solution with an IGD and thereby embed a unique key. Though this limits the distribution to only selected IGDs.

In addition to distributing the solution with the IGDs, a read-only version could also be released. This version would be without a key, and as such can only download profiles or generate local ones that cannot be shared. Thereby, it is possible to distribute it to existing IGDs, through firmware upgrades.

7.2.1 Future Use of This Solution

This solution should be considered a short-term countermeasure against IoT botnets, since security should be part of the design of IoT devices. Hopefully, manufacturers become better at creating secure IoT devices in the future, and thus removes the problem that this solution solves.

It is estimated that there will be 50 billion IoT devices connected in 2020, and if the security of the devices is not improved, it is going to become a serious issue. According to the research, which this project is based on, the attacks have only been against non-critical services, but it could easily be used to attack critical infrastructure as well.

7.2.2 Alternative Solutions

Rather than trying to fix broken products, an alternative approach could be to remove broken products from the Internet. Similar to the Conformité Européenne (CE) label, which shows that a product comply with rules within its product category. A certification process and label could be introduced for consumer IoT products. This certification process could verify that a product is following the best practices at the time of development, and complying with specific security guidelines.

Certification processes are also used within many of the existing IoT technologies, such as ZigBee, Z-Wave and Bluetooth etc. These certifications show that devices are conforming to the specifications of that technology.

Through this certification process, manufacturers could also be required to specify what Internet traffic it participates in. And thereby, high quality profiles can be produced, which can be used by this project.

7.3 Attacker's Perspective

In this section, the solution is analysed from an attacker's perspective, who would like to compromise the system. The methods that are described are not complete attacks that can be carried out, but rather entry points that possibly can be exploited.

7.3.1 Profile Generation

Since the system relies on automatic profile generation, it is possible that it can be exploited to additional information in the profile, which in turn permits more traffic in the firewall. In order to exploit this vulnerability, the attacker must be able to affect multiple profile learning phases, since the system relies on multiple profiles to improve the confidence in them.

Accordingly, the attacker must compromise the IoT device during the learning phase, and use it to connect to his own Internet service. This causes additional traffic to be recorded in the profile. If the attacker chooses to use a DNS name to connect to the Internet service, it is the DNS name that is recorded in the profile. By doing so, the attacker is able to change the IP address dynamically, since the system permits traffic to IP addresses that have been resolved behind a DNS name.

This attack vector enables specific IoT devices that are already vulnerable (since the attacker must be able to compromise it, to affect the profile generation) to participate in attacks.

7.3.2 Change Wi-Fi Network Association

This attack vector only affects IoT devices that are connected by Wi-Fi, not ethernet. If a device can be compromised, and an attacker can do RCE on it and by doing so acquire system privileges, the attacker could try to connect the device to a different Wi-Fi network. By doing this, the IoT device can be switched to a less restrictive network, e.g. the MPD network offered by the IGD. This, of course, requires the network security key to associate with the network, which is something that the attacker must brute force or otherwise circumvent.

7.3.3 Summary

This section has brought up two possible attack vectors that can be used to circumvent the system. However, both attack vectors require an IoT device that already contains a vulnerability such that an attacker can compromise the device.

BIBLIOGRAPHY

[1] (2016, Oct) Hacked Cameras, DVRs Powered Today's Massive Internet Outage. [Online]. Available: https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/

[2] (2016, Jun.) Reality Check: 50B IoT devices connected by 2020 – beyond the hype and into reality. [Online]. Available: http://www.rcrwireless.com/20160628/opinion/reality-check-50b-iot-devices-connected-2020-beyond-hype-reality-tag10

[3] (2016, Oct) Good news: The hackers who broke the internet last week are less powerful than originally believed. [Online]. Available: https://qz.com/820003/dyn-dns-ddos-the-mirai-botnet-is-smaller-than-originally-thought/

[4] (2013, Nov.) How the first botnet changed the internet forever. [Online]. Available: https://www.dailydot.com/crime/robert-morris-botnet-virus-changed-internet/

[5] (2011) How Criminals Build Botnets for Profit. [Online]. Available: https://www.damballa.com/downloads/r_pubs/DoD-Cyber-crime_Gunter-Ollmann_How-Criminals-Profit.pdf

[6] (2014, Jun.) A Guy Mined $600K of Dogecoin with a Botnet of Storage Devices. [Online]. Available: https://motherboard.vice.com/en_us/article/dogecoin-could-well-be-the-hackers-cryptocurrency-of-choice

[7] (2016, Oct) Mapping Mirai: A Botnet Case study. [Online]. Available: https://www.malwaretech.com/2016/10/mapping-mirai-a-botnet-case-study.html

[8] (2016, 8) MMD-0056-2016 - Linux/Mirai, how an old ELF malcode is recycled. [Online]. Available: http://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html

[9] (2016, Oct) Mirai-Source-Code. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code/search?q=ssh&type=Code

[10] (2016, Nov) The State of security in the Connected Home. [Online]. Available: https://medium.com/@LumaHome/the-state-of-security-in-the-connected-home-2a979b76f85b#.gmdby8io0

[11] (2017, Jan) Who is Anna-Senpai, the Mirai Worm Author? [Online]. Available: https:
//krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/

[12] (2016, Sep) BASHLITE Botnets Ensnare 1 Million IoT Devices. [Online]. Available:
http://www.securityweek.com/bashlite-botnets-ensnare-1-million-iot-devices

[13] (2017, Mar) There's a 120,000-Strong IoT DDoS Botnet Lurking Around. [Online]. Available: http://news.softpedia.com/news/
there-s-a-120-000-strong-iot-ddos-botnet-lurking-around-507773.shtml

[14] (2017, Mar) LizardStresser IoT botnet launches 400Gbps DDoS attack. [Online]. Available: http://www.computerweekly.com/news/450299445/
LizardStresser-IoT-botnet-launches-400Gbps-DDoS-attack

[15] (2017, Mar) BASHLITE Malware Uses ShellShock to Hijack Devices Running BusyBox. [Online]. Available: http://www.securityweek.com/
bashlite-malware-uses-shellshock-hijack-devices-running-busybox

[16] (2017, Mar) BASHLITE Malware leverages ShellShock Bug to Hijack Devices Running BusyBox. [Online]. Available:
https://thehackernews.com/2014/11/bashlite-malware-leverages-shellshock.html

[17] (2017, Mar) A new BASHLITE variant infects devices running BusyBox. [Online]. Available: http://securityaffairs.co/wordpress/30225/cyber-crime/
bashlite-exploits-shellshock.html

[18] (2017, Mar) BASHLITE Affects Devices Running on BusyBox. [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/
bashlite-affects-devices-running-on-busybox/

[19] (2017, Mar) Bash Vulnerability Leads to Shellshock: What it is, How it Affects You. [Online]. Available:
http://blog.trendmicro.com/trendlabs-security-intelligence/
shell-attack-on-your-server-bash-bug-cve-2014-7169-and-cve-2014-6271/

[20] (2017, Mar) Github: anthonygtellez/BASHLITE. [Online]. Available:
https://github.com/anthonygtellez/BASHLITE

[21] (2017, Mar) CVE-2014-6271: remote code execution through bash. [Online]. Available: http://seclists.org/oss-sec/2014/q3/650

[22] (2017, Feb.) IoT? I don't care. [Online]. Available:
https://www.scmagazine.com/iot-i-dont-care/article/634990/

[23] (2017, Apr.) BrickerBot, the permanent denial-of-service botnet, is back with a vengeance. [Online]. Available: https://arstechnica.com/security/2017/04/
brickerbot-the-permanent-denial-of-service-botnet-is-back-with-a-vengeance/

[24] (2017, Apr.) BrickerBot: Back With A Vengeance. [Online]. Available: https://security.radware.com/ddos-threats-attacks/brickerbot-pdos-back-with-vengeance/

[25] (2011, Sep) Morto Post Mortem: Dissecting a Worm. [Online]. Available: https://www.imperva.com/blog/2011/09/morto-post-mortem-a-worm-deep-dive/

[26] (2017, Jan) Bot Chatter: Ragebot Botnet Malware Morphs. [Online]. Available: https://securityintelligence.com/news/bot-chatter-ragebot-botnet-malware-morphs/

[27] (2013, Nov) You're infected—if you want to see your data again, pay us $300 in Bitcoins. [Online]. Available: https://arstechnica.com/security/2013/10/youre-infected-if-you-want-to-see-your-data-again-pay-us-300-in-bitcoins/

[28] (2011, May) XP AntiSpyware 2011. [Online]. Available: http://www.precisesecurity.com/rogue/xp-anti-spyware-2011

[29] (2014, Nov) Regin: nation-state ownage of GSM networks. [Online]. Available: https://securelist.com/blog/research/67741/regin-nation-state-ownage-of-gsm-networks/

[30] (2004, April) W32.Sasser.Worm. [Online]. Available: https://www.symantec.com/security_response/writeup.jsp?docid=2004-050116-1831-99

[31] (2004, May) Sasser net worm affects millions. [Online]. Available: http://news.bbc.co.uk/2/hi/technology/3682537.stm

[32] (2017, Mar) Timeline of computer viruses and worms. [Online]. Available: https://en.wikipedia.org/wiki/Timeline_of_computer_viruses_and_worms

[33] (2017, Apr.) New IoT/Linux Malware Targets DVRs, Forms Botnet. [Online]. Available: http://researchcenter.paloaltonetworks.com/2017/04/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/

[34] (2016, Mar.) Remote Code Execution in CCTV-DVR affecting over 70 different vendors . [Online]. Available: http://www.kerneronsec.com/2016/02/remote-code-execution-in-cctv-dvrs-of.html

[35] OMNIPOWER intelligente elmålere sikrer det fulde udbytte af dit smart grid. [Online]. Available: https://www.kamstrup.com/da-dk/produkter-loesninger/smart-grid/elmaalere

[36] (2016, Oct) The Best Smart Smoke Alarm. [Online]. Available: http://thewirecutter.com/reviews/best-smart-smoke-alarm/

[37] (2013, Dec) Quirky Egg Minder review. [Online]. Available: https://www.cnet.com/products/quirky-egg-minder/review/

[38] K. Angrishi, "Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets," *IEEE*, vol. 17, Feb 2017.

[39] (2016, Dec) Zero-day exploits could potentially turn hundreds of thousands of IP cameras into IoT botnet slaves. [Online]. Available: https://www.cybereason.com/zero-day-exploits-turn-hundreds-of-thousands-of-ip-cameras-into-iot-botnet-slaves/

[40] (2017, Mar.) Multiple vulnerabilities found in Wireless IP Camera (P2P) WIFICAM cameras and vulnerabilities in custom http server. [Online]. Available: https://pierrekim.github.io/blog/2017-03-08-camera-goahead-0day.html

[41] M. Stamp, *Information Security, Principles and Pratice*, 2nd ed. Wiley, 2011.

[42] rockyou.txt. [Online]. Available: http://scrapmaker.com/view/dictionaries/rockyou.txt

[43] Common User Passwords Profiler. [Online]. Available: https://github.com/Mebus/cupp

[44] (2015, 03) Password Cracking With Amazon Web Services - 36 Cores. [Online]. Available: http://blog.nullmode.com/blog/2015/03/22/36-core-aws-john/

[45] J. Galbraith et al., "Secure Shell Public Key Subsystem," Internet Requests for Comments, RFC Editor, RFC 4819, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4819

[46] D. Hardt, Ed., "The OAuth 2.0 Authorization Framework," Internet Requests for Comments, RFC Editor, RFC 6749, 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749

[47] (2017, Jun.) RSA SECURID SUITE. [Online]. Available: https://www.rsa.com/en-us/products/rsa-securid-suite

[48] D. M'Raihi et al., "HOTP: An HMAC-Based One-Time Password Algorithm," Internet Requests for Comments, RFC Editor, RFC 4226, 2005. [Online]. Available: https://tools.ietf.org/html/rfc4226

[49] Y. Rekhter et al., "Address Allocation for Private Internets," Internet Requests for Comments, RFC Editor, RFC 1918, 1997. [Online]. Available: https://tools.ietf.org/html/rfc1918

[50] T. Dierks et al., "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246

[51] (2017, Jun.) Tado API. [Online]. Available: http://blog.scphillips.com/posts/2017/01/the-tado-api-v2/

[52] (2017, Jun.) hue Developer Program. [Online]. Available: https://www.developers.meethue.com/

[53] (2017, Jun.) Squid.link Gatewat. [Online]. Available: https://www.develcoproducts.com/products/gateways/squidlink-gateway/

APPENDIX

A

IOT DEVICES

This appendix briefly describes the functionality of the IoT devices used in this project. The devices used in this project are devices which have been available from the beginning, either from the university, or personally owned devices.

The used devices are limited to three devices, namely a Tado thermostat, a Philips Hue and a IoT gateway from Develco called DEVELCO Squid.link.

A.1 Tado

The Tado set used in this project is a "Smart Radiator Thermostat", this includes a bridge which is connected to the internet, and one (or more) smart Radiator Thermostats which controls the valve on the radiator. The bridge sends commands to the thermostats, which the bridges receives from a remote management service.

The management service gets information from a smartphone application. This is used to detect if the phone is home, or away – hence if the Thermostats should be on or off [51].

A.2 Philips Hue

Philips Hue consists of a bridge and some bulbs. Like Tado, the bridge is connected to the Internet, and sends commands to the light bulbs. The lights can be controlled from an application on a smartphone. Contrary to the Tado bridge, the Philips Hue bridge hosts 2 services, a web server on port 80 and an unknown service on port 8080. This means the Philips Hue bridge can be used without Internet access, if both the smartphone and the bridge are connected to the same network. But the Philips Hue bridge also communicates with a remote server, to make it possible to connect to the hue remotely without port forwarding [52].

A.3 DEVELCO Squid.link

The DEVELCO Squid.link IoT bridge, is a bridge which can be used to connect different IoT devices to the Internet, e.g. it could replace the Philips Hue bridge. E.g. Kamstrup uses such a bridge to connect their smart electricity meter to the Internet, hence give the costumers access to the data from the smart meters.

Unfortunately, without access to any smart meters, it is only possible to watch the bridge communicate with the Internet.

The bridge provides a web server on port 80, and an SSH server on port 22, finally it also has an unknown service on port 10000 [53].

APPENDIX

B

VULNERABLE VIRTUAL MACHINE

In the System Evaluation chapter, a virtual machine has been utilised, to demonstrate that outgoing traffic is blocked. This virtual machine is supposed to be vulnerable, such that an attacker can control it. The virtual machine is described in this appendix.

B.1 Setup

In the virtual machine, a web server mimics an IoT device. When browsing the web server, a static message is shown, see Listing B.1.

```
1 <h1>DVR System</h1>
2 <p>Welcome to this poorly secured DVR system.</p>
```

Listing B.1: Static message of the web server.

However, hidden in the `cgi-bin` folder, which can hold executable files to show dynamic content, a script has been created. This script, named `cmd` has a vulnerability that allows RCE. The script is seen on Listing B.2.

```
1  #!/usr/bin/python
2  import os, urllib.parse
3  from subprocess import PIPE, Popen
4
5  def cmdline(command):
6      process = Popen(args=command,stdout=PIPE,shell=True)
7      return process.communicate()[0].decode('utf-8')
8
9  qs = urllib.parse.parse_qs(os.environ['QUERY_STRING'])
10 print()
11 print(cmdline(qs['cmd'][0]))
```

Listing B.2: Script that has a RCE vulnerability.

From inspection of Listing B.2, it can be determined that it accepts arbitrary commands and executes them. On lines 5-7, a function is defined that accepts a command and

executes it in a shell. On lines 9-11, a command is extracted from the query string, and the function is called and the output is sent to the requestor.

The script is very insecure, and no measures have been added to prevent execution of malicious commands. However, it has been protected by a password, such that only the manufacturer has access to it. The web server used is the built in `httpd` in `busybox`, where a configuration line:

```
/cgi-bin:admin:admin
```

secures the `cgi-bin` directory with the hard to guess username and password `admin` and `admin`.

B.2 Exploitation

To exploit the vulnerability that has been created, a client can craft a malicious request. For example, the user id which the web server is running as can be determined through the request:

```
http://admin:admin@192.168.1.130/cgi-bin/cmd?cmd=id
```

which returns something a response like:

```
uid=99(nobody) gid=99(nobody) groups=99(nobody)
```

Afterwards, an attacker could initiate a reverse shell, for example, to get more direct access to the system. The following request can be used:

```
http://.../cgi-bin/cmd?cmd=nc%20203.0.113.99%20443%20-e%20/bin/sh
```

Where the encoded command is

```
nc 203.0.113.99 443 -e /bin/sh
```

Which establishes an outgoing connect to 203.0.113.99 on port 443 and launches a shell, which the attacker can use.

APPENDIX

C

PROFILES

This appendix contains the profiles and firewall rules generated for Philips Hue, DEVELCO Squid.link and the virtual machine.

Philips Hue

The generated profile can be seen on Figure C.1, which shows the result from the web interface. The firewall rules that have been generated from the profile are shown in Listings C.1 and C.2.

```
Chain IN_192.168.1.120 (1 references)
target prot source    destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0  state RELATED,ESTABLISHED
ACCEPT icmp 0.0.0.0/0 0.0.0.0/0
ACCEPT udp  0.0.0.0/0 0.0.0.0/0  udp dpt:68
ACCEPT tcp  0.0.0.0/0 0.0.0.0/0  tcp dpt:80
LOG    all  0.0.0.0/0 0.0.0.0/0  LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing C.1: Iptables rules for Philips Hue ingoing traffic.

```
Chain OUT_192.168.1.120 (1 references)
target  prot  source     destination
ACCEPT  all  0.0.0.0/0  0.0.0.0/0    state RELATED,ESTABLISHED
ACCEPT  udp  0.0.0.0/0  192.168.1.1  udp dpt:53
ACCEPT  udp  0.0.0.0/0  0.0.0.0/0    match-set set_49c18b0d dst udp dpt:123
ACCEPT  udp  0.0.0.0/0  0.0.0.0/0    match-set set_28fc68cf dst udp dpt:123
ACCEPT  udp  0.0.0.0/0  0.0.0.0/0    match-set set_fe6bb532 dst udp dpt:123
ACCEPT  udp  0.0.0.0/0  0.0.0.0/0    match-set set_727c5739 dst udp dpt:123
ACCEPT  tcp  0.0.0.0/0  0.0.0.0/0    match-set set_18872aa8 dst tcp dpt:80
ACCEPT  tcp  0.0.0.0/0  0.0.0.0/0    match-set set_b790e2c0 dst tcp dpt:80
ACCEPT  tcp  0.0.0.0/0  0.0.0.0/0    match-set set_a717c899 dst tcp dpt:80
ACCEPT  tcp  0.0.0.0/0  0.0.0.0/0    match-set set_4ad8b32a dst tcp dpt:80
ACCEPT  tcp  0.0.0.0/0  0.0.0.0/0    match-set set_628bd55b dst tcp dpt:443
ACCEPT  tcp  0.0.0.0/0  0.0.0.0/0    match-set set_8399d78e dst tcp dpt:80
LOG     all  0.0.0.0/0  0.0.0.0/0    LOG flags 0 level 4 prefix "pyprofiler:"
DROP    all  0.0.0.0/0  0.0.0.0/0
```

Listing C.2: Iptables rules for Philips Hue outgoing traffic.

DEVELCO Squid.link

The generated profile can be seen on Figure C.2, which shows the result from the web interface. The firewall rules that have been generated from the profile are shown in Listings C.3 and C.4.

```
Chain IN_192.168.1.120 (1 references)
target prot source    destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0 state RELATED,ESTABLISHED
ACCEPT icmp 0.0.0.0/0 0.0.0.0/0
ACCEPT udp  0.0.0.0/0 0.0.0.0/0 udp dpt:68
ACCEPT tcp  0.0.0.0/0 0.0.0.0/0 tcp dpt:80
LOG    all  0.0.0.0/0 0.0.0.0/0 LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing C.3: Iptables rules for DEVELCO Squid.link ingoing traffic.

```
Chain OUT_192.168.1.120 (1 references)
target prot source    destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0   state RELATED,ESTABLISHED
ACCEPT udp  0.0.0.0/0 192.168.1.1 udp dpt:53
ACCEPT udp  0.0.0.0/0 0.0.0.0/0   match-set set_2c1a2c78 dst udp dpt:123
ACCEPT udp  0.0.0.0/0 0.0.0.0/0   match-set set_e7271e40 dst udp dpt:123
ACCEPT udp  0.0.0.0/0 0.0.0.0/0   match-set set_7fbcfa5a dst udp dpt:123
ACCEPT udp  0.0.0.0/0 0.0.0.0/0   match-set set_123dac95 dst udp dpt:123
ACCEPT tcp  0.0.0.0/0 0.0.0.0/0   match-set set_c8c20897 dst tcp dpt:64201
ACCEPT udp  0.0.0.0/0 192.168.1.1 udp dpt:67
LOG    all  0.0.0.0/0 0.0.0.0/0   LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing C.4: Iptables rules for DEVELCO Squid.link outgoing traffic.

Virtual Machine

The generated profile can be seen on Figure C.3, which shows the result from the web interface. The firewall rules that have been generated from the profile are shown in Listings C.5 and C.6.

```
Chain IN_192.168.1.130 (1 references)
target prot source    destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0 state RELATED,ESTABLISHED
ACCEPT icmp 0.0.0.0/0 0.0.0.0/0
ACCEPT udp  0.0.0.0/0 0.0.0.0/0 udp dpt:68
ACCEPT tcp  0.0.0.0/0 0.0.0.0/0 tcp dpt:80
LOG    all  0.0.0.0/0 0.0.0.0/0 LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing C.5: Iptables rules for virtual machine ingoing traffic.

```
Chain OUT_192.168.1.130 (1 references)
target prot source    destination
ACCEPT all  0.0.0.0/0 0.0.0.0/0   state RELATED,ESTABLISHED
ACCEPT udp  0.0.0.0/0 192.168.1.1 udp dpt:53
ACCEPT tcp  0.0.0.0/0 0.0.0.0/0   match-set set_2b0e2f72 dst tcp dpt:80
LOG    all  0.0.0.0/0 0.0.0.0/0   LOG flags 0 level 4 prefix "pyprofiler:"
DROP   all  0.0.0.0/0 0.0.0.0/0
```

Listing C.6: Iptables rules for virtual machine outgoing traffic.

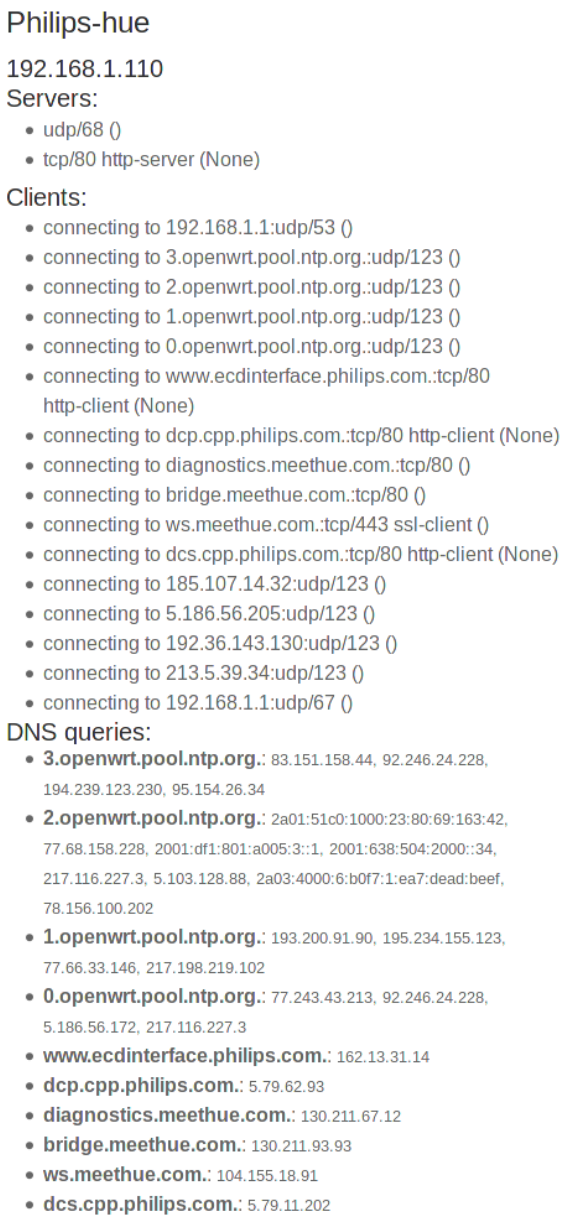

Figure C.1: Philips Hue profile.

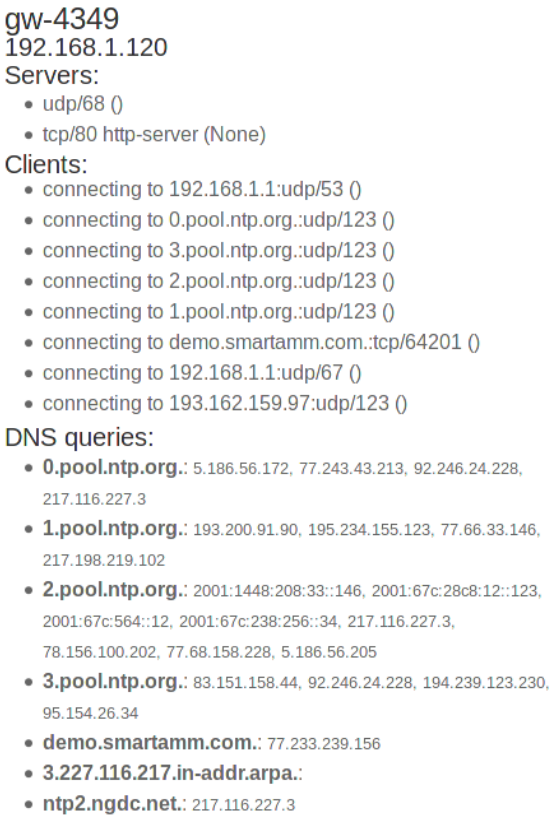

Figure C.2: DEVELCO Squid.link profile.

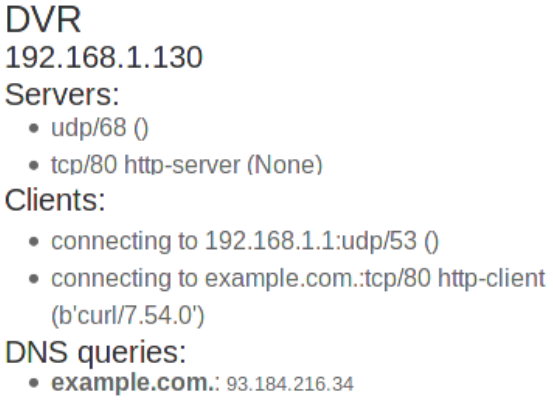

Figure C.3: Virtual machine profile.

APPENDIX

D

FIREWALL RESULTS

Appendix of the firewall log output after 24 hours.

Philips Hue

These are the results from the ingoing and outgoing chain for Philips Hue. It can be seen that it makes contact with a server, but also has incoming traffic. But no packets are dropped, which is the goal of this test.

On Listing D.2, it can be seen that one of the rules is only matched once. This is problematical, as there is a high risk that it would not be included in the profile, and traffic would be dropped as consequence.

```
Chain IN_192.168.1.110
 pkts bytes target prot
 6106  486K ACCEPT all
    0     0 ACCEPT icmp
    0     0 ACCEPT udp
  337 22620 ACCEPT tcp
    0     0 LOG    all
    0     0 DROP   all
```

Listing D.1: Iptables rules for Philips Hue, number of packets sent and dropped ingoing traffic.

```
Chain OUT_192.168.1.110
 pkts bytes target
    0     0 ACCEPT
  119  9044 ACCEPT
  122  9272 ACCEPT
  130  9880 ACCEPT
  112  8512 ACCEPT
   19  6176 ACCEPT
   96  4992 ACCEPT
   15   780 ACCEPT
   11   572 ACCEPT
    5   260 ACCEPT
    1    52 ACCEPT
    0     0 LOG
    0     0 DROP
```

Listing D.2: Iptables rules for Philips Hue, number of packets sent and dropped outgoing traffic.

DEVELCO Squid.link

The DEVELCO Squid.link has no dropped packets.

```
Chain IN_192.168.1.120
 pkts bytes target
  804 57538 ACCEPT
    0     0 ACCEPT
    0     0 ACCEPT
    0     0 ACCEPT
    0     0 LOG
    0     0 DROP
```

Listing D.3: Iptables rules for DEVELCO Squid.link, number of packets sent and dropped ingoing traffic.

```
Chain OUT_192.168.1.120
 pkts bytes target
  576 33093 ACCEPT
    0     0 ACCEPT
  131  9956 ACCEPT
  133 10108 ACCEPT
  131  9956 ACCEPT
   89  6764 ACCEPT
    2   120 ACCEPT
    0     0 LOG
    0     0 DROP
```

Listing D.4: Iptables rules for DEVELCO Squid.link, number of packets sent and dropped outgoing traffic.

Virtual Machine

The Virtual Machine has some dropped packets, but it is due to the exploit test. The dropped packets show that the firewall is working.

```
Chain IN_192.168.1.100
 pkts bytes target
  185 20301 ACCEPT
    0     0 ACCEPT
    0     0 ACCEPT
   44  2640 ACCEPT
    0     0 LOG
    0     0 DROP
```

Listing D.5: Iptables rules for Virtual Machine, number of packets sent and dropped ingoing traffic.

```
Chain OUT_192.168.1.130
 pkts bytes target
  230 14837 ACCEPT
    0     0 ACCEPT
    2   120 ACCEPT
   24  1360 LOG
   24  1360 DROP
```

Listing D.6: Iptables rules for Virtual Machine, number of packets sent and dropped outgoing traffic.