UCLouvain

epl

École polytechnique de Louvain

Design of a security gateway for domestic IoT

Author: **Franck FADEUR**
Supervisor: **Ramin SADRE**
Readers: **Etienne RIVIÈRE, Igor ZAVALYSHYN**
Academic year 2019–2020
Master [120] in Cybersecurity

Abstarct

The popularity of the Internet of Things is growing up day after day. More and more new devices proposed by the constructors nowadays can be connected to the network and to the Internet. For example, some toasters, washing machines, lamps, smart televisions, etc. All of those devices used to be disconnected but not any more.

These connected devices usually send a lot of data compared to the user's daily usage. Because of this, the manufacturers are able to collect a huge load of data and based on it, profile the users. Moreover, these data could be used on the behalf of a user without their acknowledgement.

Several problems such as the fingerprinting of the devices, analysis of the user data and side channel attack on the number of packets were highlighted within this paper.

This master thesis has explored a way to control the amount of data sent by the devices out of the network. In this case, two devices would be dependent. One would be the parent and the other one the child. The goal is to let the child's DNS request going out of the local area network while the parent is also sending DNS requests. Thanks to this concept, the amount of packet sent by the child is reduced since the only periods it is allowed to do so is while the parent is sending requests too.

"*Despite continued security problems, the IoT will spread and people will become increasingly dependent on it. The cost of breaches will be viewed like the toll taken by car crashes, which have not persuaded very many people not to drive.*"

Richard Adler, unknown

"*Despite hacks and privacy issues, people will feel a need to keep connected, partly because companies will reward them for doing so (or make life difficult if they don't).*"

Joseph Turow, unknown

Acknowledgment

First, I would like to thank my supervisor, Professor Ramin Sadre, for his patience, advice and helping me to focus on what really matters. Indeed, thanks to multiple interactions with him I could greatly improve my work. Without him, this Master Thesis would not have been the same. Professor Sadre really guided me all along the year to obtain the work you are about to read. Moreover, he succeeded to guide me when I was not confident with all the technologies used to realise the Proof of Concept. Some moments were really challenging but his unwavering support helped me to overcome difficulties.

This year was not easy due to the COVID-19 pandemic, so I had to work exclusively from home during the second semester. However, Professor Sadre did not abandon me and we still had regular meetings. Obviously, it was a bit more difficult to present the results of the past weeks but he trusted me and listened to me in order to redirect me when it was necessary.

I would like to acknowledge the two readers of my thesis Professor E. Rivière and I. Zavalyshyn for taking the time to read my work.

Finally, I would like to thank my close friends, my internship colleagues and my family for the support during this period. It was not always easy to go through all the difficulties I faced during these months and their advice and support were more than necessary to address these challenges.

Contents

List of Figures

1. Introduction

1.1 Background and objectives of the thesis

The popularity of IoT's is growing up day after day. More and more new devices proposed by the constructors nowadays can be connected to the network and to the Internet. For example, some toasters, washing machines, lamps, smart televisions ... all of those devices used to be disconnected but not any more.

These connected devices usually send a lot of data compared to the user's daily usage. Because of this, the manufacturers are able to collect a huge load of data and based on it, profile the users. Moreover, these data could be used on the behalf of a user without their acknowledgement. In Belgium, the usage of the data is regulated thanks to the General Data Protection Regulation but this is not the case everywhere in the world.

This master thesis presents the State of the Art about some existing solutions to filter data sent by connected devices. The main objective of this work is to propose a new concept to control the amount of data sent by devices on the network. Several problems have been identified.

Firstly, the traffic data can lead to an identification and a fingerprinting of the devices thanks to the amount of packets and their content. Even encrypted, it is still possible to infer information from the number of packets.

Moreover, some IoT's contact unnecessary domain names set up by the manufacturer to make statistics with the data. Finally, the end users of IoT's rarely master the security. Add to this a lack of secure firmware, insufficient authentication and authorisation, this can lead to dangerous situations.

1.2 Structure of the thesis

The thesis follows the next structure:

The chapter 2, *Background knowledge*, explains the key concepts required to correctly understand the remaining of the thesis.

The chapter 3, *State of the Art*, is mainly the results of scientific papers read. It presents and highlights scientific progress about IoT and the actual concepts regarding gateways, analysis of traffic data and data leak.

The chapter 4, *Identified problems*, is the output of the problems we identified, based on the State of the Art.

The chapter 5, *Concept*, proposes and explores some concepts. These concepts were developed thanks to the State of the Art and the Identified problems.

The chapter 6, *Proof of Concept*, is intended to implement the concept explained in the previous chapter. The results and the limitations are also presented at the end of this chapter.

The chapter 7, *Future work*, addresses the future work in order to improve the concept and the capabilities of the latter.

The chapter 8, *Conclusion*, sums up and concludes this master thesis.

1.3 Contributions

The contribution is the parent-child principle presented in the section 5.1 and implemented in subsection 6.3.2. This concept aims to control the amount of data sent by the devices out of the network. In this case, two devices would be dependent. One would be the parent and the other one the child. The goal is to let the child's DNS request going out of the Local Area Network (LAN) while the parent is also sending DNS requests.

The poster (Appendix A) produced in collaboration with Professor Sadre as part of the UCLouvain poster session gives a graphical overview of the challenges faced during the implementation.

1.4 Notations

AP Access Point

ARM Advanced RISC Machines

ARP Address Resolution Protocol

BAS Building Automation Systems

CPU Central Processing Unit

DNS Domain Name System

IBM International Business Machines Corporation

IDS Intrusion Detection System

IEEE Institute of Electrical and Electronics Engineers

IIoT Industrial Internet of Things

IoT Internet of Things

IP Internet Protocol

LAN Local Area Network

LTE Long Term Evolution

M2M Machine-To-Machine

MUD Manufacturer Usage Description

OS Operating System

pcap Packet CAPture

RAM Random Access Memory

SCADA Supervisory Control And Data Acquisition

SNI Server Name Identification

SSH Secure Shell

TLD Top Level Domain

URL Uniform Resource Locator

USB Universal Serial Bus

VLAN Virtual LAN

VPN Virtual Private Network

Wi-Fi Wireless Fidelity

WLAN Wireless Local Area Network

2. Background knowledge

This chapter aims to define the key concepts used in some chapters of this master thesis. A comparison between different definitions is firstly presented to show the multiple domains where IoT's are used. Then we present our definition, the usage within this master thesis and the different architectures.

A second concept presented in this chapter is DNS. Again, a definition is given followed by the principle and an illustration of the way it works.

2.1 Internet of Things

2.1.1 Definitions

IoT's have multiple definitions and are present in different sectors. The following definitions will give an overview of the differences:

Definition 2.1.1 IoT - Oracle[1]
"*Industrial Internet of Things (IIoT) refers to the application of IoT technology in industrial settings, especially with respect to instrumentation and control of sensors and devices that engage cloud technologies. Recently, industries have used Machine-To-Machine (M2M) communication to achieve wireless automation and control. But with the emergence of cloud and allied technologies (such as analytics and machine learning), industries can achieve a new automation layer and with it creates new revenue and business models. IIoT is sometimes called the fourth wave of the Industrial Revolution, or Industry 4.0. The following are some common uses for IIoT:*

- *Smart manufacturing*
- *Preventive and predictive maintenance*
- *Smart power grids*
- *Smart cities*
- *Connected and smart logistics*
- *Smart digital supply chains*

"

The definition of Oracle is oriented to the Industrial Internet of Things. It is interesting to note that the IoT's are present in multiple domains including industry. In this case, the objective is to automate processes thanks to the connected sensors.

Definition 2.1.2 IoT - IBM [2] - November 17, 2016
"*In a nutshell, the Internet of Things is the concept of connecting any device (so long as it has an on/off switch) to the Internet and to other connected devices. The IoT is a giant network of connected things and people – all of which collect and share data about the way they are used and about the environment around them. That includes an extraordinary number of objects of all shapes and sizes – from smart microwaves, which automatically cook your food for the right length of time, to self-driving cars, whose complex sensors detect objects in their path,*

> *to wearable fitness devices that measure your heart rate and the number of steps you've taken that day, then use that information to suggest exercise plans tailored to you. There are even connected footballs that can track how far and fast they are thrown and record those statistics via an app for future training purposes.*"

For International Business Machines Corporation (IBM), the definition given by the company is very large. They go from smart microwaves to self-driving cars. Within this definition, multiple domains are presented such as

- Health-care
- Automotive
- Home appliances
- Wearable connected devices

Definition 2.1.3 IoT - Red Hat [3]
"*In simple terms, the Internet of Things (IoT) refers to an ongoing trend of connecting all kinds of physical objects to the internet, especially ones that you might not expect. This can mean everything from common household objects like refrigerators and lightbulbs, to business assets like shipping labels and medical devices, to unprecedented wearables, smart devices, and even smart cities that only exist because of IoT.*"

As for IBM, RedHat gives a very broad definition. It is therefore complicated to understand what should or should not be considered as an IoT.
After the exposition of the definitions provided by different major manufacturers, it is important to specify the various areas where IoT's are present. This will make possible to realise the extent of the phenomenon and that the connected objects are finally almost everywhere.

Comparing the definition of Oracle (see definition 2.1.1) with the one of RedHat (see definition 2.1.3), a notable distinction emerges. Indeed, in one case, it is more industry-oriented, whereas it is more domestic-oriented in the other one. We can already distinguish two types of use.

A third statement in IBM's definition (see definition 2.1.2), the connected car, does not fit into either home-oriented or industry-oriented use. Moreover, in this same definition, IBM exposes a fourth usage which is health-oriented.

2.1.2 Thesis usages

After clarifying the different areas and definitions concerning IoT's, here is the vision that guided the realisation of this work:
As we see them, IoT's have been defined as connected objects with domestic use. This means that connected objects related to health, industry, automotive and others have not been taken into account.
The type of considered connected object are, for examples:

- Lamps
- Home assistants (Alexa)
- Sensors
- Thermostat

An IoT gateway is the link between the IoT's and the network. Some IoT's use *Zigbee* to enable communication between the device and the IoT gateway.

Definition 2.1.4 Zigbee - NXP [4]
"*Zigbee®is a networking protocol defined by the Zigbee Alliance for low-cost, low-power, wireless control and monitoring solutions. Designed on top of the IEEE®802.15.4 standard, Zigbee is a self-healing, secure, robust, mesh protocol that can scale to hundreds of nodes across large areas. Zigbee networks can participate in the "Internet of Things" (IoT), allowing the network nodes to be remotely monitored and controlled by devices on the Internet, such as smartphones, tablets, and laptops.*"

The IoT gateway is connected to the LAN. This makes it possible to control the lamps from other devices such as

- Remote controller
- Smartphone
- Connected assistant

2.1.3 IoT architecture

The architecture of the IoT's is split into several parts. The first layer of the structure is made of all devices such as smart lamps, smart fridges, personal assistant, temperature sensors ...

These smart devices are connected to the second layer, the IoT gateway. The latter allows the devices to be regrouped and to communicate via the Internet, if this gateway is also connected to the Internet. Thanks to this connection, it is able to contact the multiple servers of the manufacturers through which the end user can interact with their devices with mobile applications or web interfaces.

For example, when a user queries a home assistant like Alexa, the device sends a request to the network gateway[1], then the packet is transmitted to an Amazon back-end server to be processed. Once this action has proceeded, the response is sent back to the device to answer the query. This concept is illustrated by the Figure 2.1. The figure presents different types of connection within the LAN. A gateway is represented as the link between the connected devices and their corresponding servers in the Cloud in this case. This gateway can use the Wireless Fidelity (Wi-Fi), Zigbee or Bluetooth to communicate with the connected devices. Finally, the connection used to communicate with the Cloud can be 3G, Wi-Fi or LTE.

As shown on the Figure 2.2, some IoT's are able to work without the back-end interaction. Every action is treated within the LAN and to be more precise on the IoT gateway. That kind of installation could be illustrated by the "The Ikea TRÅDFRI Gateway" (see section 6.1.1). Ikea designed an IoT gateway able to handle the requests from their remote controller without the need to communicate with a back-end server [6], [7].

[1] A network gateway is the link between the LAN and the outside world

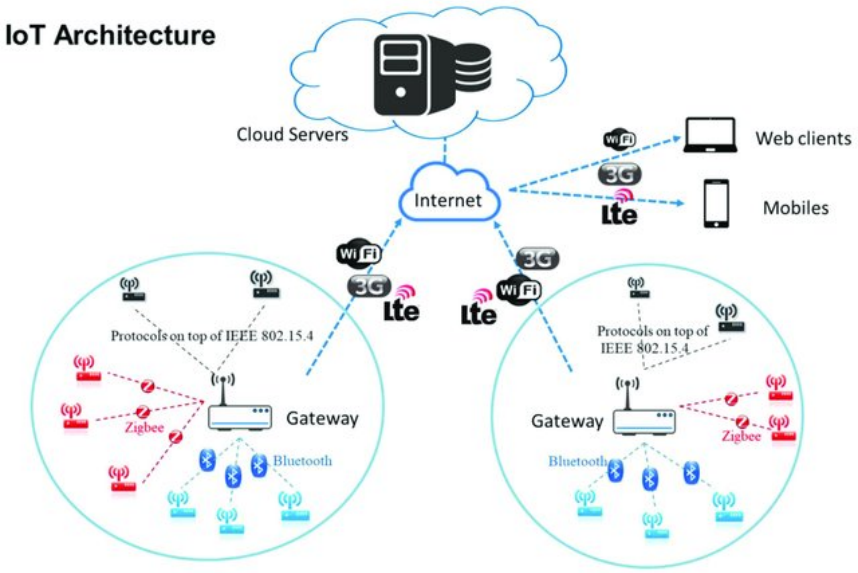

Figure 2.1: The generic architecture of the Internet of Things [5].

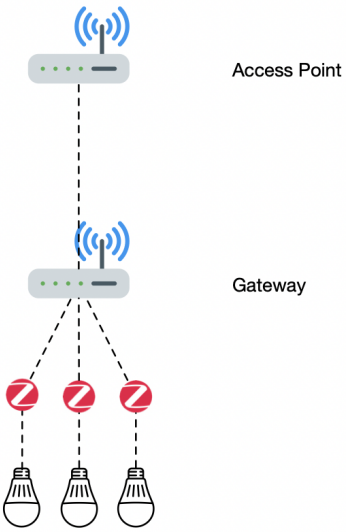

Figure 2.2: The local architecture of the Internet of Things.

The Figure 2.2 represents the communication between the IoT gateway and the connected devices. In this case, the Zigbee protocol (2.1.4) is used to enable the transmission of information between the aforementioned gateway and the IoT. However, it exists multiple protocols used for short-range communication such as[8]

- BlueTooth LE
- 6LoWPAN
- RFID
- NFC

Those protocols have a short-range transmission, from less than a metre up to one hundred metres. Regarding their data rate, it goes from around 250 kbps up to 4 Mbps [8].

Thanks to those protocols, IoT's are able to send data to the gateway and the end user is able, for example, to switch on and off the lights thanks to a local remote controller. In some cases, the lights are dimmable and it is possible to modify the light brightness via the remote controller. For the purpose of this thesis, the following IoT's have been used:

Alexa

The Amazon Alexa communicates directly through the Wi-Fi. It is connected to an Access Point (AP). It is then able to communicate with the required servers to process the requests of the user if the AP is connected to the Internet.

Ikea Tradfri

The connected lamps are linked to the gateway thanks to the Zigbee protocol.

2.2 DNS

Definition 2.2.1 DNS - RFC 1035 [9]

"*The goal of domain names is to provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets, and administrative organizations.*"

2.2.1 Principle

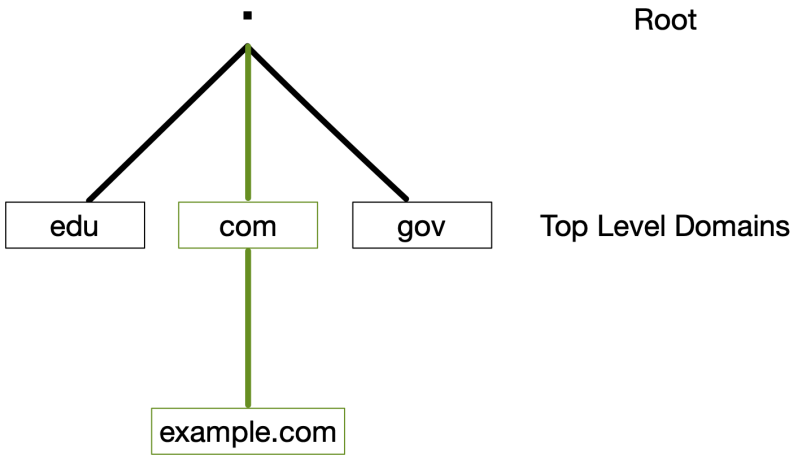

Figure 2.3: DNS hierarchy example

As shown on the Figure 2.3, DNS has been designed with a hierarchical configuration. Each domain name has multiple parts separated by dots. When a request is made, the root will be firstly interrogated if the Top Level Domain (TLD) is not cached. This one will redirect to the

TLD based on the last part of a domain name (com, gov, edu ...). Once this one has been resolved, it will resolve the other parts until it finds a match with the request made.

2.2.2 Illustration

One of the goals of DNS is to resolve hostnames into Internet Protocol (IP) addresses. In order to obtain a result, a multitude of requests are sent to different servers:

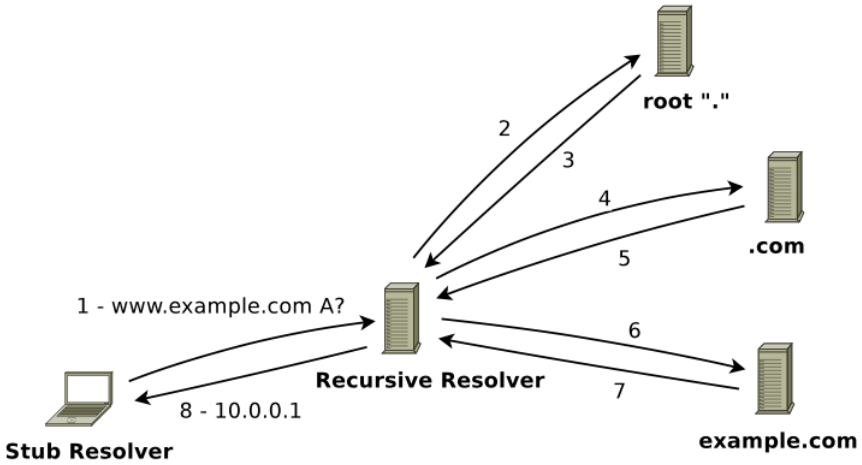

Figure 2.4: DNS request and response example

As shown on the Figure 2.4 [10], the DNS request is divided in multiple steps.
In the example above, a client wants to obtain the IP address of `example.com`. If the information is not stored into the local DNS cache of the client, the following steps are performed :

1. The client firstly contacts the Recursive DNS Resolver
2. The Recursive DNS Resolver contacts the Root Name Server if necessary.
3. The Root Name Server refers to **com** TLD Server.
4. The Recursive DNS Resolver queries the **com** TLD server.
5. The TLD refers to **example.com**.
6. The Recursive DNS Resolver queries the authoritative DNS servers for **example.com**.
7. The authoritative DNS servers send back the IP to the Recursive DNS Resolver. This one retrieves the A record for **example.com** from the authoritative name servers and stores it in cache during a certain amount of time.
8. The IP address is sent back.

In this thesis, DNS has a central place. Indeed, every IoT tested during our research performs DNS queries. Within the section 6.3, the details about the solution proposed to block some traffic based on that kind of request are exposed.

3. State of the Art

This chapter aims to present the existing research and solutions presented in several scientific papers. It is divided into three parts. The first section (see section 3.1) discusses the *IoT Gateway* and its utility. The second section (see section 3.2) addresses the traffic data and the problems raised. The last section (see section 3.3) exposes the fact that connected devices leak personal data regarding the user habits.

3.1 Gateway

Within the article "A Host based IDS and mitigation framework for Smart Home IoT using OpenFlow" [11], the authors explain that an increase in the deployment of smart cities is about to happen. At the heart of these cities, it is therefore a question of connecting any type of object to the Internet. It explains that this transformation is affecting industry and education, but also smart homes and, on a larger scale, smart cities.

The goal of the IoT is to facilitate the life of the user by working autonomously. However, as the author points out, some security issues may arise. Here are the main risks highlighted:

- "*Insufficient authentication and authorization*
- *Lack of transport encryption*
- *Insecure firmware*"

In order to avoid these risks, the authors bring a solution based on a gateway. This would be a Home-based Intrusion Detection System (IDS). The challenge is to overcome the recurrent lack of expertise and vigilance of the end user to secure a network with IoT's.
The purpose of this Host-based IDS is to be a network protection for smart devices by

- Monitoring the network
- Blocking the intruder if suspicious activity is detected

The authors of the article "Passive Inference of User Actions through IoT Gateway Encrypted Traffic Analysis" [12] have a position quite similar to the previous one. Their idea is to create a central point through which all IoT's traffic would pass. However, they point out other problems related to these connected objects and their firmware.

They highlight the lack of updates for the devices. Moreover, most of the people do not change the default credentials. This malpractice, among others, could lead to a privacy leak. That data could be related to wearable devices as written in the article, "*from the network traffic generated by Bluetooth Low Energy (BLE) wearable fitness trackers, it is possible to identify a person and its activity*".
In order to reduce the lack of privacy, the authors proposed the following solutions:

- Analysing the traffic
- IoT IP-based devices identification using header values (IP, port, protocol, payload)
- Random forest classifier to identify an IoT device from a signature
- Self-learning device type identification technique

- Privacy by design

Thanks to its central location, the gateway is able to collect, record and forward the data [13]. However, it leads to other problems such as network congestion and existence of single point of failure. This single gateway needs to process all the packets and then decide to forward it or not.

In order to perform instantaneous detection the authors of "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders" [14] propose to have a network-based approach. This would allow the system to detect and directly disconnect the devices if the latter are contaminated by a botnet. Again, the central position of the gateway allows it to analyse the traffic and block it or not. As written in this article, a host-based approach would not be realistic. Indeed, in general, the IoT's do not have so much computational power and sometimes the devices work on batteries. If such a device included a host-based detection program, this could drain the battery and affect the performances.

The methodology proposed by this article is the following:
- Data Collection
- Feature Extraction
- Training an anomaly detector
- Continuous monitoring for anomaly detection

However, in "Intrusion detection systems for IoT-based smart environments" [15], the proposed solution is a mix between the network-based and the host-based IDS. This monitors thanks to the network or the host-based sensors. Then the captured traffic is analysed to identify some patterns. The IDS is an anomaly or misuse detection system.

This article mentions the two definitions below. They also come from other articles.

"*A misuse-based intrusion detection technique uses a database of known signatures and patterns of malicious codes and intrusions to detect well-known attacks*" [15], [16]

"*In an anomaly-based intrusion detection technique, a normal data pattern is created based on data from normal users and is then compared against current data patterns in an online manner to detect anomalies.*" [15], [17]

As this new paradigm is gaining field and is present in more and more critical domains such as medicine and industry, the security is one of the priorities to ensure that everything is working properly.

The proposed solution by "IoT Security Framework for Smart Cyber Infrastructures" [18] would be the implementation of a Building Automation Systems (BAS) and a Supervisory Control And Data Acquisition (SCADA). Those systems would allow to detect anomalies that could be triggered by an attack against the sensors of the first layer.

In order to realise it, they use an IDS which is built like a database. Unfortunately, new attacks cannot all be detected via this technique. Moreover, they propose an anomaly based IDS which would define a baseline model for a "normal" behaviour. If a device goes out of this model,

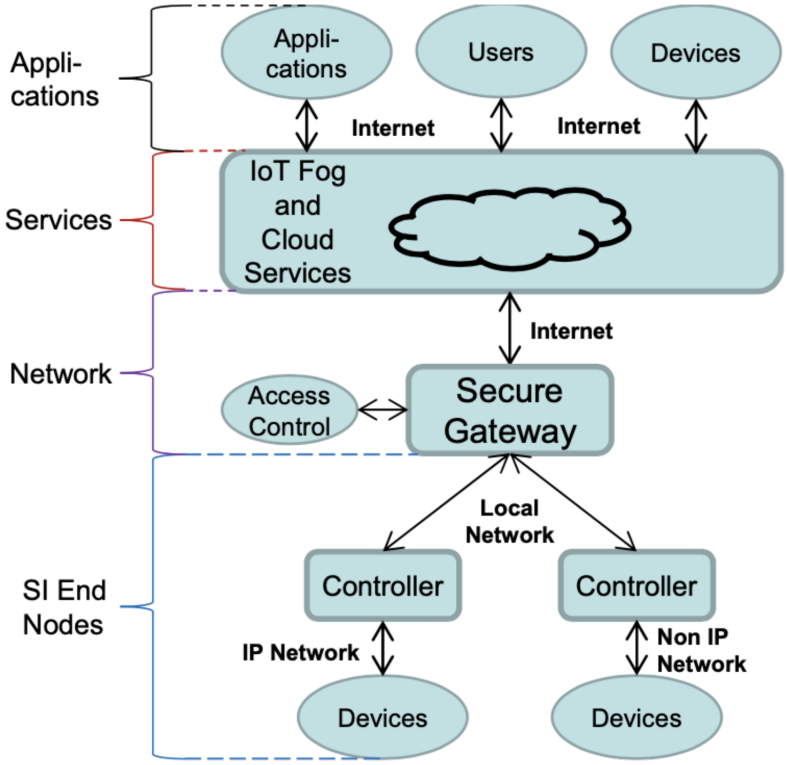

Figure 3.1: IoT security framework for Smart Infrastructures [18]

actions would be taken. The goal is obviously to mitigate potential threats.

The structure proposed in the Figure 3.1 is divided into four parts:

- The end nodes/the devices
- The network
- The services
- The Application layer

The end nodes/devices layer concerns the IoT. This is the level where the sensors and connected devices will take place. The network part is the gateway, regrouping all the devices under the same LAN. The service part is in the cloud in this case; "*Cloud Computing provides computing, storage, and applications as services that are offered on demand in a cost-effective and a scalable way. Resources can be shared among a large number of users, who can access applications and data from anywhere at any time. Fog Computing aims at providing computational power, storage, and network services to end devices. Since Fog computing host services at the network edge, its advantages include low service latency, high quality services, support for mobility, location awareness, and easier implementation of security mechanisms*" [18]. This is used in order to be cost-effective, scalable and accessible for any user at any time and anywhere.

3.2 Traffic Data

The authors of "Characterizing and classifying IoT traffic in smart cities and campuses" [19] bring an approach based on the traffic data of the devices to characterise and classify them on a network. By developing a way of classifying that learns the behaviour of an IoT, the generated traffic could lead to identification. The latter could be possible thanks to the data and some specific attributes could be extracted from the captures such as data rates, activity cycle ...

Once each device is able to be identifiable based on a "normal" behaviour at a network level, it is possible to find attacks based on the new generated traffic. It is then important to have a strong base profile for each device in order to find a "wrong" behaviour.
Based on "Peek-a-Boo: I see your smart home activities, even encrypted!" [20], the authors expose multiple problems such as:

- The encryption only hides the payload
  - Some headers are therefore leaking
- A "side channel attack" based on the number of packets sent could be performed

In order to avoid these privacy leaks, the proposed solution is to generate noise network traffic to hide the real one.
In "Design of automatic identification gateway system for different IoT devices and services" [21], the authors explain that even if it is complicated to establish, each device has a "fingerprint". The fingerprint "*is a concept that assumes that machines, like human fingerprints, have unique characteristics and can be identified through them.*" The characteristics allowing to identify the devices are the OS, the browser, the programs ...

To establish this "fingerprint", the authors present two methods:

- "*The Active method sends a network packet for direct identification to the device, analyzes the returned response packet, and obtains the OS information as an OS-specific setting value. This method is fast and accurate, but the device is aware of the attempt.*"
- "*The passive method captures traffic from the internal network or analyzes the monitored data to infer the OS information of the device to be identified. This method does not leave traces because it does not send traffic directly to the device, but it is slow if the device does not generate or generate less network traffic.*"

Once the data have been collected, this article proposes the same solution as the article "Passive Inference of User Actions through IoT Gateway Encrypted Traffic Analysis" [12]. Both of them use random forests to classify the devices based on their signature for the first one and on the fingerprint for this one.

3.3 Data Leak

In the article "Information Exposure From Consumer IoT Devices" [22], the authors stress on the fact that through all user's connected devices, the latter leak private or sensitive information.

Thanks to all connected devices, it is possible to track down the habits of the users. This has major implications regarding the privacy of the users. Some devices are able to record the audio, others can obtain the TV viewing habits, "*then share this information over the Internet with device manufacturers and unknown third parties in different countries with different privacy regulations. As most of these devices lack any interfaces that indicate information exposure,*

there is an urgent need for research that provides transparency into such exposure at scale, and that identifies relevant privacy implications within different jurisdiction" [22].

In the article, the authors found out that "*using 34,586 controlled experiments, 72/81 devices have at least one destination that is not a first party (i.e., belonging to the device manufacturer), 56% of the US devices and 83.8% of the UK devices contact destinations outside their region, all devices expose information to eavesdroppers via at least one plaintext flow, and a passive eavesdropper can reliably infer user and device behavior from the traffic (encrypted or otherwise) of 30/81 devices*" [22].

During the tests carried out by this study, they also noticed a strange behaviour regarding a connected doorbell; "*a video doorbell sends video recordings to its service provider based on movement sensors, without any notification or consent from recorded parties*" [22].

A way to avoid the data leak and the analysis of the traffic by an attacker is the solution proposed in "Securing the Insecure Link of Internet-of-Things Using Next-Generation Smart Gateway" [23]. The authors of this article expose a solution based on Virtual Private Network (VPN) to ensure the privacy. Thanks to this solution, an adversary cannot inject of modify the packet. Each tunnel uses encryption and authentication mechanism to protect the data. Moreover, "*Malicious VPN client cannot identify the traffic type of legitimate clients*" [23].

3.4 Summary

Each line of the Table 3.1 represents an article or a survey read during the realisation of this master thesis. The column named *Identified Problems* discusses the different problems identified within the materials. The last columns highlights the solutions proposed by the authors to resolve the raised problems.

	Identified Problems	Solutions
A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow [11]	• Security exposure caused by – Insufficient authentication and authorisation – Lack of transport encryption – Insecure firmware – Lack of user expertise	• Host-based IDS – Network protection for smart devices – Monitor the network – The intruder would be blocked if suspicious activity is detected • Machine learning techniques used for detection based on learned signature pattern or known attacks
Passive Inference of User Actions through IoT Gateway Encrypted Traffic Analysis [12]	• Lack of updates • Use of default credentials • Privacy leakage – IoT can be used to identify a person – Network traffic generated by wearables – Reveal user activity	• Analysing the traffic • Random forest classifier to identify an IoT device from a signature • Self-learning device type identification technique • Behavioural fingerprint based on header and payload based features
N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders [14]	- Proliferation of botnets	• Network based approach – Deep learning to perform anomaly detection • Host based approach – Less realistic for IoT
Intrusion detection systems for IoT-based smart environments: a survey [15]	• IoT's are subject to various attacks : DoS, DDoS • Security of IoT is a serious issue due to the increasing numbers of services and users in IoT networks	• IDS implementation – Misuse-based intrusion detection – Anomaly-based intrusion detection
IoT Security Framework for Smart Cyber Infrastructures [18]	• Computing platform, constraints: – memory and processing capability – may not support complex security algorithms • Some devices and services may be shared with different ownership, policy, and connectivity domains	• BAS and SCADA – Detect anomalies against sensors at the first layer • IDS – Signature based – Anomaly Based
Characterizing and classifying IoT traffic in smart cities and campuses [19]	• Big infrastructure could not be aware of the actual devices on the campus • These devices could be under attack or not working properly	• Analyse the traffic to characterise statistical attributes such as activity cycles, signalling patterns ... • Identify IoT to detect anomalous behaviour
Peek-a-Boo : I see your smart home activities, even encrypted [20]	• Encryption only hide the payload • Attack by observing passively the wireless traffic from smart home devices • "Side channel attack" based on the number of packets sent	• Machine learning to detect and identify particular types of IoT devices, their actions, states and ongoing user activities by only observing the wireless traffic • Protect privacy leakage : generate spoofed network traffic to hide the real activities
Design of automatic identification gateway systems for different IoT devices and services [21]	• Not easy to create fingerprint information for all devices as new devices continue to emerge.	• Automatic identification of the IoT devices in a gateway thanks to machine learning based on fingerprint analysis method
Information Exposure From Consumer IoT Devices [22]	• Private and/or sensitive information exposure	• US (enforced by the FTC) and UK (GDPR) – Impact on the data collection
Securing the Insecure Link of Internet-of-Things Using Next-Generation Smart Gateway [23]	• Unsecure link between the IoT and the cloud	• VPN to ensure privacy

Table 3.1: Summary of the State of the Art

4. Identified problems

During the reading of the articles to compose the chapter 3, several problems were identified.

The first problem is related to the traffic data. Indeed, as written into the section 3.2, based on the amount of packets and their content, it is possible to create a fingerprint for each connected device. Moreover, some devices do not send all their communication in an encrypted way or those devices do not communicate only with their constructors. All of the actions performed thanks to the connected devices should be considered as private information. Moreover, it is not because the end user does not realise what is really happening behind the scene that the constructors are authorised to exploit that and take advantage of those users.

The second problem is the way to handle it. Indeed, as presented in chapter 3, if the goal is to prevent the manufacturer from using the data in order to make some statistics and prevent an attacker from analysing the traffic data, it is necessary to create a filter before the packets go out of the LAN. If the proposed solution allows having such a filter to prevent devices from querying some unessential domains, this would reduce the capability to fingerprint the connected device and therefore lower the “data leakage”.

Another problem exposed in the chapter 3 is the fact that even if the traffic is encrypted, it is possible to infer information from the amount of packets. Moreover, the encryption on the packet still leaks some headers.

As written into “A host-based intrusion detection and mitigation framework for smart home iot using openflow” [11], a recurrent problem within the IoT’s is the insufficient authentication and authorisation and insecure firmware. That kind of problem added to the unmastering of the user regarding the security of the IoT’s, can lead to dangerous situations such as data leakage, botnet ... The mission of the manufacturer should be to ensure security and regular update to protect the consumers.

Based on the first two exposed problems, the idea of creating a filtering gateway where the IoT’s would connect was born. This gateway is an access point in order to connect the devices thanks to the Wi-Fi and its goal is to filter the queries to be sure that none of the IoT’s sends packets when it should not.

5. Concept

In this chapter, the different concepts that were implemented are discussed. The DNS filter will be firstly presented, followed by the Parent-child principle. The ubiquity of the IoT's implies the presence of a lot of devices at home. Sometimes it appears that several devices depend on each other. Therefore, some of them want to communicate with the outside world and it is not always required. Indeed, some devices send too much data. The latter can be exploited by the manufacturers to profile the users or make statistics. As explained into the chapter 4, even encrypted, it is possible to infer information from the traffic data. Finally, we will present an attack that this thesis concept should theoretically prevent.

5.1 DNS filtering

In order to reduce the unnecessary traffic data, the concept of a DNS (see section 2.2) filter was firstly imagined.

The idea of a DNS filter is to check all the requests on the port 53 and block the packets with a non-whitelisted domain. This process enables a check on the contacted domains in order to allow only the trusted ones.

If the domain is blacklisted, the packet should be dropped. However, if the domain is whitelisted, the packet should be accepted in order to query the recursive resolver (see Figure 2.4). If this packet is transmitted to the resolver, the latter will at the end return the corresponding IP address where the searched resources are located.

5.2 Parent-child

Some IoT's could be "dependent" from the others. For example, the connected lights (see section 6.1.1) can be controlled by the home assistant (for example, Alexa (see section 6.1.1)) if they are connected together. Therefore, it is interesting to treat them in a way to show up the link between the devices.

The Figure 5.1 shows the number of frames arising from the Ikea Gateway and the Alexa home assistant. The red trace corresponds to the Alexa device and the blue trace is the Ikea gateway network traffic. Based on the following figure, it is quite clear to figure out the link between the two lines.

Each time the home assistant is triggered to switch on or off the lights, the red trace has a peak over 100 frames. Then a peak on the blue trace can be noticed. In this specific case, the home assistant would be the parent and the Ikea Gateway would be the child. Therefore, it would be interesting to exploit this relationship between the two of them to reduce the number of DNS requests sent.

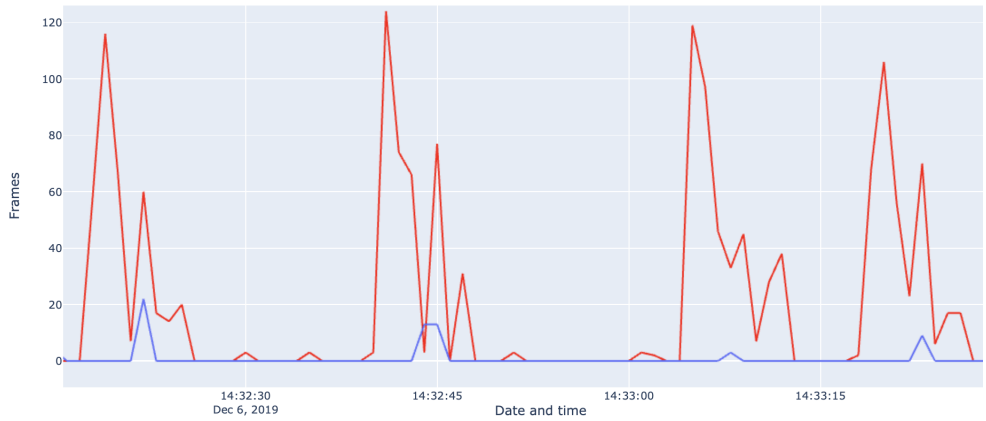

Figure 5.1: User requests extracted from the treatment of the pcap file from the 06/12/2019

The Figure 5.2 stands out the possible link between the different devices. If the connected lamps are controlled by a home assistant, it seems clear that the lamps do not need to send information when the user does not ask the assistant to switch on/off the lamps.

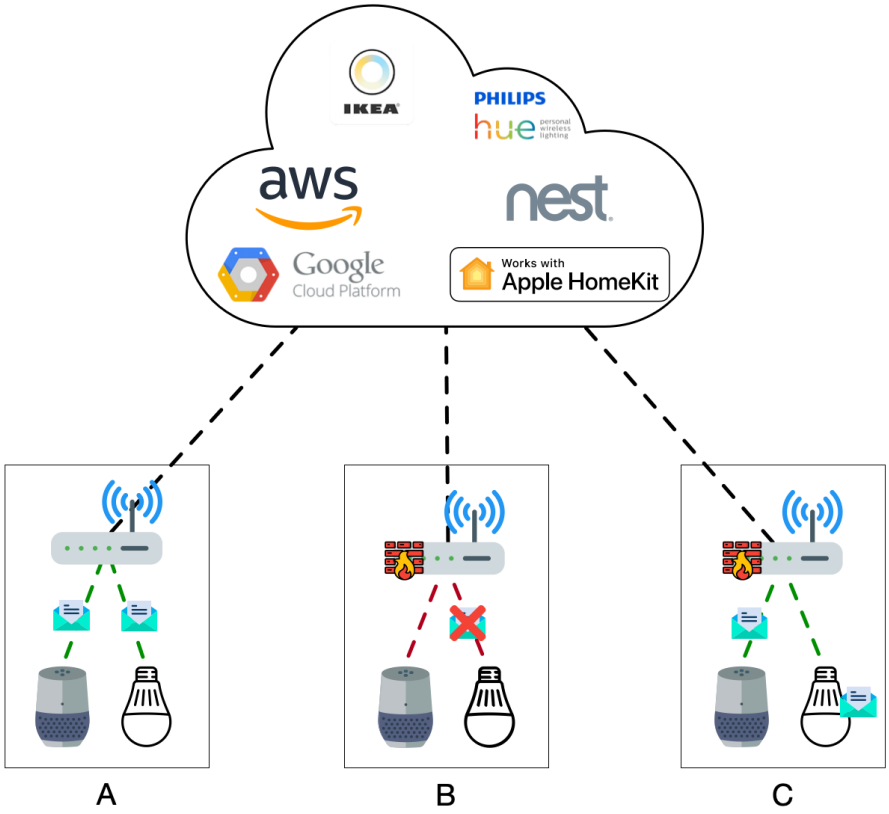

Figure 5.2: Figure showing the dependency between the devices

In the **A** scenario, the access point has no parent child implementation. Therefore, the home assistant and the connected lamp can send packets as they wish.

The access point presented in the scenarios **B** and **C** are equipped with the parent-child concept. It is worth noting in both cases, the parent is the home assistant and the child is the lamp. Therefore, in the scenario **B** the parent is not sending DNS requests so, the child is not able to send a request either.

In the scenario **C**, the parent is sending DNS requests. Therefore, the child is allowed to send data during a defined time window.

This principle allows to avoid sending DNS requests without interaction from the "parent". Thanks to that concept, the data leakage is lower and only the necessary packets go out. To prevent the child from contacting unnecessary domains and good domains at unnecessary time, this traffic is also filtered as presented in the section 5.1.

As shown on the Figure 5.2, the parent-child principle is hosted on a gateway (see section 3.1). The goal of this principle is to decrease the amount of DNS requests sent when a user is not asking to perform an action. This principle has a significant influence on the data leak of the "child" devices as they are not able to send DNS requests when the parent does not.

5.3 Attack

DNS tunnelling

Within the article "On the viability and performance of DNS tunneling" [24], the DNS tunnelling is defined as follows

Definition 5.3.1 DNS tunnelling

"*DNS tunnels are network covert channels that allow the transmission of arbitrary data using the DNS infrastructure. Users can use such tunnels to hide their communication sessions in order to bypass local security and accounting policies.*" [24]

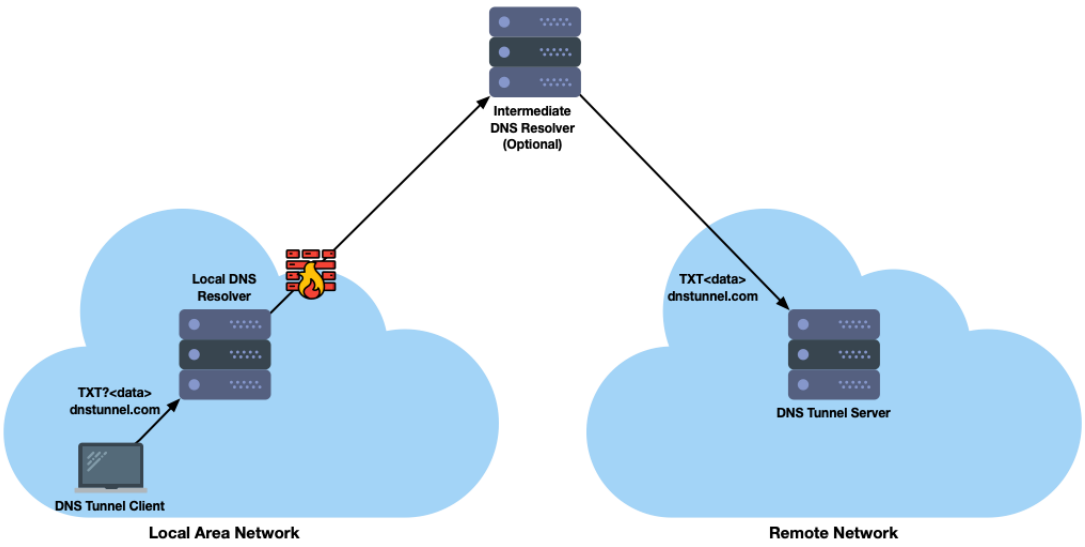

Figure 5.3: DNS Tunnelling example with TXT record

As presented on the Figure 5.3, the DNS tunnelling uses the DNS packets to exfiltrate data without being noticed.

Thanks to the whitelisting, the DNS tunnelling cannot happen. Indeed, as the user has to select the domains allowed to be contacted, the other ones are dropped by default. As a result, an attacker is not able to reach their domain if the user did not whitelist it. The filtering applied on the domains exposed in section 5.1 is strict. This means that there is no wildcard for all the domains and subdomains. The permission is granted only for a complete domain. For example, if "`metric.amazon.com`" is authorised, "`amazon.com`" and "`test.metric.amazon.com`" are not necessarily authorised.

6. Proof of concept

The objective of this chapter is to present the different parts of the *Proof of concept*. The section 6.1 addresses the configurations to clarify the environment where the tests were performed. The section 6.2 aims to present the methodology followed to capture the data and then analyse it. The section 6.3 is dedicated to the experiment with the explanations of the code and the several tools created. In this section, the results and limitations are presented at the end.

6.1 Configurations

The goal of this section 6.1 is to present the different configurations and the devices used during the next phases. Each device has a list of its network, hardware and software specifications if applicable. It has been documented in order to be able to reproduce the experiment in the same conditions.

6.1.1 Network, hardware and software configuration

Firewall

The environment where the tests were performed is behind a firewall. A firewall is a kind of filter. This filter will check all the packets sent or received by the machines of the network. Based on the encoded rules, it will discard or allow the packets to go through.
The operating system running on this home-based firewall is pfSense[1]. "*pfSense is a stateful firewall, which means that it remembers information about connections flowing through the firewall so that reply traffic can be allowed automatically. This data is retained in the State Table. The connection information in the state table includes the source, the destination, the protocol, the ports, and more: Enough to identify only a specific connection.*" [25]
In order to not bias the results, a "permit any" rule has been applied to Ikea and Alexa onto the firewall.

```
IP:                192.169.0.254
```

Listing 6.1: Network details

```
Version:           2.4.4-RELEASE-p3 (amd64)
Release:           FreeBSD 11.2-RELEASE-p10
built:             Wed May 15 18:53:44 EDT 2019
```

Listing 6.2: Software details

[1] https://docs.netgate.com/pfsense/en/latest/book/firewall/firewall-fundamentals.html. Visited on 07/03/2020

Linksys network switch

A network switch is a kind of controller allowing the transfer of the packet from one port to another. This is crucial to let the devices communicate to each other or send data to the firewall and then to the Internet. The particularity of this switch is that it is manageable. This means that we are able to access it via its IP and create VLANs, to do port mirroring ... As explained later in the subsection 6.2.1, the port mirroring was necessary to capture the data. This requirement guided the choice to work with this type of device[2].

```
IP:              192.168.0.253
```

Listing 6.3: Network detail

```
Firmware Version:    1.1.1.9
Boot Code Version:   1.0.0.5
System Description:  LGS308 8-Port Gigabit Smart Switch
```

Listing 6.4: Software details

```
Hardware Version:    V02
```

Listing 6.5: Hardware detail

Raspberry pi

A Raspberry Pi[3] is a small computer based on an ARM processor. The first one was commercialised in February 29^{th}, 2012. Some models come without Wi-Fi connector or without standardised ports such as USB or Ethernet interface. The most recent one can embed 8GB RAM, 1.5 GHz quad-core CPU, Ethernet interface, USB ... To lead this proof of concept, a Raspberry Pi 3B was used. The main reasons for this choice are the following:

- Cheap
- Portable
- Low power consumption

There are two configurations for this device. During the different phases of the experimentation, the configuration has been changed for a better fitting of the requirements.

Raspberry pi - Monitoring

During the monitoring phase, Raspbian[4] has been installed on the Raspberry Pi.

[2] https://www.ldlc.com/fr-be/fiche/PB00183262.html. Visited on 18/05/2020
[3] https://www.raspberrypi.org/. Visited on 18/05/2020
[4] https://www.raspberrypi.org/downloads/raspbian/. Visited on 18/05/2020

```
IP:                  192.168.0.243
```

Listing 6.6: Network detail

```
VERSION_ID:         "10"
VERSION:            "10 (buster)"
NAME:               "Raspbian GNU/Linux"
PRETTY_NAME:        "Raspbian GNU/Linux 10 (buster)"
```

Listing 6.7: Software details

```
Raspberry pi version:     3B V1.2
Manufacturing:            2015
Wifi module:              Yes
```

Listing 6.8: Hardware details

```
SHA256      22a9a725288aeb7f3e7b9cda3be3c1887639e5aae3bf529b18c9011a63cfa471
```

Listing 6.9: Hash of 2020-02-05-raspbian-buster.img

Raspberry pi - Experiment

Once the capture of the data done, the Raspberry pi is reflashed with the same image (6.1.1). Different packages were installed such as `tcpdump`. Moreover RaspAp[5] was installed and configured to create an access point with the Raspberry in order to connect the different devices via Wi-Fi.

Ikea Gateway

The Ikea Gateway[6] is used to communicate with the Ikea IoT's contained in the LAN of the user. This gateway communicates with a Smart Light Bulb or a group of Smart Light Bulb via the Zigbee protocol [26]. Zigbee is a high-level protocol enabling the communication between devices equipped with a radio transceiver. It is often used in IoT because this protocol is energy-efficient [27].

[5] `https://raspap.com/`. Visited on 18/05/2020

[6] `https://www.ikea.com/be/fr/p/tradfri-passerelle-blanc-40337806/`. Visited on 18/05/2020

```
IP:              192.168.0.250
```

Listing 6.10: Network details

Alexa

The Echo dot or as called here, Alexa, is a voice-controlled smart speaker. The uses are varied, it is possible to ask it to play music, to know the weather of the day or even to control connected devices such as Ikea lamps. There are several models proposed by Amazon. The model used during the different phases (data capture (6.2.1), data analysis (6.2.2), experiment (6.3)) is the Echo dot third generation[7].

```
IP:              192.168.0.249
```

Listing 6.11: Network details

Tools

Several tools have been used during the realisation of this proof of concept. Here is a list with the version for each of them.

```
Scapy             2.4.3
Python            3.7.3
Django            3.0.4
tcpdump           4.9.3
iptables          1.8.2
NetfilterQueue    0.8.1
```

Listing 6.12: Tools and versions

[7] https://www.amazon.fr/echo-dot-3eme-generation-enceinte-connectee-avec-alexa-tissu-a dp/B07PHPXHQS/ref=sr_1_1?keywords=echo+dot&qid=1583662745&sr=8-1. Visited on 18/05/2020

6.2 Methodology

In this section 6.2, the methodology to capture the data will be presented. Then, we will explain the data analysis to extract some key points for the continuation of this work.

The following Figure 6.1 is a part of the Figure A.1 poster which was produced in collaboration with Professor Sadre as part of the UCLouvain poster session. This extract shows the way we proceeded to monitor, capture and exploit the data generated by the IoT's.

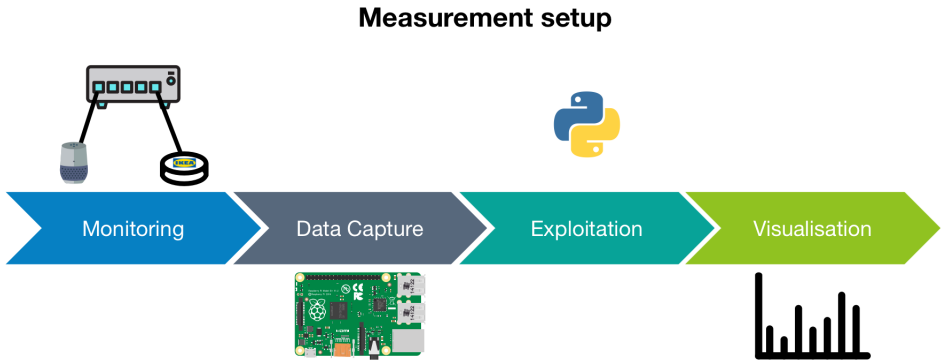

Figure 6.1: Methodology - Extract from the poster exposed at the UCL poster session

6.2.1 Data Capture

As shown on the Figure 6.1, the capture of the traffic data is the first step after the configuration of the network to monitor the traffic. The Raspberry is configured to be able to sniff on the Ethernet port and still have a Secure Shell (SSH) access on the device thanks to the wireless connection. Moreover, as the whole traffic of the network was not interesting for the later analysis, it was therefore important to retrieve the data only from the concerned devices.

The Raspberry (see section 6.1.1) has been set up to sniff some devices and ignore the other ones based on the IP addresses.

In addition to the setup of the Raspberry, a manageable switch (see section 6.1.1) has been configured to apply a port mirroring and be able to retrieve the incoming and outgoing packets.

As represented on Figure 6.2, the port mirroring allows to obtain a copy of the message sent from the computer A to the computer B on the port indicated as the destination port for the copy. The same process will occur when the computer B sends a message to the computer A. All communications from and to this port are then forwarded to the mirror port. The Figure 6.3 is the implementaton of the port mirroring on the manageable switch (see section 6.1.1).

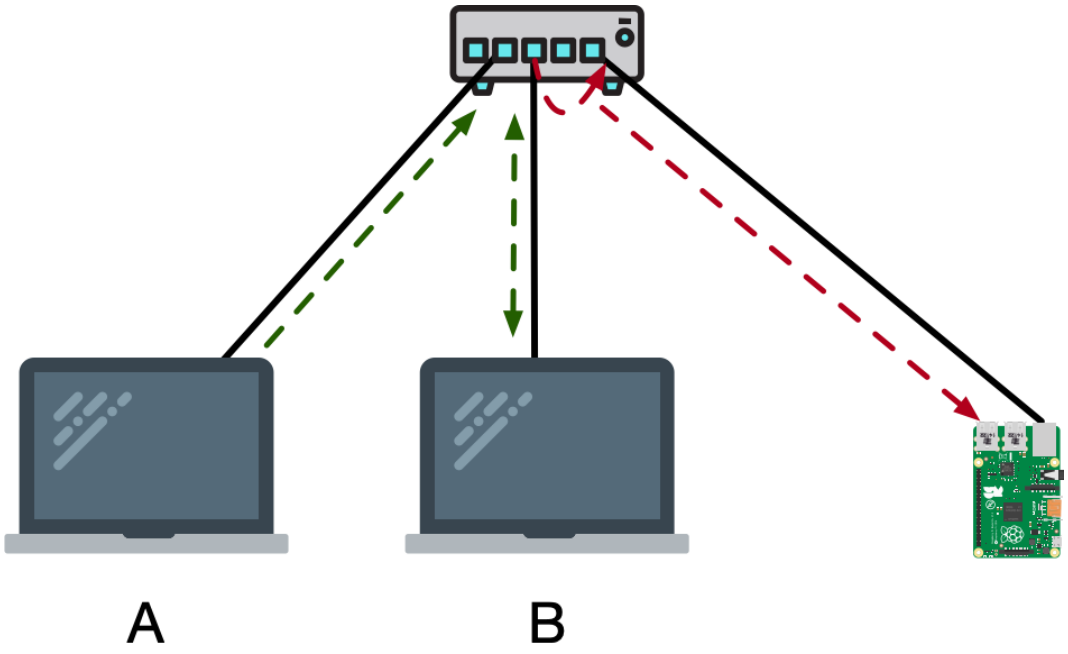

Figure 6.2: Illustration of Port Mirroring

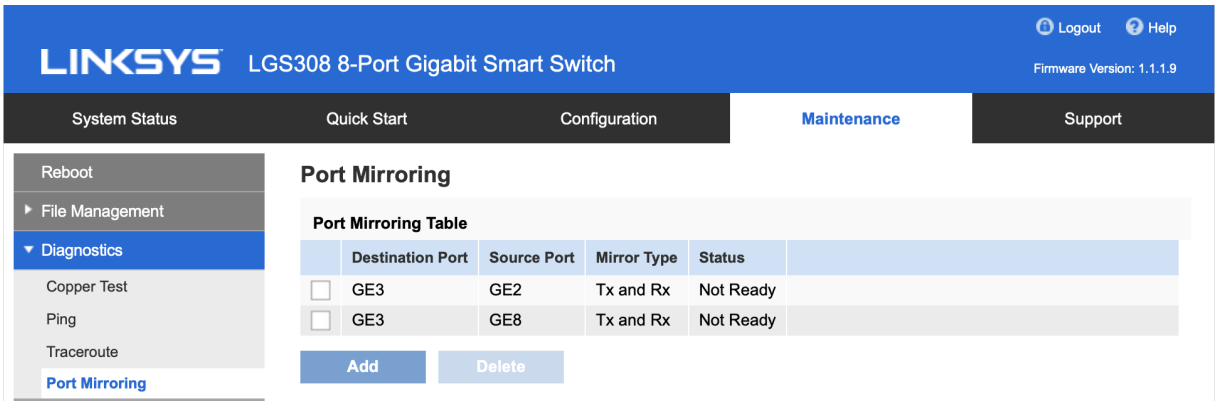

Figure 6.3: Port mirroring encoded on the manageable switch

On the Figure 6.3, the first line in the “Port Monitoring Table” corresponds to the rule for the port mirroring used to copy from the port where the Access Point (AP) is plugged to the port where the Raspberry Pi (see section 6.1.1) is plugged. The second line represents the port mirroring used to copy the incoming and outgoing transmissions of the Ikea TRÅDFRI Gateway (see section 6.1.1).

The capture of the network traffic was operated thanks to `tcpdump`. It is a powerful command line tool, used in this case to capture the traffic. However, it can also be used to display the content of the packets sent or received by the computer on its network. The following commands were used to capture the data coming to the third port of the manageable switch:

- The first line corresponds to the capture of the data coming from the Alexa home assistant (see section 6.1.1).
- The second line is defined to capture the data coming from the Ikea TRÅDFRI Gateway (see section 6.1.1).

```
tcpdump net 192.168.0.249 -w /mnt/thesis/Alexa/alexa_capture.pcap
tcpdump net 192.168.0.250 -w /mnt/thesis/Ikea/ikea_capture.pcap
```

The commands above store the incoming and outgoing packets of the devices in a pcap file on an external drive mounted on the Raspberry Pi. As the purpose of those data is to be analysed, it is important to store the data in a useful and standardised format. It is easier to process this data using different Python libraries.

6.2.2 Data Analysis

The data analyis is crucial. This step guided the research in order to know what was necessary to implement to respond to identified problems (see chapter 4).

The main tool used to process the pcap file was Python3. Several libraries were used to do so.

- pandas
- plotly.express
- plotly.graph_object

Plots were created thanks to scripts and the following result was obtained:

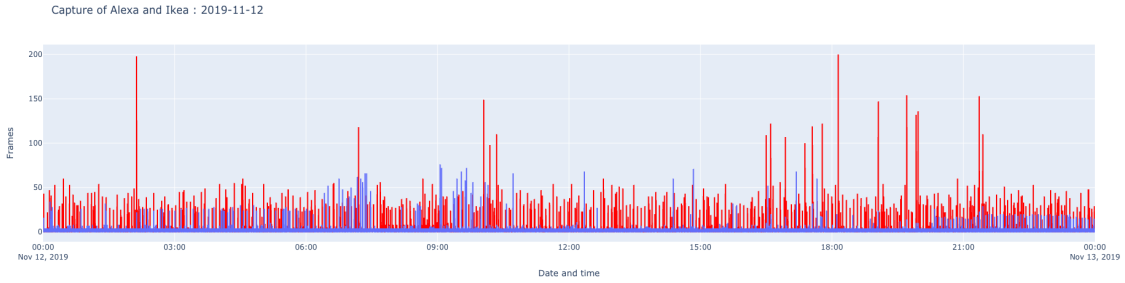

Figure 6.4: Visualisation of the content of the pcap file from the 12/11/2019

On the Figure 6.4, the red trace corresponds to the network traffic of the Alexa device. The blue trace is the Ikea device network traffic capture. This graph represents all the packets sent and received by the devices in a period of one day. The traces have been both captured on the switch thanks to the port mirroring (see Figure 6.3) and saved on an external hard drive mounted on the Raspberry Pi (see section 6.1.1).

Based on the several graphs created thanks to the pcap files we can notice that the connected devices "speak" very much compared to the number of requests executed by the user during the day. Moreover, after comparing the different traces, it appears that some data peaks systematically show up at the same times when no queries are proceeded.

After a few days, we noticed that it would be much more relevant to store the captured data in one file per day. Thanks to this classification and the notes taken during the day to know when the users were doing some requests to the devices, it was much easier to target and visualise where the user requests were and where the requests made by the device were without the interaction of the end user.

6.2.2.1 Users' requests

Shape

During the data collection, we have, in parallel, recorded all the interactions done by the end users with Alexa or Ikea lamps. This allowed us to establish a kind of grid with the device used, the command done (example: turn on the lamps, turn off the lamps, turn on the plug ...). This grid was then used in order to find easily on the graph the places where requests were made by the user and not by the device without the command of the user.
Here is an excerpt from a trace harvested on the 06^{th} December 2019. On the latter, we have isolated a few users' interactions with the connected lamps. The red trace corresponds to the Alexa device and the blue trace is the Ikea gateway network traffic. A request is a set of frames. On the Figure 6.5, four requests from Alexa can be easily noticed.

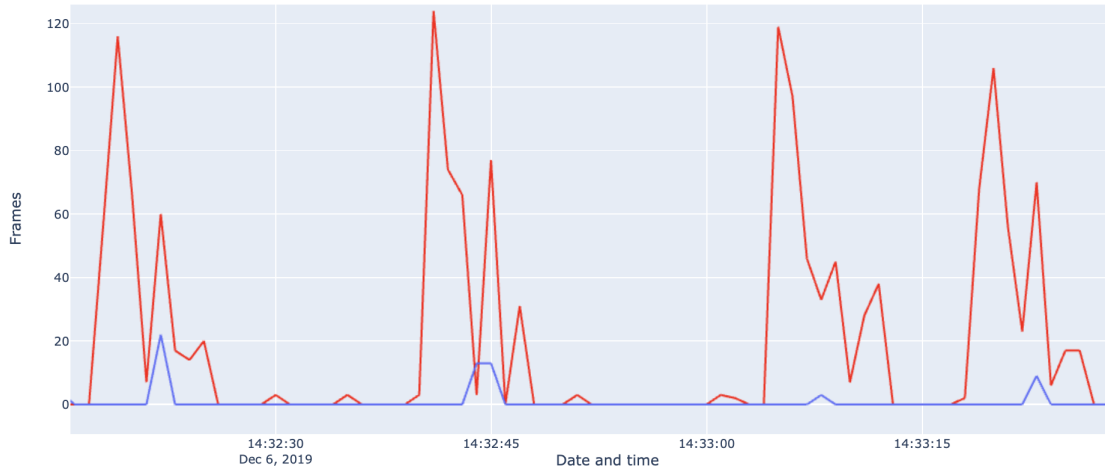

Figure 6.5: User requests extracted from the treatment of the pcap file from the 06/12/2019

Timing

In order to calculate the elapsed time between the beginning of an Alexa request and the end of the Ikea one, a script has been crafted. Thanks to the latter, an upper bound and a lower bound as well as the mean time and the median could be calculated. Once those values were calculated, the following box plots were drawn

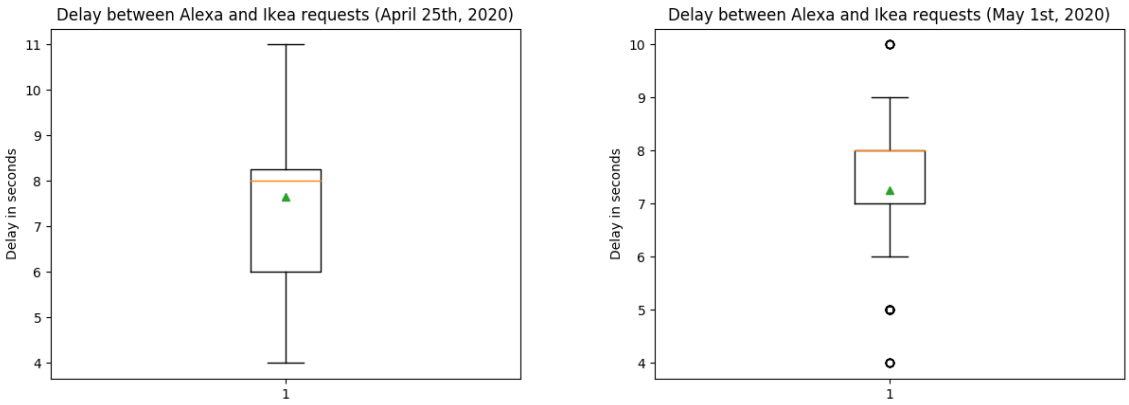

Figure 6.6: Box plots representing the time between the start of a request from Alexa and the end of a request from Ikea

	25/04/2020	01/05/2020
Mean	7.644736842105263	7.258503401360544
Median	8.0	8.0
Upper bound	11	9
Lower bound	4	6

Table 6.1: Key times calculated from the capture (timing is expressed in seconds)

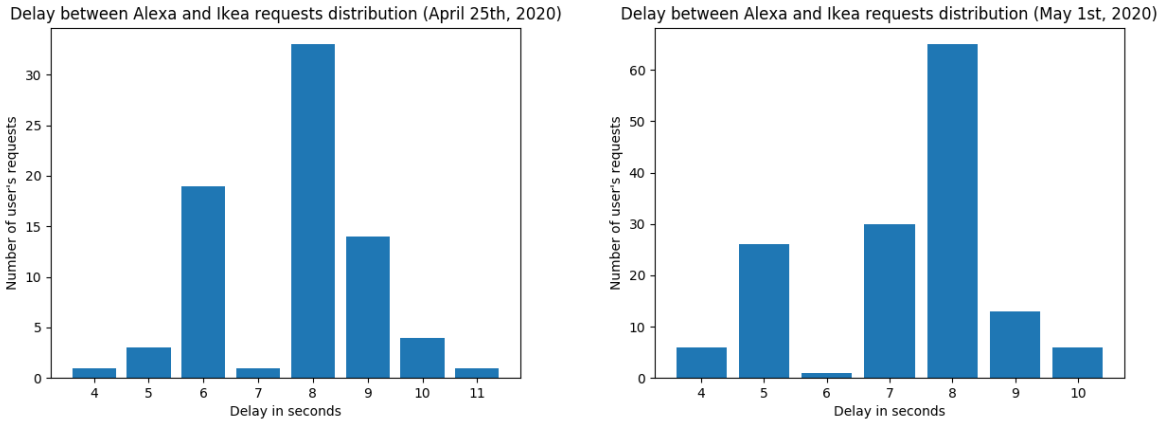

Figure 6.7: Distribution representing the time between the start of a request from Alexa and the end of a request from Ikea

The start of the request for Alexa was set thanks to the script used to ask Alexa to light up the room. The script computing the statistics sets the end of a request whenever the number of frames is equal to 0. In order to find the end of the Ikea device request, the same script checks backward from the end of the Alexa's request until finding traffic from Ikea. If the end of Ikea's requests ends after the end of Alexa's traffic, the script moves forward until the end of Ikea's traffic.

The figures 6.6, 6.7 and Table 6.1 were generated from 76 requests done in a controlled environment on the 25^{th} April 2020.

The others were generated from 147 requests captured on the 1^{st} May 2020. The results are quite similar. The medians are equal in both test cases and the means are closed from each other.

It is worth noting that to obtain more than 90% of reliability, the threshold should be set at 10 seconds.

Based on the first capture, shown on the leftmost part of the figures 6.6 and 6.7, if the threshold is equals to 9 seconds, the percentage of reliability is at most 75%. However, if it is set to 10 seconds, it goes up to 93,42%. These results are the cumulative sum of all user requests made in less than 9 seconds for the first percentage or 10 seconds for the second one.

Based on the second capture, shown on the rightmost part of the figures 6.6 and 6.7, which is larger than the first one, if the threshold is equals to 9 seconds, the percentage of reliability is at most 87,07%. However, if it is set to 10 seconds, it goes up to 95,91%. These results are the cumulative sum of all user requests made in less than 9 seconds for the first percentage or 10 seconds for the second one.

Two approaches can be imagined. The first one takes into account the user's perspective and takes care of the comfort. In this case, the threshold for the time window should be increased to 12 seconds to ensure that all the user's requests would be correctly treated. However, this approach is in opposition to the spirit of this master thesis. If the time window is that big, it means that more DNS requests are able to go out of the LAN. This means that not only the necessary ones, but also other requests could be sent in this time window if the domain is not blacklisted. A second approach is to ensure in most cases a correct usage of the devices but sometimes, if the delay is too long, the user's request would unfortunately be dropped. Thanks to this proposition, most of the requests will be performed and only the slowest ones will be discarded. Moreover, this allows to control and reduce the number of undesired requests that would appear after the child has performed its task.

6.2.2.2 Server Name Identification

Within the article "Implicit SSL certificate management without server name indication (SNI)" [28], Server Name Identification is defined as follows

Definition 6.2.1 Server Name Indication
"*Server Name Indication (SNI) is provided by TLS extensions, which allow clients to provide the name of the server they are contacting in the Client Hello message. The hostname contains the fully qualified DNS hostname of the server. This functionality is desirable, for example, to*

facilitate secure connections to servers that host multiple virtual servers at a single underlying network address."

The extraction of the different Server Name Identification (SNI) on a few pcap files captured (see subsection 6.2.1) via the Raspberry gave the following results.

Amazon Alexa

- device-artifacts-v2.s3.amazonaws.com
- wl.amazon-dss.com
- api.amazon.com
- api.amazonalexa.com
- dxz5jxhrrzigf.cloudfront.net
- device-metrics-us.amazon.com
- arcus-uswest.amazon.com
- softwareupdates.amazon.com
- unagi-na.amazon.com
- dcape-na.amazon.com

Ikea Gateway

- webhook.logentries.com

6.2.2.3 Patterns

Alexa Request pattern

As shown in the Figure 6.5, the requests made by Alexa are easily identifiable. The first part of a request shown on the graph always starts with a fairly high peak. This is followed by two other smaller peaks. This pattern seems to be systematic when Alexa is asked to turn on or off the lamp.

Daily pattern

After a few days of capture, some Alexa's "behavioural" patterns were starting to appear on a daily basis. The Figure 6.8 and Figure 6.9 represent two days of capture with the similar points that appeared.

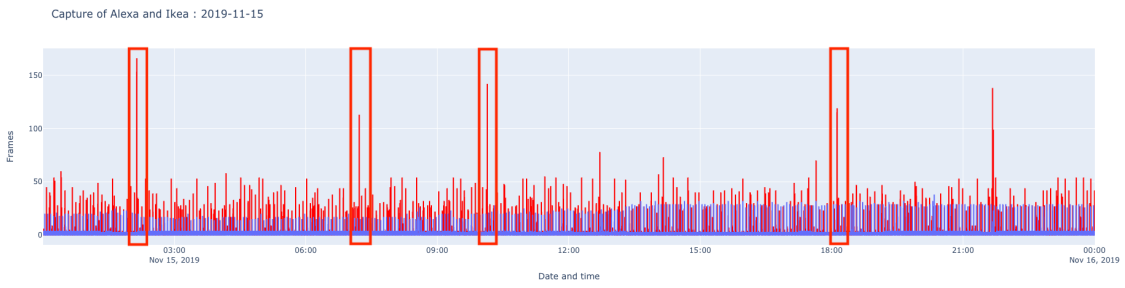

Figure 6.8: Visualisation of the content of the pcap file from the 15/11/2019

On the Figure 6.8 and Figure 6.9 the legend is still the same. The red trace corresponds to the Alexa device and the blue trace is the Ikea gateway network traffic.
The following facts can be noticed:

- Around 2:00 AM as shown on the Figure 6.8 and Figure 6.9 Alexa sends a huge amount of frames. In comparison with time when it does not respond to a query, what it is doing is

intriguing. In this case, it sends around one hundred and fifty frames every day at this time.
- A behaviour similar to the previous one also appears around 7:00 AM.
- Then at 10:00 AM
- And finally around 6:00 PM

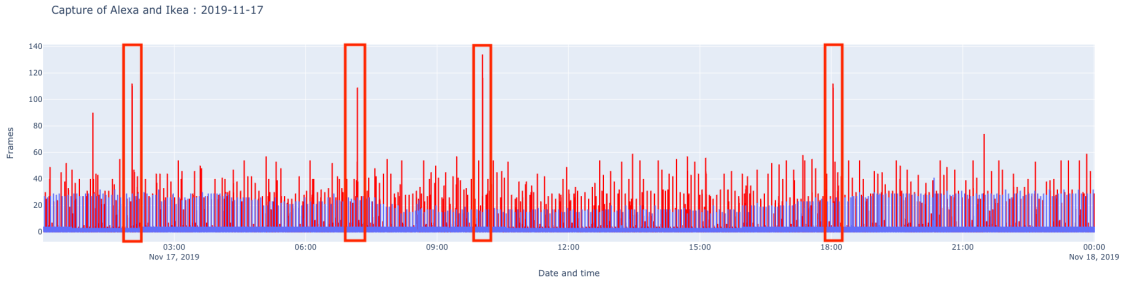

Figure 6.9: Visualisation of the content of the pcap file from the 17/11/2019

Within these four moments that seem to be repeated every day, we could have misunderstood and thought that numbers 2, 3, 4 were actually user interactions. But as shown previously, when a user is making a request, a clear pattern is displayed in the graphs corresponding to the incoming and outgoing packets. On the other hand, here is the shape of the 7:00 AM peak :

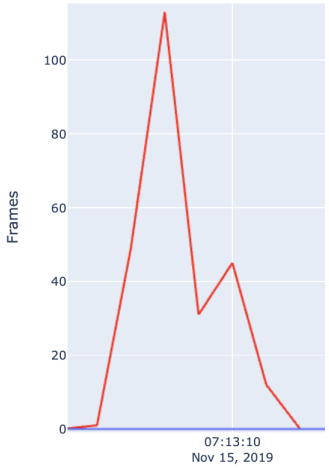

Figure 6.10: Pattern of the seven o'clock peak on the 15/11/2019

The SNIs contacted at this time are the following:
- dcape-na.amazon.com
- dxz5jxhrrzigf.cloudfront.net
- device-metrics-us.amazon.com

whereas the SNIs contacted during the requests presented on the Figure 6.5 which are queries made by a user are the following:
- unagi-na.amazon.com
- device-metrics-us.amazon.com

6.2.3 Conclusions

Based on the facts found and presented in the subsection 6.2.2, it appears that some devices send a lot of requests even when they should not. Indeed, based on the Figure 6.8 and Figure 6.9, we could remark that several SNI were contacted when the user did not request anything. It is therefore interesting to implement the concepts explained within the chapter 5 in order to reduce the number of contacted domains and the moments when they are contacted. Indeed, as some devices are dependent on each other, if the "parent" is not sending DNS requests, the child should not either.

6.3 Experiment

This section 6.3 will discuss the several steps and the implementation accomplished during the experiment. The last part of this chapter will be dedicated to the results (see subsection 6.3.4) emanating from this implementation and finally the limitations.

6.3.1 Sniffer

The first step of this proof of concept is being able to sniff all the DNS requests and store it within a database. In order to save it, the following script has been written:

```
1  import os
2  from subprocess import Popen, PIPE
3  from db_actions import DatabaseAction
4
5  def main():
6      conn = DatabaseAction().create_connection("../UCLThesis/db.sqlite3")
7
8      process = Popen('sudo tcpdump -n -l -i wlan0 port 53 --immediate-mode',
       bufsize=1, universal_newlines=True, shell=True, stdout=PIPE, stderr=
      PIPE)
9
10     while True:
11         currentLine = process.stdout.readline().split()
12
13         if(currentLine[6] == "A?"):
14             currentIP = os.path.splitext(currentLine[2])[0]
15             currentDeviceId = DatabaseAction().searchDeviceByIp(conn,
      currentIP)
16
17             if(currentDeviceId == -1 and not (".domain" in currentLine[2]))
      :
18                 DatabaseAction().insertDevice(conn, "", currentIP)
19                 deviceId = DatabaseAction().searchDeviceByIp(conn,
      currentIP)
20
21                 addDomain(conn, currentIP, deviceId)
22             elif not (".domain" in currentLine[2]):
23                 domains =  DatabaseAction().searchDomainByDomainAndDevice(
      conn, currentLine[7], currentDeviceId)
24                 if(len(domains) == 0):
25                     addDomain(conn, currentLine[7], currentDeviceId)
```

```
26          else:
27              continue
28
29  def addDomain(conn_, domainName_, currentDeviceId_):
30      DatabaseAction().insertDomainsByDeviceId(conn_, domainName_, False,
        currentDeviceId_)
31
32  if __name__ == '__main__':
33      main()
```

Listing 6.13: DNS sniffer python script

On line 6, the connection to the database is established. Thanks to that, we are able to add the rows when a new domain is found. The purpose of the line 8 is to sniff all the data going through the WLAN0 on the port 53. The following line is an example of the output obtained:

```
['13:49:26.328277', 'IP', '10.3.141.175.52697', '>', '1.1.1.1.53:',
    '50829+', 'A?', 'unagi-na.amazon.com.', '(37)']
```

Listing 6.14: Example of the tcpdump command output

The goal is to isolate the packets with “A?”. Indeed, these packets are the DNS queries used to request a new domain name. In this case, the requested domain was “unagi-na.amazon.com.”.

Thanks to the `while` loop at line 10, the script is going through all the packets in realtime. Line 13 checks if the packet has “A?” to ensure that it is a DNS request. Once this verification is done, the script checks if the device is new and if so, it is added to the database. Then, the domain is added and linked to the device. If the device is already known, the presence of this domain in the database is checked. If it is, the script performs the same verification with the next packet.

It is really important to sniff all the traffic data in order to add the contacted domains. In this case, all the domains added are “blacklisted”. An action from the user is needed in order to whitelist only the necessary ones and the ones which seem to be legitimate from their point of view.

6.3.2 DNS Filter

Once the sniffer is done, the next goal is being able to filter a part of the traffic data. This subsection will discuss the several concepts and technical choices made.

The first step was about filtering the data based on the DNS requests. As written in the section 5.1, the point is to deny everything but some domains. The whitelisting should be done by the end user in order to review each domain and allow only the needed ones. In order to analyse those packets, it is firstly required to redirect all the traffic data to a queue. Thanks to the following commands, the traffic is sent to the queue number zero.

```
1 MAIN_QUEUE_NUM = 0
2 os.system("iptables -A FORWARD -j NFQUEUE --queue-num " +  MAIN_QUEUE_NUM)
```

```
mainQueue = NetfilterQueue()
try:
    mainQueue.bind(MAIN_QUEUE_NUM, process_packet)
    mainQueue.run()

except KeyboardInterrupt:
    os.system("iptables --flush")
```

Listing 6.15: Iptables rules and binding on the queue

When the queue rules are set, the binding can be launched. The "except" block allows the user to flush all the rules and come back to a normal configuration.
The following snippet of code checks if the packet is a DNS Resource Record. If so, the DNS domain is extracted and compared with the database row in order to verify if it is a legitimate domain or not. If it is not, the packet is dropped.

```
def process_packet(packet):
    scapy_packet = IP(packet.get_payload())
    if scapy_packet.haslayer(DNSRR):
        qname = scapy_packet[DNSQR].qname.decode('utf-8')
        if not DatabaseAction().isLegitByDomainDevice(conn, qname,
    scapy_packet[IP].dst):
            packet.drop()
            return
```

Listing 6.16: Database request to filter the packets based on the DNS

Once the packets are filtered, the concept of parent-child (5.2) is used. The device is checked based on its IP address. If the device is present in the `linked_devices` table, the script checks, based on the query response if the current device is the parent. If so, the packet is accepted. Otherwise, that means the packet comes from a child and the parent did not "speak" and it is dropped.

```
    devices = DatabaseAction().searchLinkedDevicesByIp(conn, scapy_packet[
    IP].dst)
        if devices != -1:
            currentDevice = devices[1]
            childDevice = devices[0][0][1]
            parentDevice = devices[0][0][2]
            if currentDevice == parentDevice:
                child = DatabaseAction().searchDeviceByID(conn, childDevice
    )
                # Accept the child
                t = Thread(target = launch_thread, args = (child[2],))
                t.start()
                packet.accept()
                return
```

Listing 6.17: Parent-child principle applied to a python script

If the parent sends packets, a thread is launched in the `process_packet` function to let the latter running and does not pause the other transmissions. Within the function called by the `process_packet` function, some IPTables rules are created to redirect the child packet to another queue. Those rules are inserted before the one used to redirect all the traffic to the queue 0. Once these new rules have been applied, the traffic coming from the child is redirected and processed in such a way to still accept the whitelisted domains. The following rules enable the routing to the child queue. The traffic forwarded is only the one from a given IP address.

```
1 def create_rule(childDeviceIp_):
2     os.system("iptables -I FORWARD 1 -s " + childDeviceIp_ + " -j NFQUEUE
    --queue-num " + CHILD_QUEUE_NUM)
3     os.system("iptables -I FORWARD 1 -d " + childDeviceIp_ + " -j NFQUEUE
    --queue-num " + CHILD_QUEUE_NUM)
```

Listing 6.18: Creation of the rules to redirect the traffic to the child queue

As written in the section 5.2, the goal of this concept is to drop the DNS requests from a child when the parent is not sending DNS requests. Therefore, the time window in which the child device can send packets is limited. In order to calculate this time frame, several tests were performed (6.2.2.1). From these tests, several figures were created (Figure 6.6, Table 6.1, Figure 6.7). Based on those figures, it is worth noting the distribution of the data. Indeed, the script needs a time window in which the child is allowed to send packets out of the LAN. The threshold has been set at 10 seconds to have a system working in more than 90% of the cases as justified in section 6.2.2.1.

The binding of the packets is done in the same way for the two queues. In both cases, the DNS will be checked and if the domain is not whitelisted, the packet will be dropped.

If the user blacklisted all the domains contacted by the devices, even if the child is allowed to send DNS requests, the filter above will drop it. It is then important for the end user to whitelist and blacklist the domains carefully for the good work of the device.

```
1 def process_child_packet(packet):
2     scapy_packet = IP(packet.get_payload())
3     if scapy_packet.haslayer(DNSRR):
4         qname = scapy_packet[DNSQR].qname.decode('utf-8')
5         child_conn = DatabaseAction().create_connection("../UCLThesis/db.
    sqlite3")
6         if not DatabaseAction().isLegitByDomainDevice(child_conn, qname,
    scapy_packet[IP].dst):
7             child_conn.close()
8             logging.debug("DROP PACKET CHILD BLACK LISTED domain \"{a}\"".
    format(a=qname))
9             packet.drop()
10            return
11        else:
12            child_conn.close()
13            logging.debug("ACCEPTED PACKET CHILD \"{a}\"".format(a=qname))
```

```
14              packet.accept()
15              return
16      # accept the packet because not a DNS question one
17      packet.accept()
```

Listing 6.19: Function to process the child packets during a given time

When the 9 seconds come to the end, the rules are deleted thanks to the method below and the child's traffic goes back to the regular queue.

```
1 def delete_rule(childDeviceIp_):
2     os.system("iptables -D FORWARD -d " + childDeviceIp_ + " -j NFQUEUE --
    queue-num "+ CHILD_QUEUE_NUM +" 2> /dev/null")
3     os.system("iptables -D FORWARD -s " + childDeviceIp_ + " -j NFQUEUE --
    queue-num "+ CHILD_QUEUE_NUM +" 2> /dev/null")
```

Listing 6.20: Deletion of the rules to redirect the traffic to the child queue

After that, the child is not able to send packets any more until the parent sends some packets again.

6.3.3 Web interface

During the implementation of this proof of concept, we remark that it was important to visualise and easily manage the devices, the domains and the linked devices. In order to do so, a Django web interface has been created. The latter is divided into two parts; the **administration panel** (see Figure C.1) which requires a username and a password. On the other hand, a **regular view** (see Figure C.6) where non-administrator users can view the domains, devices and linked-devices. On the latter, none updates are allowed.

The administrator can manage the different part of the configuration related to the filter 6.3.2 such as

- Display, create, update, delete a **device** (see Figure C.2)
- Display, create, update, delete a **domain** and modify the linked device(see Figure C.3 and Figure 6.11)
- Display, create, update, delete a **link between two devices** (see Figure C.4 and Figure C.5)

The Figure 6.11 is an example of the administration interface.
Regarding the non-administrator user view, the design has been created in order to display the information in a simple way. On the top menu bar, the user can go to the administration view which asks for a username and password or obtain the device list. As shown on the Figure C.6, the ID of the device is clickable. If the user clicks on it, they are redirected to the view exposed on Figure 6.12 which is a list of domains filtered based on the device. The displayed domains are only the ones corresponding to the clicked device. Finally, the user can see a table with the relationships between the devices (see Figure C.7).

The Figure 6.12 is an example of the non-administration interface.

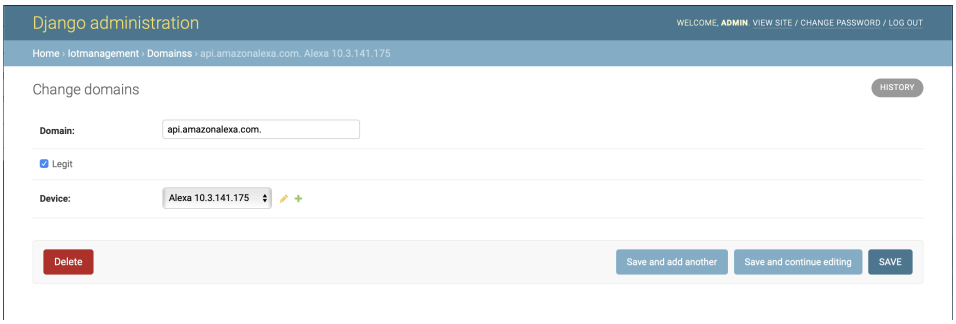

Figure 6.11: Django administration web interface for the handling of the mapping between a domain and a device

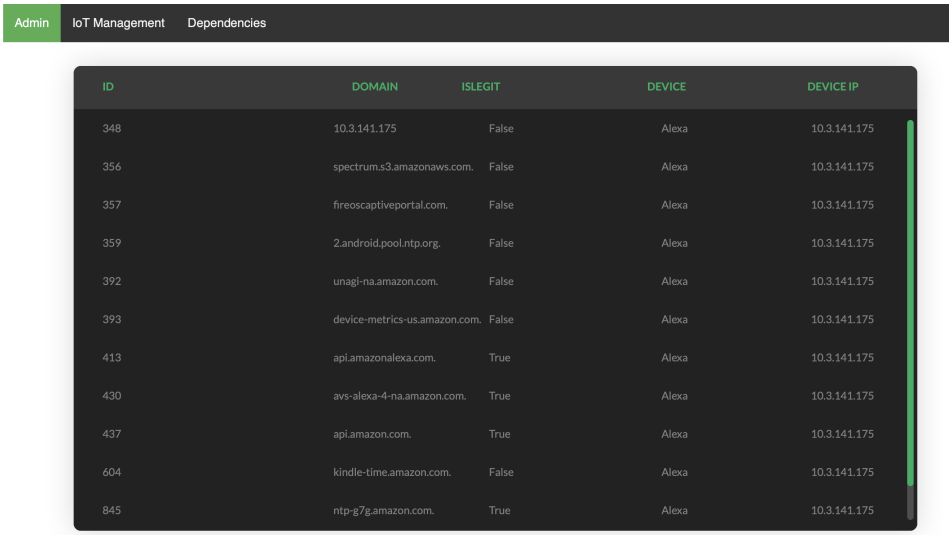

Figure 6.12: Domains list on the regular web interface

No modification is allowed on the regular user web interface. If any modification has to be done, the user needs to be connected to the administration panel in order to do it.

The python code used to implement the web interface is not displayed in this thesis. The whole code can be found on GitHub on request[8]: `https://github.com/alect096/memoire`

6.3.4 Results

This section will present the results of the implementation of the several concepts explained into the chapter 5. These concepts were implemented and explained within parts of the section 6.3.

In order to present some data, a logging method has been implemented. As written in the Appendix B, when the script is launched or interrupted or a packet is dropped, a log is added into a log file. The format of the logs is set at the beginning with the datetime, the user performing the action, the method where the log comes from and the message. Here is an example of the log result:

[8] You can find my GitHub credentials here: `https://github.com/alect096`

```
2020-05-08 01:30:46.792 DEBUG filter - process_packet: ACCEPTED PACKET
   PARENT "www.facebook.com."
```

Listing 6.21: Example of the log result from the filter script

To avoid the overloading of the file with useless information, several levels of logs have been set up. For example, the start and stop of the script have an `INFO` level. The dropped packet has a `DEBUG` level.

To prove that the packets are dropped, the logs were used to show which action was performed depending on the packets. For example, the following log shows a DNS request dropped because the parent was not "speaking" and the child tried to. The ID of the device is written in this case to know which one tried to send requests.

```
2020-05-08 01:31:21.856 DEBUG filter - process_packet: DROP CHILD (ID:11)
   PACKET WITHOUT PARENT TALKING
```

Listing 6.22: Example of the log result from the filter script

The following log extract shows an example with the parent-child principle. It is worth noting that at a given time, the parent did "speak" (`2020-05-08 01:31:39.114`) and then, the packets sent by the child are allowed to be sent.

```
2020-05-08 01:31:39.114 DEBUG filter - process_packet: ACCEPTED PACKET
   PARENT "asadventure.demdex.net."
2020-05-08 01:31:39.577 DEBUG filter - process_child_packet: ACCEPTED
   PACKET CHILD "api.amazonalexa.com."
2020-05-08 01:31:40.372 DEBUG filter - process_child_packet: ACCEPTED
   PACKET CHILD "unagi-na.amazon.com."
```

Listing 6.23: Example of the accepted child packet logged from the filter script with parent-child principle

At the beginning of the tests, the parent was not connected to the access point created by the Raspberry and then, the following behaviour could be noticed.

```
2020-05-08 01:24:56.227 DEBUG filter - process_packet: DROP CHILD (ID:11)
   PACKET WITHOUT PARENT TALKING
2020-05-08 01:25:01.593 DEBUG filter - process_packet: DROP CHILD (ID:11)
   PACKET WITHOUT PARENT TALKING
2020-05-08 01:25:04.594 DEBUG filter - process_packet: DROP CHILD (ID:11)
   PACKET WITHOUT PARENT TALKING
2020-05-08 01:25:06.233 DEBUG filter - process_packet: DROP CHILD (ID:11)
   PACKET WITHOUT PARENT TALKING
```

```
2020-05-08 01:25:09.598 DEBUG filter - process_packet: DROP CHILD (ID:11)
    PACKET WITHOUT PARENT TALKING
```

Listing 6.24: Example of the dropped packet logged from the filter script with parent-child principle

If the parent is not sending DNS requests, the child will not be able to send either and the DNS requests will be dropped.

6.3.5 Limitations

This part of the thesis will highlight the different limitations faced during the realisation of this proof of concept.

At the time of writing, the gateway is able to filter the traffic of the interface WLAN0. It means that the IoTs connected to the Internet via an Ethernet cable cannot be filtered. Because of a lack of material and the Raspberry has only one Ethernet interface it was not possible to plug multiple network cables and filters on the WLAN0 interface and on the eth0 interface.

The whole experiment has been tested on the WLAN0. That means that the principle of parent-child could not be tested on the Ikea Gateway as it required a connection via a network cable. In order to test the concept anyway, the parent was an iPhone and the child was Alexa. Thanks to that, it was possible to check that when the parent is sending packets, the child can too and when the parent does not send packets, the child is not allowed to do so.

During the development of the proof of concept, we noticed that when the filter was activated, the bandwidth was considerably impacted. Without it, we got around 10 Mb/s and when it was enabled, it dropped to 4-5Mb/s. However, the goal in this case was not to have a high data transfer but more to prove that the parent-child principle could be a concept to apply to reduce the data leak.

Finally, based on the implementation of the filter (see subsection 6.3.2) it appears that if the device already knows the IP address of the server, it is able to contact it anyway. Indeed, for now, it is only filtered based on the DNS request and not on the IP address. Moreover, it also appears that Alexa caches the queries for certain amount of time. If Alexa is set as a child, it will be able to send requests tanks to the cache. However, it will not be able to perform new DNS requests.

7. Future work

This chapter aims to provide insights in the possible future work of this master thesis. The several propositions below would improve the filter and the usage by the end users.

7.1 Manufacturer Usage Description

During the experiment, a sniffer (see subsection 6.3.1) was implemented to fill a database, whitelist some domains. After discussion with Professor Sadre and the reading of "Information Exposure From Consumer IoT Devices: A Multidimensional", it appeared that some improvements could be applied to the initialisation of a device. Indeed, at the time of writing, when a device is added to the network, the sniffer sniffs all the packets going through the WLAN0 and adds the contacted domains to the database as blacklisted domains. An action from the user is needed to whitelist it in order to let the device contact this domain. However, it is not really user-friendly to process like this.

A future work that could facilitate the user life would be to use the Manufacturer Usage Description (MUD) [29] to initialise the whitelisted domains for a device. Thanks to this initialisation, this would be easier for an end user to deploy the filter without configuring from scratch the required domains. Moreover, each device would have a profile on the acess point and based on it, it would accept or not a communication between two devices within the LAN [30].

7.2 Zigbee Gateway

At the time of writing, the gateway allows only the filtering on the WLAN0 interface. However, it could be useful to replace the Ikea Gateway by a Raspberry in order to directly interact with the lights without the intermediate gateway.

Thanks to a *ZigBee module*, it would be possible to interact with the lights directly from the Raspberry and then directly send the requests. This would allow the developer to know which packets it decides to send or not.

The goal is not to create the tool but integrate it to the actual filter. Indeed, this project[1] already exists. The added value would be the integration with the actual filter, in order to get rid of the manufacturer's gateway and bring together several technologies to propose a more advanced concept.

[1] `https://github.com/sandyjmacdonald/ikea-smartlight`. Visited on 09/05/2020

8. Conclusions

The goal of this master thesis was to provide a new concept in order to reduce the data leak.

To do so, a *state of the art* was, first of all, made. The latter is divided into three parts. The first one concerns the gateway. The second section of this chapter is dedicated to the traffic data. The last section discusses the data leak.

Based on this chapter, multiple problems could be highlighted such as the fingerprinting of the devices, analysis of the user data, side channel attack on the number of packets, ...

In order to respond to the problematic of the data leak, the concept of parent-child has been imagined. To implement this concept, a central position was needed to access to filter the traffic data. The concept of gateway presented in the state of the art has been used. The IoT's are connected to a filtering gateway where the parent-child principle is implemented. This concept filters the incoming and outgoing packets. If the parent is not sending DNS requests, the child is not able to. Moreover, if the domain of the DNS request is not whitelisted, the packet is dropped.

Thanks to this concept, the amount of packet sent by the child is reduced as the only periods it is allowed to do so is when the parent is sending requests too.

Once the concept working, it was necessary to figure out the time windows needed when the child is allowed to send DNS data. In order to determine this amount of time, several tests were performed and a specific time for the Ikea transmission was found. It is worth noting that each child should have a specific time window based on the time a request needs to be executed.

As explained in the chapter 7, *future work*, the next steps are the following. The first one could be the usage of MUD to initialise the whitelist of a device. This would simplify the first connection on the network for a new device. The second future work proposes to implement a Zigbee gateway to control some IoT's from the gateway and not use the constructors'. Thanks to that, this would improve the actual one and know which message needs to be sent. Using this type of device also allows managing devices without using cloud services.

Bibliography

[1] Oracle. What is the internet of things (iot)?, [Online]. Available: https://www.oracle.com/internet-of-things/what-is-iot.html#link3 (visited on 04/12/2020) (cit. on p. 5).

[2] I. J. Clark. (Nov. 17). What is the internet of things (iot)?, [Online]. Available: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/ (visited on 04/12/2020) (cit. on p. 5).

[3] RedHat. What is iot?, [Online]. Available: https://www.redhat.com/en/topics/internet-of-things/what-is-iot (visited on 04/12/2020) (cit. on p. 6).

[4] NXP. Zigbee mesh networking protocol | nxp, [Online]. Available: https://www.nxp.com/products/wireless/zigbee:ZIGBEE-PROTOCOL (visited on 04/13/2020) (cit. on p. 7).

[5] Y.-H. Chuang, N.-W. Lo, C.-Y. Yang, and S.-W. Tang, "A lightweight continuous authentication protocol for the internet of things", *Sensors*, vol. 18, p. 1104, Apr. 2018. DOI: 10.3390/s18041104 (cit. on p. 8).

[6] M. Morgenstern, *IKEA TRÅDFRI: A smart light in the darkness of IoT-Security*, en-US. [Online]. Available: https://www.iot-tests.org/2017/04/ikea-tradfri-a-smart-light-in-the-darkness-of-iot-security/ (visited on 04/24/2020) (cit. on p. 7).

[7] P. Schoutsen, *IKEA Trådfri: Internet of Things done right*, en. [Online]. Available: https://www.home-assistant.io/blog/2017/04/17/ikea-tradfri-internet-of-things-done-right/ (visited on 04/24/2020) (cit. on p. 7).

[8] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (iot) communication protocols: Review", in *2017 8th International Conference on Information Technology (ICIT)*, 2017, pp. 685–690. DOI: 10.1109/ICITECH.2017.8079928. (visited on 10/08/2019) (cit. on pp. 8, 9).

[9] P. Mockapetris, *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*, Nov. 1987. [Online]. Available: https://www.ietf.org/rfc/rfc1035.txt (visited on 06/07/2020) (cit. on p. 9).

[10] J. Jiang, J. Liang, J. Li, H. Duan, and J. Wu, "Ghost domain names: Revoked yet still resolvable", Apr. 2020 (cit. on p. 10).

[11] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home iot using openflow", in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 2016, pp. 147–156. DOI: 10.1109/ARES.2016.64. (visited on 10/09/2019) (cit. on pp. 11, 16, 17).

[12] P. Junges, J. François, and O. Festor, "Passive inference of user actions through iot gateway encrypted traffic analysis", in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 7–12. (visited on 10/14/2019) (cit. on pp. 11, 14, 16).

[13] J. Huang and H. Hsieh, "Design of Gateway for Monitoring System in IoT Networks", in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, IEEE, Aug. 2013, pp. 1876–1880. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.348. (visited on 10/15/2019) (cit. on p. 12).

[14] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders", *IEEE Pervasive Computing*, no. 3, pp. 12–22, 2018, ISSN: 1536-1268, 1558-2590. DOI: 10.1109/MPRV.2018.03367731 (cit. on pp. 12, 16).

[15] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed, "Intrusion detection systems for IoT-based smart environments: A survey", *Journal of Cloud Computing*, vol. 7, no. 1, p. 21, Dec. 2018, ISSN: 2192-113X. DOI: 10.1186/s13677-018-0123-6 (cit. on pp. 12, 16).

[16] W. Bul'ajoul, A. James, and M. Pannu, "Improving network intrusion detection system performance through quality of service configuration and parallel technology", *Journal of Computer and System Sciences*, vol. 81, Dec. 2014. DOI: 10.1016/j.jcss.2014.12.012 (cit. on p. 12).

[17] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools", *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, 2014. DOI: 10.1109/SURV.2013.052213.00046 (cit. on p. 12).

[18] J. Pacheco and S. Hariri, "IoT Security Framework for Smart Cyber Infrastructures", in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Sep. 2016, pp. 242–247. DOI: 10.1109/FAS-W.2016.58. (visited on 11/02/2019) (cit. on pp. 12, 13, 16).

[19] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying iot traffic in smart cities and campuses", in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 559–564. DOI: 10.1109/INFCOMW.2017.8116438. (visited on 10/14/2019) (cit. on pp. 14, 16).

[20] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A. Sadeghi, and A. S. Uluagac, "Peek-a-Boo: I see your smart home activities, even encrypted!", *ArXiv*, vol. abs/1808.02741, Aug. 2018. eprint: 1808.02741 (cit. on pp. 14, 16).

[21] H. Lee, J. Lee, J. Lee, and J. Lee, "Design of automatic identification gateway system for different iot devices and services", in *2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, G. Wang, Q. Han, M. Z. A. Bhuiyan, X. Ma, F. Loulergue, P. Li, M. Roveri, and L. Chen, Eds., IEEE, Oct. 2018, pp. 2022–2025. DOI: 10.1109/SmartWorld.2018.00338. (visited on 10/15/2019) (cit. on pp. 14, 16).

[22] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach", in *Proceedings of the Internet Measurement Conference*, ser. IMC '19, Amsterdam, Netherlands: Association for Computing Machinery, 2019, 267–279,

ISBN: 9781450369480. DOI: 10.1145/3355369.3355577. (visited on 11/02/2019) (cit. on pp. 14–16).

[23] S. R. Hussain, S. Nirjon, and E. Bertino, "Securing the Insecure Link of Internet-of-Things Using Next-Generation Smart Gateways", in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, IEEE, May 2019, pp. 66–73. DOI: 10.1109/DCOSS.2019.00032. (visited on 10/21/2019) (cit. on pp. 15, 16).

[24] T. van Leijenhorst, K.-W. Chin, and D. Lowe, "On the viability and performance of DNS tunneling", 2008 (cit. on p. 21).

[25] pfSense. (Aug. 5). Firewalling fundamentals, [Online]. Available: https://docs.netgate.com/pfsense/en/latest/book/firewall/firewall-fundamentals.html (visited on 07/03/2020) (cit. on p. 23).

[26] A. Dalvi, S. Maddala, and D. Suvarna, "Threat modelling of smart light bulb", in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1–4. DOI: 10.1109/ICCUBEA.2018.8697723. (visited on 07/03/2020) (cit. on p. 25).

[27] "Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans)", *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1–314, 2011, ISSN: null. DOI: 10.1109/IEEESTD.2011.6012487 (cit. on p. 25).

[28] W. S. Yoo, E. Kim, J. Lawrance, A. S. D. Marco, Y. Jagadeesan, and W. Hilmo, "Implicit SSL certificate management without server name indication (SNI)", en, pat. US8738902B2, Library Catalog: Google Patents, May 2014. [Online]. Available: https://patents.google.com/patent/US8738902/en (visited on 04/20/2020) (cit. on p. 32).

[29] Cisco. What is MUD? - Manufacturer Usage Description - Document - Cisco DevNet, [Online]. Available: https://developer.cisco.com/docs/mud/#!what-is-mud/what-is-mud (visited on 05/08/2020) (cit. on p. 43).

[30] *Practical Use of the MUD Specification to Support Access Control in the IoT*. [Online]. Available: https://www.youtube.com/watch?v=A-_HVsGjB0s (visited on 05/30/2020) (cit. on p. 43).

[31] M. Walker, *CEH Certified Ethical Hacker All-in-One Exam Guide, Second Edition*. McGraw-Hill Osborne Media, 2014, ISBN: 978-0-07-183648-7.

A. UCLouvain Poster Session

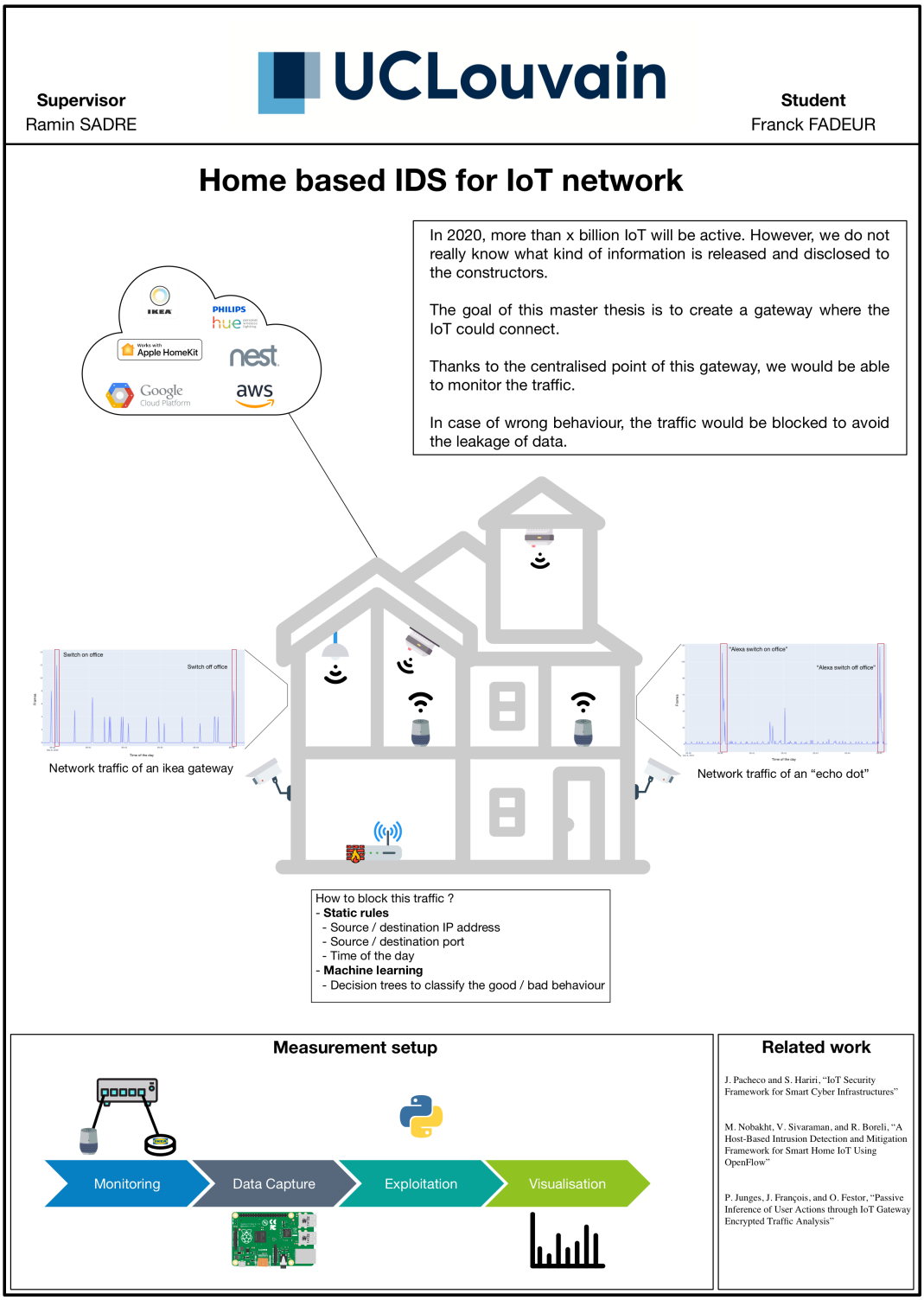

Figure A.1: Poster exposed at the poster session at the UCLouvain

B. Filter Python Script

This script is the full version of the current python filter script. This one filters the packets based on their DNS requests and with the parent-child principle (section 5.2).

```
import os
from db_actions import DatabaseAction
from threading import Thread
import time
import logging
from netfilterqueue import NetfilterQueue
from scapy.all import *

def process_packet(packet):
    # convert netfilter queue packet to scapy packet
    scapy_packet = IP(packet.get_payload())
    if scapy_packet.haslayer(DNSRR):
        # if the packet is a DNS Resource Record
        qname = scapy_packet[DNSQR].qname.decode('utf-8')
        if not DatabaseAction().isLegitByDomainDevice(conn, qname, scapy_packet[IP].dst):
            logging.debug("DROP PACKET BLACK LISTED domain \"{a}\"".format(a=qname))
            packet.drop()
            return

        devices = DatabaseAction().searchLinkedDevicesByIp(conn, scapy_packet[IP].dst)
        if devices != -1:
            currentDevice = devices[1]
            childDevice = devices[0][0][1]
            parentDevice = devices[0][0][2]
            if currentDevice == parentDevice:
                child = DatabaseAction().searchDeviceByID(conn, childDevice)
                # Accept the child
                t = Thread(target = launch_thread, args = (child[2],))
                t.start()
                logging.debug("ACCEPTED PACKET PARENT \"{a}\"".format(a=qname))
                packet.accept()
                return
            else:
                logging.debug("DROP CHILD (ID:{a}) PACKET WITHOUT PARENT TALKING".format(a=currentDevice))
                packet.drop()
                return
    # accept the packet because not a DNS question one
    packet.accept()


def launch_thread(childDeviceIp_):

    isPresent = check_rules(childDeviceIp_)

    if isPresent == 0:
         return

    create_rule(childDeviceIp_)
    t = Thread(target = binding)
    t.start()
    time.sleep(10)
    delete_rule(childDeviceIp_)
    t.join()
    return

def check_rules(childDeviceIp_):
    return os.system("iptables -C FORWARD -s " + childDeviceIp_ + " -j NFQUEUE --queue-num " + CHILD_QUEUE_NUM)

def create_rule(childDeviceIp_):
    os.system("iptables -I FORWARD 1 -s " + childDeviceIp_ + " -j NFQUEUE --queue-num " + CHILD_QUEUE_NUM)
    os.system("iptables -I FORWARD 1 -d " + childDeviceIp_ + " -j NFQUEUE --queue-num " + CHILD_QUEUE_NUM)

def delete_rule(childDeviceIp_):
    os.system("iptables -D FORWARD -d " + childDeviceIp_ + " -j NFQUEUE --queue-num "+ CHILD_QUEUE_NUM +" 2> /dev/null")
    os.system("iptables -D FORWARD -s " + childDeviceIp_ + " -j NFQUEUE --queue-num "+ CHILD_QUEUE_NUM +" 2> /dev/null")

def binding():
    childQueue.bind(CHILD_QUEUE_NUM, process_child_packet)
    childQueue.run()

def process_child_packet(packet):
    scapy_packet = IP(packet.get_payload())
    if scapy_packet.haslayer(DNSRR):
        qname = scapy_packet[DNSQR].qname.decode('utf-8')
        child_conn = DatabaseAction().create_connection("../UCLThesis/db.sqlite3")
        if not DatabaseAction().isLegitByDomainDevice(child_conn, qname, scapy_packet[IP].dst):
            child_conn.close()
            logging.debug("DROP PACKET CHILD BLACK LISTED domain \"{a}\"".format(a=qname))
            packet.drop()
            return
```

```
            else:
                child_conn.close()
                logging.debug("ACCEPTED PACKET CHILD \"{a}\"".format(a=qname))
                packet.accept()
                return
        # accept the packet because not a DNS question one
        packet.accept()

logging.basicConfig(level=logging.DEBUG,
    filename='filter.log',
    format='%(asctime)s.%(msecs)03d %(levelname)s %(module)s - %(funcName)s: %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S')

logging.info("Filter has start running")
MAIN_QUEUE_NUM = 0
CHILD_QUEUE_NUM = 1

# insert the iptables FORWARD rule
os.system("iptables -A FORWARD -j NFQUEUE --queue-num " + MAIN_QUEUE_NUM)

# instantiate the netfilter queue
mainQueue = NetfilterQueue()
childQueue = NetfilterQueue()

# DB connection
conn = DatabaseAction().create_connection("../UCLThesis/db.sqlite3")

try:
    # bind the queue number to our callback `process_packet` and start it
    mainQueue.bind(MAIN_QUEUE_NUM, process_packet)
    mainQueue.run()

except KeyboardInterrupt:
    # remove rules inserted, back to normal.
    logging.info("The filter has been interrupted")
    os.system("iptables --flush")

```

Listing B.1: DNS filter script

C. Web Interface

C.1 Administration Web Interface

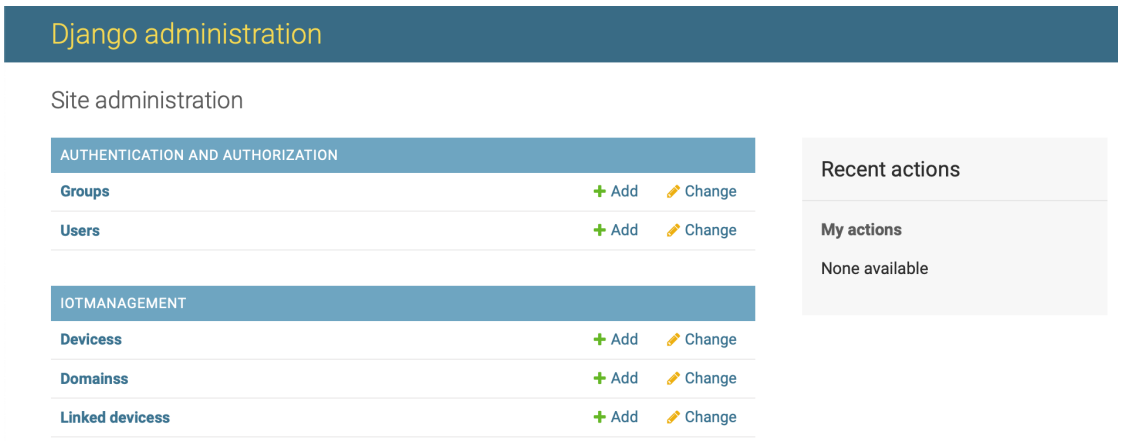

Figure C.1: Django administration web interface for the handling of the filter

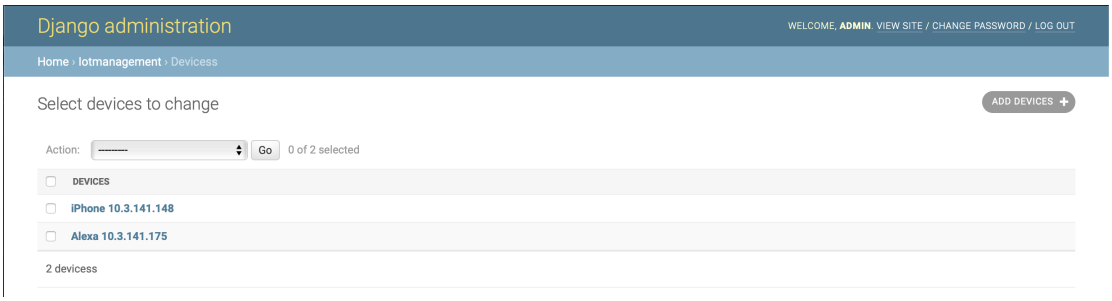

Figure C.2: Django administration web interface for the handling of the devices

Figure C.3: Django administration web interface displaying all the domains

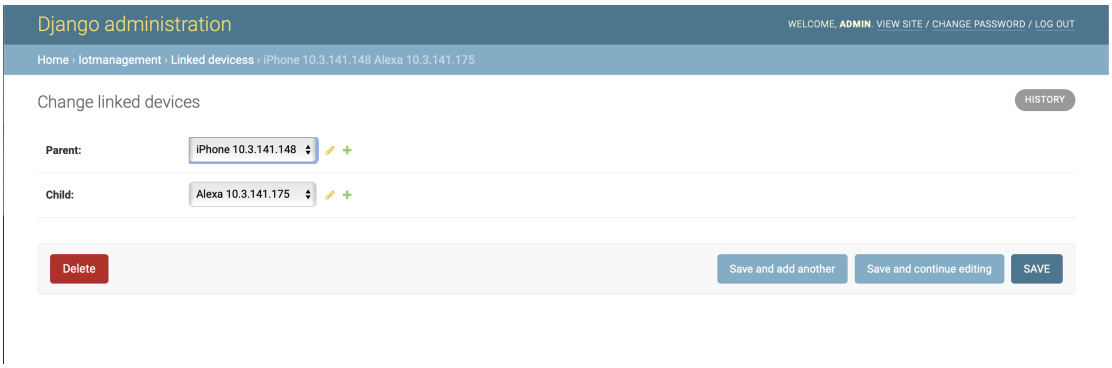

Figure C.4: Django administration web interface for the handling of the link between two devices

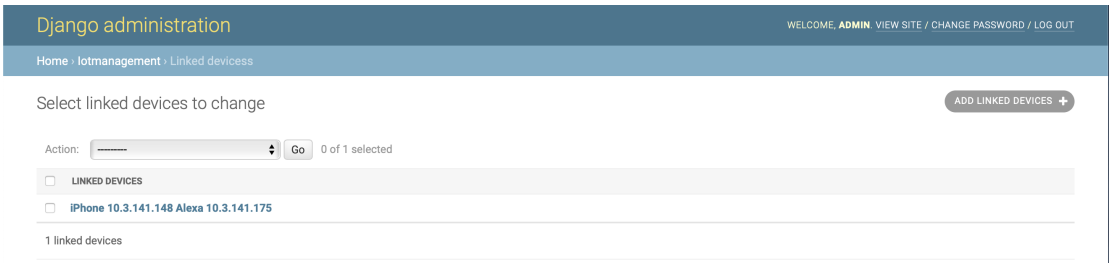

Figure C.5: Django administration web interface for the displaying all devices linked

C.2 Regular User Web Interface

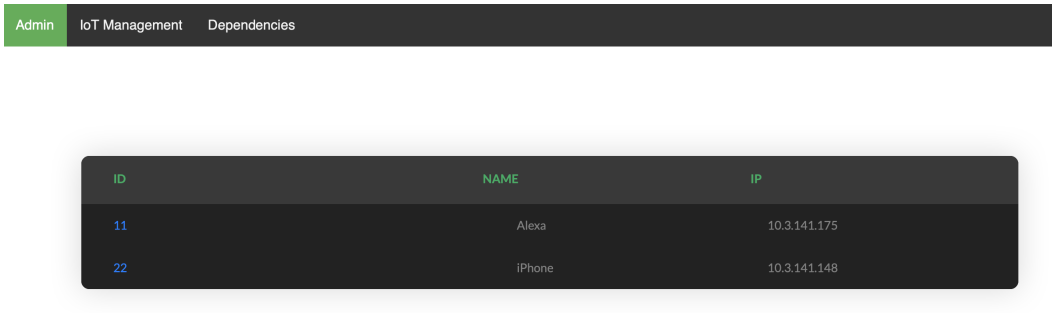

Figure C.6: Device list on the regular web interface

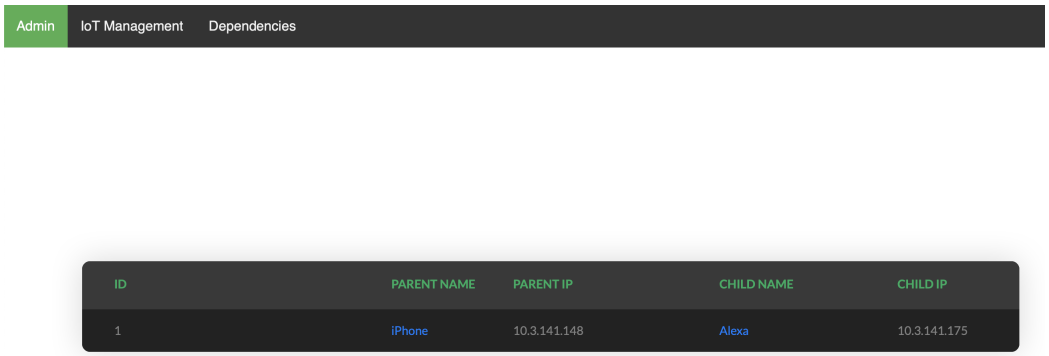

Figure C.7: Linked devices list on the regular web interface

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**