CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Investigating Privacy Protection in the Smart Home

Master's thesis in Computer science and engineering

Sara Boström Jonatan Nylund

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

MASTER'S THESIS 2020

Sara Boström Jonatan Nylund

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Investigating Privacy Protection in the Smart Home
Sara Boström Jonatan Nylund

Supervisor: Magnus Almgren, Department of Computer Science and Engineering
Advisor: Erik Dahlgren, Scionova
Examiner: Tomas Olovsson, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A word cloud generated by our most common phrases in the thesis.

Typeset in LaTeX
Gothenburg, Sweden 2020

Investigating Privacy Protection in the Smart Home
Sara Boström
Jonatan Nylund
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Consumer-oriented IoT devices, or smart home devices as they are also called, are getting more common. Projections suggest there will be more than 75 billion of these devices in people's homes by 2025. Given the location of these devices and the nature of the data they collect, this raises questions about how the user's privacy can be protected.

Given this background, this thesis investigates relevant security concepts and the feasibility of some of their software implementations as privacy tools. Moreover, smart home devices are also compared in regards to privacy based on brand recognition, which market they are aimed for and which device category they belong to. This is done in order to be able to draw conclusions about these properties' impact on privacy.

The comparison is made using a testbed where traffic from the smart home devices is analysed in relation to the properties above as well as to a threat model developed based on privacy threats found in the literature. The software Princeton IoT Inspector and Snort in combination with the ELK stack are used and compared in regards to both how well each manage to identify and highlight privacy threats but also their applicability to different user groups. Furthermore, we also design a proof of concept for the viability of a cloud solution. For this we simulate a third party developing rules, based on user-generated Snort logs, which a user can subscribe to.

The results show that the properties mentioned above have a significant impact on how the devices behave. That is, they affect which endpoints the devices connect to, which cloud provider they rely on and also the shape of their traffic to a large extent. Furthermore, the results also show that a cloud solution is possible, although the size of the logs quickly becomes an issue. Thus further study on how to optimize the logs is needed while avoiding proprietary solutions.

None of the investigated software solutions succeeds in striking a perfect balance between usefulness and user-friendliness. Future work needs to be done on multiple levels, ranging from how to increase user awareness, involve community and third party initiatives as well as to investigate what role legislation might play. This will not be an easy undertaking, although a necessary one in order to protect the privacy in our own homes.

Keywords: Smart Home, IoT, Privacy, Data collection, Packet sniffing, IPS/IDS, Cloud developed rules, Community, Third party, Legislation.

Acknowledgements

First, we would like to thank Magnus Almgren, our supervisor from Chalmers, for being a great help in developing our methodology and guiding us through this process. It has been a bit of a bumpy road, but thanks to him we have managed to stay on track despite the pandemic. We are also grateful for Tomas Olovsson for taking the time to be our examiner and providing insightful feedback.

Additionally, a big thanks to Scionova for giving us the opportunity to write our master thesis with their company. A special thanks goes out to Erik Dahlgren for all the feedback throughout the process and for lending us some of his own devices for the experiments. We would also like to thank Henrik Sjöström for taking his time to help us with the administrative business, provide conversations about the future and for making us feel welcome at Scionova.

Finally, to our friends and family: you have been a source of constant encouragement and inspiration. This could not have happened without your support. Thank you, for everything.

Sara Boström, Jonatan Nylund, Gothenburg, November 2020

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The Internet of Things (IoT) is on the rise and quickly so. The number of devices has seen a significant increase in the last few years, and just by the year 2019 alone, more than 800 million additional devises were estimated to enter the market [1]. Moreover, by the year 2022, more than half of the Internet traffic is expected to be generated from IoT devices and approximately half of this volume is expected to come from consumer oriented IoT devices [1].

The diversity of consumer IoT devices, ranging from smart bulbs to smart speakers, offer a wide range of functionality to the user and help to increase everyday convenience. A house that contains a communication network that connects different IoT devices and allows them to be remotely controlled, monitored and accessed, is often called a smart home [2]. This has resulted in IoT devices designed to be used in the home to be called *smart home devices* to distinguish them from other kinds of IoT devices (such as devices associated with Industry 4.0, smart hospitals, driverless cars, etc.). Given that smart home devices typically are connected to the Internet and designed to be integrated in our daily lives, this raises questions about convenience versus privacy.

1.1 The Smart Home and User Privacy

There is not much ambiguity in how important user data is for businesses today. Today the world's largest companies in terms of market capitalization primarily work with information technology, a development that has been intensified the last decade [3]. These companies, often referred to as *The Big Five*; Apple, Alphabet, Microsoft, Facebook and Amazon, heavily depend on user data [4]. In contrast to these companies wanting user data, the everyday user is worried about their data collected by third parties without their knowledge. According to a study by Pew Research Center, a majority of Americans were concerned about the way their data is being used by companies (79%) or the government (64%). Most also felt they have little or no control over how these entities use their personal information [5].

Due to the general placement of smart home devices, i.e. in our homes, these concerns might be especially valid for data generated by smart home devices. For instance, some data might be particularly sensitive, such as data generated from medical smart home devices. It is also possible that the user is never informed about what data is being collected and shared, and when these policies change. An example of the latter would be the incident back in 2016 when it was revealed that four manufacturers of fitness wristbands violated rules regarding the collecting and selling of data [6]. Such incidents might further increase the general public's unease regarding data collection.

This imbalance of the interest of companies in user data and the users' lack of awareness could be considered problematic. Smart home data collection raises questions: What information is being shared? To what extent is the information being shared and with whom is it being shared? Furthermore, is there a difference between different manufacturers or product categories (where some categories could arguably be considered to be more intrusive)? Does privacy-related legislation affect the way smart home devices collect data? These question might be even more relevant as inexpensive smart home devices, often offered by less known companies, are becoming increasingly popular.

Besides user data collection, there are other ways user privacy could be compromised in the smart home. For instance, many of the devices are constantly connected to the Internet in order to be able to communicate with their back-end services. As such, they are more susceptible to draw attention from people with malicious intent. Furthermore, the lack of encryption by certain devices might unintentionally expose information to eavesdroppers. This further highlights the complexity of privacy and that some aspects of security and privacy are to a high degree intertwined.

There is research conducted on IoT devices that suggests that privacy concerns are justified, such as the research from Moghaddam et al. [7] and Ren et al. [8]. It highlights the need for increased user awareness regarding what information he or she is (voluntarily) sharing. Nevertheless, many solutions are arguably inaccessible for a majority of users due to lack of either hardware, software or knowledge requirements. For that reason, legislation plays a key role for making sure that the general public's privacy is being protected.

1.2 Privacy and the General Data Protection Regulation

Since user data plays an increasingly important role for businesses today, different legislative measures have been implemented in the EU in the last few decades to increase the protection of its citizens' privacy. One such legislation is the ePrivacy Directive which requires companies to get consent from users for non-essential tracking cookies [9]. Another, more recent legislation, is the General Data Protection Regulation (GDPR). It is by some argued to be the most challenging security and privacy law as of today [10]. GDPR governs how companies are allowed to process and store personal data on EU individuals [11]. Personal data consists of many different types of information such as name or email address but also location data that are tied to an individual [12]. It covers all EU citizens if they reside in the EU no matter where the site the user might visit resides. This forces large businesses to respect the law, although they might have business in other parts of the world [10]. Breaking this law can have severe consequences since the fines can be significant. For instance, Google has been fined for 50M € on lack of consent to advertisement [13].

1.3 Aims

Given the context described in previous sections, this thesis has the following four aims:

I. Find a consensus of privacy threats against consumer-oriented IoT devices, or a unification of threats identified in the research literature if a consensus can not be found.

II. Investigate if there are differences in regards to privacy between consumer IoT devices from well known compared to less known brands aimed for the EU market, as well as between those devices and devices aimed for markets not covered by the GDPR.

III. Explore different commercially available programs that are used to detect suspicious network traffic to see how efficient they are at detecting privacy threats to the smart home. They will also be considered from the perspective of ease of use for the average consumer.

IV. Investigate how third party services might improve the privacy protection for the smart home in the future.

1.4 Scope and Limitations

Previous research has shown that privacy is a real concern regarding many smart home devices [7][1][8]. This thesis' main contribution to the scientific community originates from doing privacy comparisons between different products aimed for the GDPR market as well as between those products aimed for markets not covered by the GDPR legislation. The main reason for doing this is twofold. Firstly, GDPR is a significant legislation affecting both companies and citizens within the EU. Focusing on its potential effect on smart home devices behaviour in regards to privacy can arguably be considered important. Secondly, regional comparisons have not been that prominent in research we have been exposed to, which further makes this thesis' comparisons even more important.

The second area of contribution is investigating ways a user can apply different protection mechanisms as used in the different experiments throughout this thesis. Moreover, and based on this, the thesis also discusses the different level of protection and what role users, community, service providers and legislation might play in the future. Although similar discussions have been made in other research, this thesis gives a more hands-on approach to how third parties might be incorporated to help increase user privacy protection.

Nevertheless, in order to make the scope of the thesis manageable, several limitations have been made. First and foremost, we had to consider how many vendors/brands, regions, devices and device categories to choose in order to create a representative set. The choices in regards to these aspects were carefully considered.

We also limited the scope by not focusing on different privacy legislation in the regions being compared. Instead, we assumed that the GDPR ought to have an impact on device behaviour and thus work as a good baseline. If no apparent behavioural difference is detected between the regions, that can be the result of good privacy legislation in other regions unknown to us, although unlikely.

Finally, there is always a trade-off in regards to time and effort between theory and experiments when there are time constraints to consider. In this thesis, we believe a reasonable balance has been struck between conducting experiments and obtaining results and relating these to the current knowledge within the field by dividing the time between the two somewhat equally.

1.5 Thesis Structure

The thesis has the following structure:

Chapter 1 introduces the research topic and puts it in context. Furthermore, in this chapter, the aims of the thesis are declared along with a description of its scope and limitations.

Chapter 2 presents previous research with respect to privacy in smart homes in order to highlight the findings of current research. A motivation is also given to why particular research papers were chosen for the literature study. Moreover, this chapter also states what this thesis' contribution is in relation to previous research and sets the target threat model.

Chapter 3 describes what constitutes a smart home as well as introduces some of the protocols it relies on and that are of particular interest in terms of privacy and IoT. These are also essential for understanding the different experiments conducted in later chapters. It also presents some of the concepts that are fundamental to smart homes today, such as the role of back-end services.

Chapter 4 is dedicated to concepts that can be used for privacy protection. Some of the concepts introduced in this chapter are also essential for understanding the different software solutions used for the experiments in the thesis.

Chapter 5 features the design of the experiments. It begins with a description of the purpose of the experiments and how they map to the different aims. This is followed by a discussion on the segmentation (why certain devices belong to a certain test group) and on why some devices were chosen instead of others. The chapter concludes with a description of the software used for the different experiments, which are based on some of the concepts previously presented in Chapter 4.

Chapter 6 describes the implementation of the experiments. It explains the testbed and the execution of the three experiments in greater detail. It discusses how the traffic and the logs are captured, stored, sent, and analysed throughout the different experiments.

Chapter 7 presents the results from the experiments. It begins with showing the results from the smart home devices in relation to the threats in the threat model presented in Chapter 2 and concludes with a section dedicated to the main observations from the cloud solution in the final experiment.

Chapter 8 begins with discussing the interpretation of the results from Chapter 7. This is followed by a discussion on technical limitations throughout the experiments as well as ethical issues and sustainability. The chapter concludes with a dedicated section to future work.

Chapter 9 summarises the main findings and general conclusions of the thesis.

CHAPTER 2

Related Work

This chapter covers some of the current research in regards to privacy and smart homes in order to give context to the thesis, and by doing so, highlight its contribution.

It begins with a section dedicated to the results from previous studies conducted on consumer IoT devices and is concluded with a subsection discussing why it is difficult to detect privacy threats to consumer IoT devices in the first place.

The final section in the chapter highlights this thesis' contribution by discussing the aims of the thesis. Furthermore, the threat model used throughout the thesis is also defined in this section. The threat model is significant as it serves as the model to which the different software solutions, as well as device behaviour, are evaluated.

2.1 Data Collection from the Smart Home

This section presents some of the major findings from four recent and quite diverse studies conducted on smart home devices.

2.1.1 Paper Collection Methodology

Section 2.1 is, to a large extent, based on a number of research papers. The reasons as to why these particular papers were chosen are several. First and foremost, all papers are current (as in being published at the earliest in 2019). Secondly, they differ from one another in some key aspects. For instance, two of the papers conduct their studies on testbeds, whereas the other two are done on real-world traffic generated by regular users participating in the studies. This arguably gives more credibility to the idea of finding general patterns of privacy threats to the smart home since they analyse different environments.

Furthermore, since most traffic today is encrypted, analysing actual payload requires a significant undertaking and understanding of the platform one is analysing. Thus, papers that deal with analysis of payload tend to cover a smaller range of devices. For that reason, some of the papers presented below will cover fewer product categories although in more depth, whereas others span a wider range of products. As such, we considered it important to include both kinds of papers.

Finally, since one of the focus areas of this thesis is to investigate how legislation impacts technology by comparing products from different regions, we also include a study that covers regional comparisons between the UK and the US.

2.1.2 Research on Streaming Devices and Payload

Recent studies show that many smart home devices expose or share information with third-party sources and advertisers [7][1][8]. For instance, when analysing Internet traffic from over-the-top (OTT) streaming devices, Amazon Fire Stick and Roku TV specifically, Moghaddam et al. found that a majority of the apps/channels on both platforms contacted third parties [7]. Furthermore, they also found cases of persistent identifiers being collected. As they point out, the collecting of such identifiers, like MAC addresses, is problematic since they can not be reset by the user to avoid further tracking [7]. Moreover, some channels even shared personal data like email addresses or zip codes with third parties [7].

OTT streaming devices have relatively complex operating systems compared to devices such as smart plugs or thermometers and offer the user a variety of channels to choose from. Moghaddam et al. chose to divide the channels into different categories to investigate if there are any differences between the categories in regards to privacy [7]. As it turned out, the channels contacting the most tracking domains belonged to the news category, where some channels contacted almost as many as 60 tracker domains [7]. They also looked into what content was being shared, and it turned out that titles were leaked to third parties, particularly so amongst news channels [7]. Recalling the Cambridge Analytica scandal in 2018 [14], this could be considered problematic given the nature of the content. Moreover, since some companies such as Google and Amazon et al. are present on many platforms, i.e. not isolated to the world of IoT, this consolidation of data by a few actors might further enhance similar future risks. As Kumar et al. also highlights in their study, despite there being approximately 14 000 different vendors within IoT, about 90% of the devices worldwide are produced by merely 100 companies [15].

Another disconcerting discovery from the study conducted by Moghaddam et al. is the somewhat widespread use of unencrypted communication, where a significant majority of channels on both platforms use unencrypted communication for at least one request [7]. Other studies have shown similar concerns regarding the use of HTTP and the lack of encryption for IoT devices [1][8].

2.1.3 Research on Real-World Traffic

In contrast to the study by Moghaddam et al., Mazhar et al. conducted a study in 2018 on real-world traffic from American households and collected data on more than 1000 different devices [1]. Of these devices, 66 were classified as unique IoT devices, i.e. many users had the same type of device and a significant portion of the devices consisted of laptops, smartphones, etc., which is not considered to be IoT devices.

In the study by Mazhar et al. no payload traffic was analysed [1], which means that they could not investigate the content of the messages between the device and the back-end servers. Nevertheless, what they did find out regarding the shape of the traffic was that there was a clear difference between devices that relied on user input and those that did not [1]. The former primarily generated traffic when the user was at home/using the devices whereas the latter (smart cameras, sensors, etc.), instead displayed a more regular pattern [1]. Moreover, as Mazhar et al. highlight, given the size of the packet flows, one could also conclude what kind of data a certain flow contains. For instance, more than 25 % of the packet flows from smart cameras were larger than one megabyte [1]. This indicates that a cunning person could draw conclusions on what kind of interactions certain flows are associated with based on their time and size patterns. Similar concerns are raised by Ren et al. based on the results from their study [8].

Another large IoT security study by Kumar et al., done on 83 million devices from 16 million households across the world in collaboration with Avast, an anti-virus company, showed that there are large variations in terms of what types of smart home devices are being used in different parts of the world [15]. For instance, they show that Internet-connected TV:s or streaming devices are common in the US, whereas they constitute just a small percentage in regions such as South Asia. Instead, surveillance cameras are much more common in this region [15]. Kumar et al. also found that devices from the same manufacturer, aimed for different markets, sometimes had different security postures in terms of default passwords [15]. This highlights the heterogeneity of smart homes and why it is a complex domain to study.

However, as Mazhar et al. highlight, although the frontend for smart home devices is heterogeneous, the back-end is highly centralised [1]. They claim that many of the back-end services are hosted on just a few cloud platforms. For instance, for smart TVs, smart speakers, smart assistants, and home automation devices, Google Cloud and Amazon Web Services (AWS) account for somewhere between 60-90% of the traffic from those device categories [1]. Worth noting though is that both Google and Amazon are major actors in some of these product segments, which most likely impact those numbers.

Furthermore, Mazhar et al. found that six out of the eight product categories in their study accessed hostnames associated with either ad servers or trackers. This shows that tracking or advertisement delivery is occurring for a variety of smart home equipment, at least to some extent [1]. The study also highlights that most device categories, particularly prominent among smart assistants, had devices that accessed public DNS servers [1]. There are arguments for using public DNS-servers however. For instance, as Mazhar et al. mention, it avoids having to rely on the ISP supplying a correctly configured DNS, as well as helps to protect geolocation sensitive data [1]. Nevertheless, they point out that this limits the user's ability to block traffic on a network level [1], making software solutions such as Pi-hole [16] pointless.

As in the case of the OTT devices, the study of Mazhar et al. showed that all device categories in their study, except for smart cameras, used the HTTP protocol to at least some extent [1]. For some device categories, HTTP was used instead of HTTPS for almost as much as 20% of the traffic [1]. They argue that one of the reasons for why this is still being used for smart home devices, despite its obvious disadvantage to HTTPS, could be related to compatibility with third parties [1]. Moreover, in their study, advertisers and trackers were proportionally highly represented in their statistics of the HTTP traffic [1]. However, in a study conducted by Ren et al. where the actual payload was studied, minimal personal identifiable information (PII) was sent in plaintext [8], which is at least somewhat reassuring.

2.1.4 Research on Regional Differences

Ren et al. studied smart home devices from two different regions, one test environment located in the UK and another in the US [8]. Similar to Moghaddam et al., they also found that devices such as Smart TVs and OTT streaming devices contacted many third party sources [8]. This category of devices contacted the most third party sources of all the device categories in their study [8].

Furthermore, they also found that a majority of the traffic from both the UK and the US devices in their testbeds terminated in the US [8]. Moreover, most of the traffic that did not terminate in either the EU, the UK or the US, terminated in China. They argue that this is related to the fact that many Chinese devices host their services on the Alibaba Cloud [8]. Ren et al. also highlight the fact that AWS and Google are contacted by a large number of devices in their study and that significant outsourcing to cloud providers for computations took, which they claim partly explains the high proportion of traffic to non-first parties [8].

Furthermore, regarding regional differences and endpoints, they report that US devices tend to contact third parties to a higher degree compared to the UK devices. They argue that this discrepancy might be related to differences in legislation [8]. Interestingly enough, they did not observe any major change in this regard when using VPN [8] which might indicate that the difference is in the device itself (possibly in firmware).

However, they did find quite big differences between different products segments regarding encryption, where cameras were the worst performers and audio devices the best [8], a striking difference compared to the results in the study of Mazhar et al. where cameras performed the best in this regard. Ren et al. argue that this might be explained by the fact that the audio devices in their testbeds were produced by large companies such as Google and Amazon which are associated with solid security standards [8]. They also found difference in the unencrypted traffic between devices, although they argue that this discrepancy had more to do with the actual device rather than which category it belonged to [8]. Nevertheless, despite that unencrypted traffic only constituted a minority of the observed traffic, it was still present across all devices and categories in both of their testbeds [8].

Regarding strange device behaviour, they found evidence of a camera being triggered and recording video although it was not triggered by human actions. As they point out, it could be due to background noise, but could also be that it intentionally captures video without being triggered [8]. This highlights the difficulty of defining and then detecting unexpected behaviour amongst consumer IoT devices.

2.1.5 The Difficulty of Detecting Privacy Threats

One of the main difficulties in trying to detect and defend against privacy threats has to do with the complexity of defining/labelling "misbehaviour". How to we distinguish acceptable behaviour from an IoT device? One attempt to tackle this is the Manufacturer Usage Description (MUD), an IETF protocol that aims to define how a non-general purpose devices is expected to work. This, as IETF express, would amongst other benefits, help reduce the number of attack surfaces on the device [17].

Furthermore, Ren et al. discuss the role of machine learning and statistical techniques in order to be able to draw conclusions on what is normal device behaviour from what is not [8]. However, machine learning algorithms tend to be CPU intensive, and as the number of IoT devices are expected to grow in the coming years, this might not be a feasible solution for a typical smart home user, at least not regarding running it locally.

In recent years, a software called Princeton IoT Inspector has been developed by Huang et al. [18]. It is a tool that helps to visualise the traffic of smart home devices and does so by requiring little to no technical knowledge from the user. Furthermore, by using this software, the user contributes to the scientific community by allowing the developers of the software (i.e. the researchers) to gather data on the devices which in turn helps them to better map the behaviour of smart home devices [19]. More on this software, which also is an essential tool used in the experiments in later chapters, can be read in Section 5.4.1.

2.2 Our Contribution

Given the threats presented in Section 2.1, many of them reoccur in a majority of the studies. However, it might be an exaggeration to talk about a consensus of privacy threats due to the diversity of smart home devices and environments. Instead, one could argue that it would be more suitable to discuss privacy threats in terms of finding a unification rather than a consensus. Although by no means a complete list of privacy threats to the smart home, the elements in Table 2.1 summarises some of the most prominent privacy threats presented in the previous section.

Privacy Threats to IoT Devices Found in the Literature
• Lack of encryption
• Third party tracking
• Collection of PII
• Unexpected device behaviour
• Shape of traffic (might reveal information)
• Enforced public DNS usage (primarily smart speakers)
• Concentration of cloud providers (outsourcing computational capabilities)
• Concentration of market shares by a small fraction of companies
• Concentration of geographical endpoints (politically troublesome)
• Weaker privacy laws in certain parts of the world

Table 2.1: Some of the identified privacy threats to consumer IoT devices found in the literature.

Although this list does not say anything about the extent nor severity of the different threats, it arguably answers the first aim of the thesis - to find a unification of privacy threats to the smart home. However, the first aim does not necessarily contribute with anything new to the scientific community. Instead, the remaining aims serve this purpose (but are grounded in the list above).

The second aim of the thesis is twofold. Firstly, we investigate if there are any differences between devices of well-known brands compared to less known brands aimed for the market affected by the GDPR, i.e. the EU. Secondly, we compare all of these to devices aimed for other markets, outside the EU. Based on the literature we have read, not as much research has been conducted in this regard. For instance, with the exception of Ren et al. [8] who studied differences between devices from the UK and the US, Kumar et al. [15] focused on the security posture of IoT devices (although with a focus on regional differences). Both Moghaddam et al. [7] and Mazhar et al. [1] studied American household and/or specific devices. For that reason, we believe this comparison is of importance.

The third aim deals with exploring different commercially available software that can help the user to visualise and protect the smart home from privacy threats. This will give valuable hands-on insights into how well some of the more modern solutions such as Princeton IoT Inspector [20], and Snort [21] perform on IoT devices in regards to privacy. However, since no baseline exists, their performance will have to be evaluated in relation to each other rather than any kind of baseline. Nevertheless, this will raise relevant questions regarding usability, expandability and effectiveness.

Furthermore, the fourth and last aim is about investigating what role third parties might play in order to improve privacy solutions such as those used in the previous experiments. For instance, we will investigate the feasibility of developing rules on the cloud and subscribing to these. This in itself is not a new concept nor a new research question. However, given the setting it is applied in, we believe this proof-of-concept will highlight certain important aspect of using third parties for privacy protection. For instance, it will raise questions such as how best to send the data (with or without payload), at what frequency and at what privacy cost. This will lead to a general discussion about the viability of delegating responsibility of privacy protection to different parties, ranging from users all the way to legislation.

Finally, these devices and software solutions somehow needs to be compared and benchmarked against each other. In order to do so a threat model has to be defined.

2.2.1 Threat Model

The threat model can be seen in Table 2.2 and is based on the list of privacy threats identified from previous research.

Threat Model
• Lack of encryption
• Third party tracking
• Unexpected device behaviour
• Shape of traffic (might reveal information)
• Concentration of cloud providers (outsourcing computational capabilities)
• Concentration of geographical endpoints (politically troublesome)

Table 2.2: Our threat model.

From the threat model one can see that the analysis primarily will focus on metadata (data about data) rather than threats related to payload. The goal is to see if any patterns emerge and if there are any differences between the devices in regards to these threats. Moreover, this model also serves as the basis for the evaluation of the performance of the different software solutions used throughout the experiments, which are presented in later chapters.

CHAPTER 3

The Smart Home

This chapter introduces the reader to what characterises a smart home, ways users can interface with their smart home devices and some of the diversity of protocols that are part of the smart home universe. Moreover, certain protocols within the TCP/IP model are presented in more detail as they are either more problematic in regards to privacy and IoT and/or crucial to the experiments presented in later chapters.

The chapter concludes with a section dedicated to a few of the concepts and technologies behind the trend towards cloud solutions/cloud computing which also impacts the field of consumer IoT.

3.1 Characteristics of the Smart Home

This is the definition of the smart home according to the UK Department of Trade and Industry:

“A dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed.” [2]

From this we can surmise that there are three main things a home needs to make it smart. They need the following:

1. An internal and external communication network – this can be wired or wireless. Simply put, the devices in the home need to be connected to each other, but also to the Internet for external control.

2. The possibility of remote control – some sort of gateway to manage the systems. There has to be an intelligent way to control the system. Examples could be apps on a smartphone or a central gateway.

3. Home automation – products within the homes and links to services and systems outside the home.

There are three main types of technologies present in a smart home - sensors, a control centre and actuators [22]. Sensors provide information about the real-world environment and sends the information to the control centre. Secondly, the control centre makes a decision, based on user preference, and sends a command to an actuator that then act upon this information. As an example, a temperature reader measures the temperature at a set interval and sends this information to the command centre. If the current temperature is too high compared to the preferred temperature (as defined by the user), the control centre starts the air conditioner to make it cooler.

Smart home devices often have restrictions in their resources, especially regarding power supply and in CPU, memory or bandwidth [23]. These restrictions create challenges that are not present for general-purpose devices like workstations and laptops, such as communication and computational issues.

A communication problem in a smart home can, for example, be that battery-operated devices periodically go offline in order to save power. This can result in another device wanting to communicate with a device that is currently offline. This creates a need to be able to combine data since it cannot always be transmitted instantaneously. Another challenge for communication between devices in the smart home is the need to communicate with many different devices at different ranges, capabilities and that are made by different manufacturers. This keeps communication from being straightforward and creates a need for new or updated communication protocols where transmission only happens during specific periods with concatenated packets. These protocols will be discussed further in Section 3.2.

The restrictions on CPU and memory are the source of the computational challenges. Limited CPU resources may make traditional solutions for computation and security impossible to implement. For example, some cryptographic functions need a substantial amount of CPU resources to be fast enough to be practical. This means that there are specific design requirements for smart home devices in order to ensure that communication is possible.

Furthermore, the user needs to be able to interact with the devices in a smart home. There are multiple ways to design the smart home interface for the user. If the devices are WiFi-enabled then they are often provided with an accompanying smartphone app to control them. The apps are usually focused around being easy to use in order to be accessible for everyone. Smart home devices from the same manufacturer tend to be compatible with each other. This means they do not generally need separate applications to control them. Though, if a user gets devices from different manufactures, this often means different applications per device.

These days the building itself can have an integrated smart home systems built-in. It can include support for air-conditioning and surveillance. When the system is built into the house, the interface is often a physical device, like a dedicated touch screen. The control panels communicate with a central control system that controls the different connected devices.

Some companies on the market, such as Google and Amazon, have cloud-based support and provide an API that other manufacturers can use. They provide an interface, in this case Google Home Assistant and Amazon Alexa respectively, which enables the user to connect their devices to the interface if they are supported. Another benefit is that this allows for cross-compatibility between different brands without having to create multiple interfaces for the end-user. The user can then often control all the devices through a single application on their smartphone.

The final type of interaction mode is a hub-based system that has a central unit. This unit connects all the supported devices to a common interface, much like the cloud-based system. Another purpose of the hub is to behave as an interpreter or translator between the devices. It collects and translates various protocol communications from different devices in order to facilitate communication between, for example, a smartphone application and a smart lock. This is needed if the smart lock use protocols that are not native to a smartphone, i.e. protocols that are not Wi-Fi or Bluetooth. The benefit of a hub-based system over the cloud-based one is that the devices do not have to communicate over the Internet. The drawback in comparison is that since this architecture requires a physical hub it is more expensive.

3.2 Protocols

The Internet of Things has required the development of new communication protocols since IoT devices bring challenges like resource restrictions. An example would be that many IoT devices broadcast their wireless traffic on lower frequencies since such traffic penetrate walls better [24]. However, IoT devices communicate using a variety of protocols, ranging from simple unidirectional protocols to WiFi.

A technology specifically designed for IoT devices is Zigbee [25]. It is designed to be used in smart homes and is energy efficient due to relatively low transmission rates. The broadcasting frequencies are either on 2.4GHz or 900MHz. It is a non-proprietary protocol, which was developed by the Zigbee Alliance, and it is based on the IEEE 802.15.4 standard.

Another common IoT protocol is Z-Wave [26], which broadcasts on frequencies around 900MHz. The Z-wave protocol, much like Zigbee, focuses on low latency and reliability rather than maximizing data throughput. However, it is a proprietary protocol, owned by the Z-Wave Alliance, which means there has not been much open research performed regarding its security aspects. Though, in 2016 the Z-Wave Alliance issued the S2 standard and decided that all devices using Z-wave had to follow that security standard to get a Z-Wave certificate.

Finally, WiFi is frequently used in smart home devices since it makes the devices compatible with smartphones and other already existing equipment. This makes it easy for the user to get started without any additional gateway or similar products.

Despite there being a variety of different protocol stacks developed and used in smart home environments, all with their own advantages and disadvantages, this thesis will solely focus on TCP/IP traffic and protocols that are part of the TCP/IP protocol stack (such as ARP, DNS, WiFi, etc.). The following section will introduce the TCP/IP protocol and some of the many protocols it includes that are vital for understanding privacy threats to the smart homes today and/or the experiments conducted in later chapters.

3.2.1 TCP/IP

TCP/IP, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols that is used to interconnect network devices on the Internet or devices on a private network (see Figure 1 for an overview of TCP/IP). It functions as an abstraction layer between network applications and the routing/switching fabric. It specifies how data is exchanged over the network by providing communication protocols that identify how it should be broken into packets and segments, addressed, transmitted, routed and received at the destination.

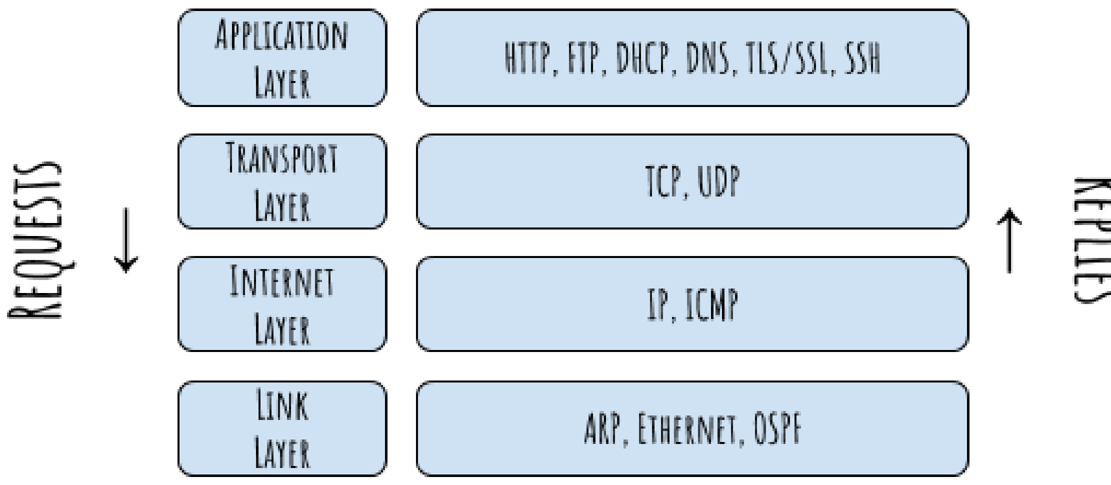


Figure 1: The TCP/IP Model which includes many well-known network protocols (not showing all protocols). Figure inspired by [27].

TCP/IP consists of several layers. The top layer, known as the Applications layer, contains a large variety of application protocols, including well-known ones such as DNS, HTTP/HTTPS and SSH but also encryption protocols such as TLS/SSL.

At the next layer, the Transport layer, we find the TCP and UDP protocols. These are responsible for opening, closing and managing a session between end-user application processes. The major difference between the two is that UDP is a so called unreliable protocol in contrast to TCP which instead is designed to make networks reliable, with the ability to automatically recover from the failure of any device on the network. TCP defines how applications can create channels of communication across a network. It also manages how a message is disassembled into smaller packets, so-called *segments*, before they are then transmitted over the Internet and reassembled in the right order at the destination address. None of the two transport protocols are superior to the other but instead serves different purposes, depending on how important reliability is compared to throughput.

At the Internet layer, the IP protocol defines how to address and route each packet to make sure it reaches the right destination. A router checks a packet's headers to determine where to forward the packet next which is also known as packet forwarding.

At the fourth and final layer, the Ethernet and the IEEE 802.11 (WiFi) protocols are found. It is at this layer a device determines whether a frame is destined for it or not.

Many consumer IoT devices natively support protocols that belong to TCP/IP, which makes them able to communicate with existing network infrastructure without the need for a central unit/hub. However, not all devices do. If not, they require a central unit/hub that can encapsulate the data from one protocol into a TCP/IP compatible format that can be routed on the network.

As seen, TCP/IP consists of many different protocols. However, some of these are more susceptible to privacy threats than others. The following subsections will focus on some of the protocols that are either more problematic in regards to privacy and IoT or crucial for the experiments in later chapters.

3.2.1.1 DNS

The domain name system (DNS) is a system for mapping domain names to IP addresses and vice versa. As described in the previous section, the Internet is based on the concept of IP-addresses in order to be able to locate specific machines/servers. However, IP-addresses are generally hard to memorize for us as humans, much in the same way as phone numbers are, and thus domain names make it more convenient (similar to memorizing the name of a contact in the phone instead of his or her phone number). The system is distributed around the world, stored on so-called domain name servers that all communicate with each other on a regular basis to provide updates and redundancy. With more than 350 million domain names listed at the end of the third quarter of 2019, storing all those records on a single server would not be feasible [28].

Another reason for why the DNS system is distributed is related to latency. It would simply take too long to get a response when making a request if there was only one location for the DNS database. To get around this issue DNS information is not only distributed and replicated among many servers, but is also cached locally on client computers.

Moreover, the system is also hierarchical, which means that a particular server only has the responsibility to know about the level below itself. An examples of this is that root servers (the top layer) know only where the top-level domain (TLD) servers resides, such as .com, .org, etc. TLD servers then know only where the domain name servers are, such as Google or Facebook and so on. This leads to multiple server lookups for a simple request such as www.google.com since it requires the root servers to find the .com-server, the .com-server to find the Google domain server and the Google domain server to find Google's web server (www).

The query can be done using iterative or recursive resolvers. For both types of resolvers, the DNS server will attempt to find the website in question in its local cache. The main difference is how it continues if the first query did not result in finding the website. If it is an iterative query, it will return with a message to query another server and give the address of that DNS server. As for a recursive request, it will query other DNS servers until it finds the address.

The domain name information is stored publicly, but the transactions performed by the hosts are not. This includes the specific information being queried and the identity of the nodes that queried the DNS. Unfortunately, there is no inherent mechanism in the DNS protocol to ensure encryption for these transactions. This means that the information can be logged by those who operate DNS name servers and resolvers. It can also be eavesdropped on by unwanted third parties. The DNS information might leak at multiple levels [29]. The different attack surfaces are represented in Figure 2 with different colour codings:

I. The communications links and devices between the stub resolver and the recursive resolver (blue)

II. The recursive resolver (green)

III. The communications links and devices between the recursive resolver and the authoritative DNS servers (red)

IV. The authoritative name servers (orange)

It is possible for an entity to passively collect all or part of the DNS transactions if they have access to the recursive resolver and the authoritative servers, or if they have access to the communication links or devices between the stub resolver and the recursive resolver [29].

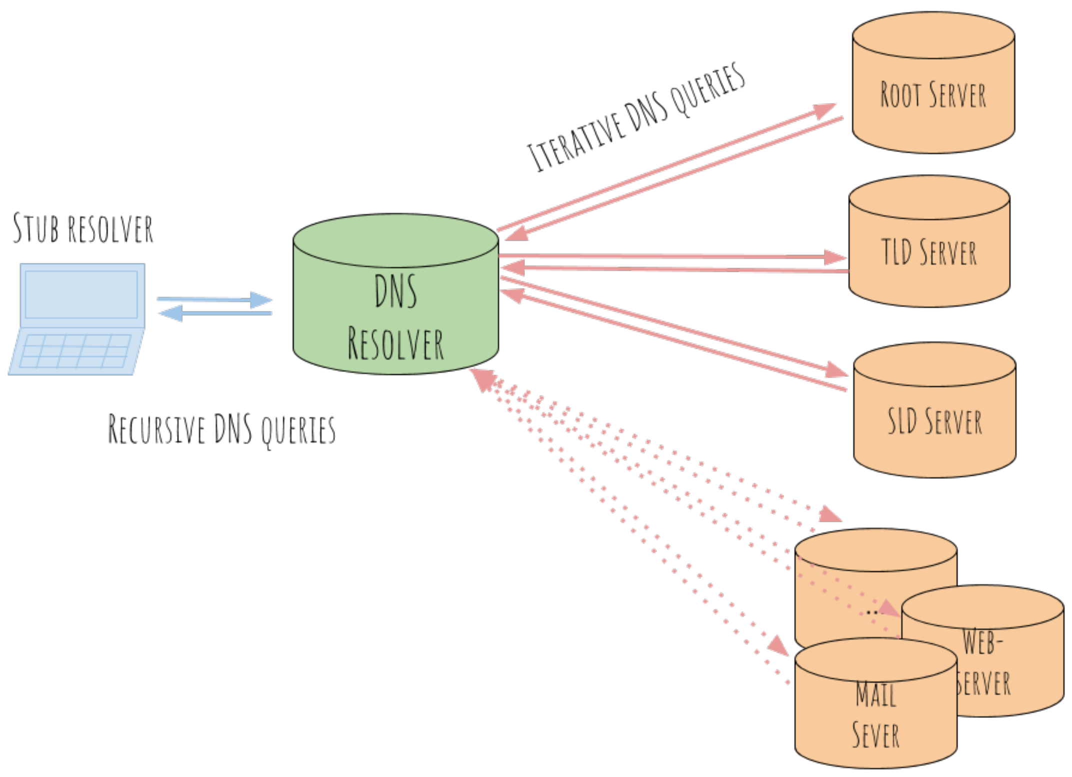


Figure 2: An example of how a typical DNS request might look. The colours in the figure highlight the different attack surfaces to the system.

As DNS lookup requests can be logged and reveal which endpoint a certain IP requested, one can argue that it is of importance which provider one chooses to rely on for DNS resolutions. Some well known public DNS revolvers are Google's 8.8.8.8 or Cloudflare's 1.1.1.1. As seen in Chapter 2, it is not uncommon for smart home devices to have predefined/hard-coded DNS addresses which can not (easily) be modified. This could arguably be considered problematic in terms of privacy.

A resolver can also filter unwanted DNS-requests. One such filtering technique is by using a so called DNS sinkhole, where unwanted requests are redirected to sink-holes and thus prevented.

3.2.1.2 ARP

The hardware/physical/link layer address of a NIC is known as the Medium Access Control (MAC) address. When a NIC is manufactured, it is allocated a globally unique 6-byte link address (stored in a PROM). This is the link source address used by an interface when sending messages. When a computer then sends IP packets, it also includes its link address in the frame part of the message. This enables it to receive all packets that match the same hardware address in the destination field (or one, or more, pre-selected broadcast/multicast addresses).

The address resolution protocol (ARP) is an IPv4 protocol for mapping the hardware address associated with a specific IP-address to that IP-address. The device that wants to know the physical address associated with a specific IP-address sends a broadcast ARP message to the nodes in the same broadcast domain, requesting information about who has the specific IP-address. In the message, the physical address and the IP-address of the sender are both included. This helps the receiver know where to send the reply as a unicast message.

This means that a link layer address is dependent on the interface card that is used on the device. In contrast, IP operates at the Internet layer and is not dependent on the link addresses of individual nodes. This is why the address resolution protocol (ARP) is used to translate between the two types of addresses.

In order to reduce the number of address resolution requests, a client typically caches resolved addresses for a short period. The ARP cache is of finite size and is periodically flushed of all entries. This also removes any unsuccessful attempts to contact computers that are not currently running. The ARP cache can be used by other hosts to detect that a host changes what IP address it associates with a certain MAC addresses since an entry will be deleted and a fresh ARP message will be sent to establish the new association.

Since a smart home contains devices that communicate with each other, their manufacturers and third-party service providers, it can be valuable to monitor the network activity. It can be done by exploiting the design of the ARP protocol and using a technique known as ARP spoofing. This concept is discussed in Chapter 4 and plays a crucial role in the experiments conducted in later chapters.

3.3 Back-end Services

When the user asks something out of their IoT device, this request will sometimes require the device to forward the request along to back-end servers. This is because IoT devices generally have a comparably low CPU performance and thus have to delegate more complex calculation, for example encryption or data analysis, to other entities. These entities usually reside on the cloud for more computational ability and to be able to service a large number of devices at once.

However, many IoT device manufactures and software developers will rent this server space from other big companies in order to not have to worry about the required infrastructure themselves. Examples of these large server companies would be Amazon and their Amazon Web Services [30] and Microsoft with Azure [31]. They offer three different models of cloud service: Platform, Software and Infrastructure as a Service (SaaS, PaaS and IaaS, respectively). The features of these services can be seen and compared in Figure 3.

This section thus aims to describe what these services are and highlight what role they have within the field of smart homes in regards to privacy.

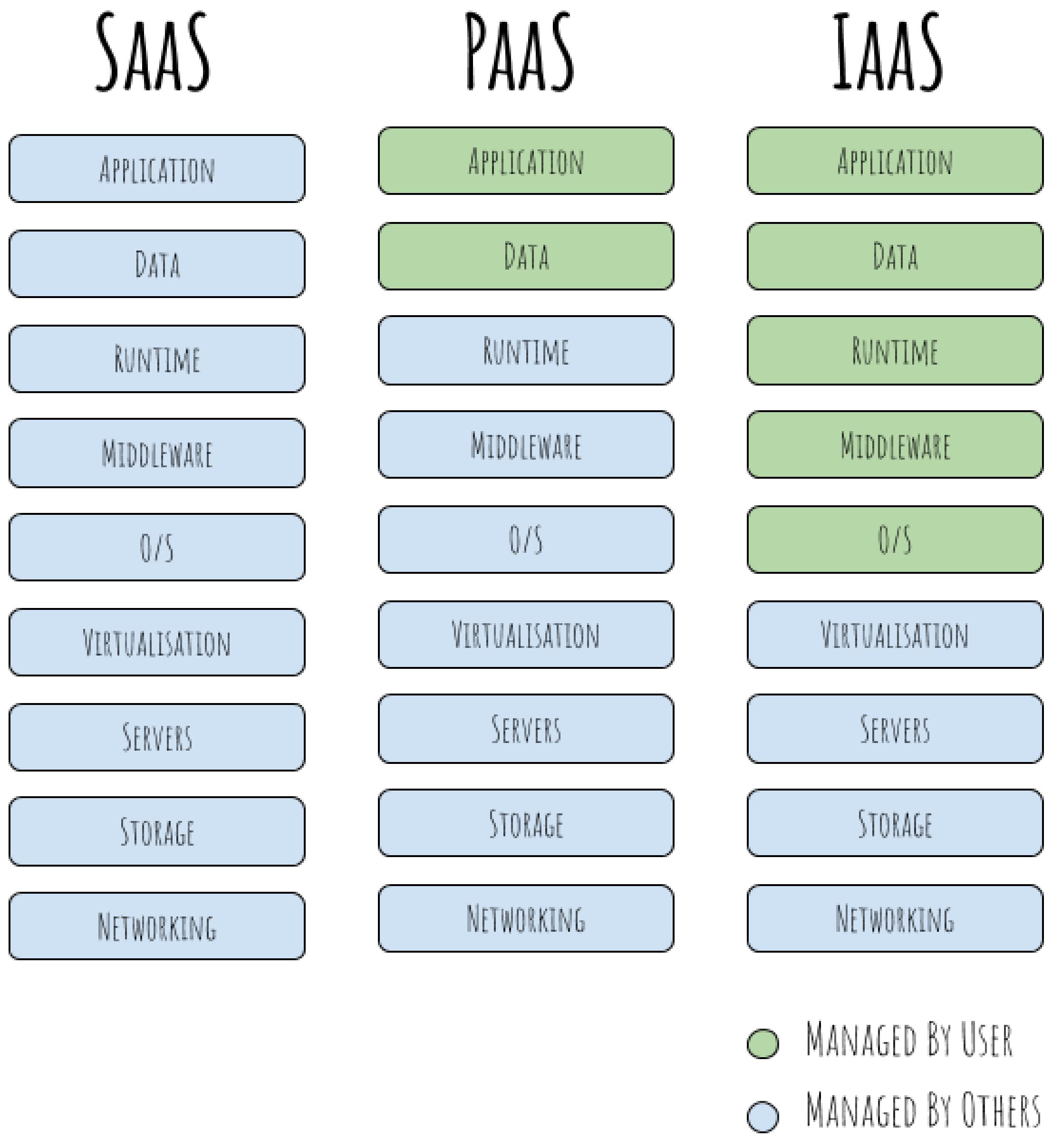


Figure 3: The difference between SaaS, PaaS and IaaS, inspired by [32].

3.3.1 Software as a Service

Software as a Service is the most commonly used cloud service option for businesses [33]. It uses a web delivery model to distribute applications to the user while allowing third parties to manage the applications. Most SaaS applications will be run through the user's web browser, so they do not need to be downloaded and installed by the client. This also means that the third-party vendors will handle any technical issues on top of handling the data storage and servers. Since this removes most of the responsibility from the manufacturer of the device or developer of the software, it creates a streamlined experience for them.

On the other hand, SaaS solutions also mean that the control of both the service's performance and security is handed over to the cloud provider. This is not limited to the software, meaning how it is updated, but also the data collected by the applications. Moreover, large volumes of data may need to be exchanged to the data centres that host the software. This means that sensitive information can be sent to public cloud-based SaaS services.

3.3.2 Platform as a Service

The idea behind PaaS is similar to that of SaaS. Except PaaS provides a platform for software creation, rather than offering software [34]. It enables the user to access everything between simple apps to more sophisticated enterprise applications on the cloud. The PaaS provider will host the software and hardware on its own infrastructure. This makes it so that developers using PaaS do not have to worry about getting any specific hardware or software themselves.

PaaS does not typically replace a business's entire IT infrastructure. Instead, it is used to incorporate different underlying infrastructure components with the business own infrastructure, such as operating systems, databases and servers. These functions are owned and maintained by the PaaS provider. Another benefit of PaaS from the developer's perspective is that it provides additional resources, like programming languages and libraries but also database management systems.

In regards to the privacy concerns for the user of any PaaS model it is the same as it was for SaaS since the data is still stored on third-party web servers.

3.3.3 Infrasturcture as a Service

Infrastructure as a Service provides the same capabilities as a traditional data centre without forcing the user to physically maintain or manage it [35]. The customer can still access their servers and storage directly, but it is all outsourced to the cloud using virtualisation technology. It allows businesses to purchase resources as they need them instead of buying the hardware. It is often provided to the customer through a dashboard or an API.

In comparison to SaaS and PaaS, the customer is responsible for managing things like applications, operating systems and middleware. However, IaaS providers manage the virtualisation, storage and servers. Sometimes providers also offer services beyond the virtualisation layer, message queuing and databases.

Security threats can still be sourced from the host even if the customer is in control of the platform for the operating system, data and apps. System vulnerabilities can expose data communication between the host infrastructure and the virtual machines to unauthorised entities.

CHAPTER 4

Privacy Protection Concepts

This chapter explains some useful concepts for privacy protection: packet analysers/sniffers, ARP spoofing, intrusion detection/prevention systems as well as log management and visualisation tools. Some of these concepts can however also be used for malicious purposes, such as packet sniffing and ARP spoofing, although, in this context, they are discussed as protection concepts. Nevertheless, this chapter is going to explain what their purpose is and how they differ from one another, which means that this section will not be about a specific software implementations of the concepts, but rather about the underlying ideas different software solutions are based upon. This is used to provide a basis for the experiments presented in later chapters.

4.1 Packet Analysers and Packet Sniffers

A packet analyser or sniffer is a software or hardware tool that can be used to monitor network traffic and troubleshoot network issues. There is a semantic difference worth noting between packet sniffers and packet analysers even if the two terms are often used interchangeably. A packet sniffer records packets observed on a network interface, while an analyser looks at the packets and attempts to make some inferences about what they contain. In practice, the most commonly used programs perform both functions meaning that they both log and analyse the packets. Examples of this would be Wireshark [36], Snort [21] and Princeton IoT Inspector [20].

A packet sniffer is composed of two main parts: a network adapter to connect the sniffer to the network the user wants to analyse and the actual software to provide a way to see, log and analyse the collected data. A packet analyser can reveal how the traffic is shared between nodes and what the traffic pattern is on the network. It does this by intercepting, capturing and analyzing packets that flow through the network. The data is then presented in a form readable to the user for network analysis.

There are two main types of packet sniffers: hardware and software packet sniffers. This thesis will solely focus on the software version since the hardware-based packet sniffer is often single-purpose and require dedicated hardware. This means it is less accessible to the average user, which is something we wanted to keep in consideration throughout the thesis, as stated in our Aims (see Section 1.3).

A software packet sniffer is an application that is run on the host computer that utilises the computer's network hardware to perform the packet capture. Typically, a network interface card (NIC) is set to only be able to read packets that are destined for itself, i.e. when the target IP address of the packet matches its IP address. However, when a NIC is configured for network sniffing the network card is set to a configuration mode commonly called promiscuous mode. This mode allows the NIC to intercept and read any arriving network packet in its entirety. Once it is in this mode, the packet sniffer can separate, reassemble, and log all packets that pass the interface, regardless of their destination addresses.

The amount of information that can be captured by the sniffer depends on if the sniffing is done over a wireless or wired connection. If the former is used, and by having set the NIC in promiscuous mode, it is able to capture any packets from other wireless clients on the same network that are using the same channel [37]. The reason to this has to do with the nature of wireless signals, i.e. they propagate in all directions and can thus be intercepted by anyone on the same network who are listening on that channel (has their NIC in promiscuous mode). However, packet sniffers are usually limited to capturing one channel at a time. Though if the computer the sniffing software is running on has multiple wireless interfaces, this allows for multi-channel packet capture. This could be relevant if the network in question has multiple access points that transmit over different channels.

In contrast, on a wired network, a packet analyser will see different things depending on if the network is built using hubs or switches [38]. Hubs receive incoming packets on one port and broadcast them to all other ports by default, meaning they are ideal for promiscuous network monitoring/sniffing. If the network uses switches instead, things are not as trivial. For the packet sniffer to be able to capture the packets, the switches need to manually be configured to allow port mirroring. Port mirroring is a feature that allows the switch to redirect the traffic that occurs on some or all ports to a designated monitoring port. This monitoring port copies the network packets that are detected on the designated ports and sends them to a network monitoring connection on another switch port to which the sniffer is connected. The NIC on the computer running the packet sniffer will operate as if it were in regular mode unless the switches are configured to allow port mirroring. This means that if multiple switches segment a network, they all need to be connected and have port mirroring active to enable sniffing of the entire network.

4.2 ARP Spoofing

In computer networking, spoofing means that a party manages to be identified as another party by falsifying data to obtain an illegitimate advantage [39]. In regards to ARP spoofing, it is a technique where the attacker sends falsified ARP messages over a local area network. It can only occur on local area networks that utilise the Address Resolution Protocol (IPv4) and the attacker attempting to perform ARP spoofing needs to have direct access to the LAN segment they wish to spoof. It is done by exploiting the lack of authentication in the ARP protocol. In this case the attacker exploits that if a unicast ARP reply is received by any NIC then it is automatically cached regardless of if an ARP request has been sent in the first place. The results of ARP spoofing is the linking of the attacker's MAC address with the IP address of another legitimate computer or server on the network, for example the default gateway. This means that the attacker is now pretending to be the default gateway.

Once this is done, the attacker will begin receiving any data that is intended for that IP address. It enables a third party to intercept, modify and even stop the traffic. Access to the LAN can be granted by either using a compromised host on the LAN, or from the user's machine that is connected directly to the LAN they wish to target.

It is possible to perform ARP spoofing after a MAC flooding attack. In that case, the method is slightly different than the one described above. The attacker opens an ARP spoofing software tool and sets the tool's IP address to match the IP subnet of the target LAN. The attacker then scans for the IP and MAC addresses of hosts in the target's subnet. The tool constructs a large number of forged ARP request and reply packets to overload the ARP table belonging to the switch. This sets the switch to forwarding mode and makes it send messages on all ports, making the switch act more like a hub. The attacker can now pick a target and begins sending the forged ARP packets across the LAN that contain the user's MAC address and the target's IP address. Other hosts on the LAN then cache the spoofed ARP packets, adding them to their own ARP table. This means the data that those hosts send to the spoofed network will go to the attacker. This means that all packets on the network will go through the user. These packets can only be read if the NIC used for the spoofing is set in promiscuous mode, as mentioned in Section 4.1.

4.3 Intrusion Detection and Prevention Systems

An Intrusion Detection System (IDS) and an Intrusion Prevention Systems (IPS) have many similarities. While both monitor and alert on suspicious network traffic, the main difference is that the IPS can be used to also prevent traffic. Examples of software that use these concepts are Snort and Suricata [40].

In order to decide if traffic is malicious or not, they can either use signature-based or anomaly-based detection. Signature-based detection uses a database of known threats for pattern matching, while anomaly-based detection instead looks for anomalies in the traffic that might indicate suspicious activity. Anomaly-based detection can be based on machine learning algorithms which allow the detection software to make judgement calls on unknown traffic that is not yet defined. They both have their drawbacks however.

A signature-based IDS/IPS requires constant attention and adaptation to its signature database to stay up to date, whereas an anomaly based IDS/IPS relies on algorithms, data and computational power in order to be effective. Both might pose a problem for a regular user, where lack of updates is common, and disinterest in spending money on expensive equipment necessary for enabling anomaly based detection could be a hindrance. This highlights the need to look for solutions that would put less responsibility on the actual user. Signature and anomaly-based detection are large topics on their own, which is why this thesis is only looking at signature based solutions.

The placement of the IDS/IPS is also important. It can be placed differently in relation to the firewall depending on the use case (whether one is interested in analysing internal or external traffic). However, many IPS vendors even integrate IPS systems with firewalls that combine the functionality of the two into a single unit.

4.4 Logging and Visualisation of Traffic

If all traffic is logged by the IDS/IPS, then the result might be noisy and hard to interpret without the right tools. There is also the risk that one packet can trigger multiple rules/matching multiple signatures, meaning that the resulting log volume from the logged events might be even larger than the monitored traffic data. Though, even without packets triggering multiple rules, a typical IT environment tends to generate large traffic volumes. Thus, it would be a time-consuming task to search through the resulting alert logs manually. Luckily, there are log management and visualisation tools to assist in this task. By using such tools, it is easier for the user, or a developer of IDS/IPS software, to get a bird's view perspective on the traffic and more easily distinguish threats.

Visualisation tools today range from terminal-based ones like tcpdump [41] to GUI based like Wireshark [36]. Others are more suitable to large amount of data, like Splunk [42] or Kibana [43]. This is why the user needs to consider what the visualisation tool will be used for specifically. For example, a versatile visualisation tool can offer different visualisation options such as graphs and chart that can help in getting a better understanding of the logs by presenting them in different ways.

CHAPTER 5

Design of the Experiments

This chapter describes the design, methodology and motivation behind the different experiments presented in Chapter 6.

It begins with a section discussing the purpose of the experiments, how they align with the aims introduced in Chapter 1 and relate to the concepts described in Chapter 4. This is followed by a section describing the methodology behind the comparisons, i.e. how and why different devices were grouped together. The third section presents which device categories were used throughout the experiments and why certain categories/devices were chosen instead of others. Finally, the chapter concludes with a discussion on and motivation to the software used in the experiments.

5.1 Experiments: Purpose and Design

The purpose of the experiments was to answer the remaining aims of the thesis, more specifically aim II - IV. In order to achieve this, three different experiments were conducted, each corresponding to at least one of the three remaining aims.

This section begins with a subsection dedicated to describing how the experiments map to the different aims of the thesis. It is followed by three separate subsections discussing the general idea and design choices behind each of the three experiments.

5.1.1 Mapping the Experiments to the Aims

The different experiments map to different aims, which can be seen in Table 5.1.

From Table 5.1, one can see that the second aim was handled by the first two experiments, where the traffic from the different smart home devices was logged and analysed. Using two different software solutions to analyse the traffic also helped with reducing the risk of missing privacy threats during the testing. The testing was

Aim \ Experiment	Experiment I	Experiment II	Experiment III
Aim II	X	X	-
Aim III	X	X	-
Aim IV	-	X	X

Table 5.1: How the different experiments map to the remaining aims of the thesis.

done for roughly twenty days (half of the time dedicated for devices aimed for the EU market and another ten days for the other devices) and consisted of a mixture of active and passive testing, much like a real smart home environment.

The third aim was also tackled by the first two experiments. Two quite different software solutions (based on two largely different concepts) were chosen. This was done in order to explore a wider range of software solutions and be able to reason about their strengths and weaknesses as well as usability for different kind of users. However, since no baseline existed, the performance of the two software solutions was evaluated by comparing them to one another.

The fourth and last aim was met with the second and third experiment where rules were developed on the cloud and pulled on a regular basis by the server running the local IPS. This was finally tested in order to confirm that the integration of the cloud developed rules worked as intended.

5.1.2 Experiment I - ARP Spoofing

The first experiment used the concept of ARP spoofing to intercept the network traffic. By exploiting the lack of authentication in the ARP protocol, we could spoof the IP address of the router and end devices on the network and intercept the communication between them. This was done with the software Princeton IoT Inspector in order to see how well it performed in identifying threats from the threat model on the given traffic (see Section 5.4.1 for further information about Princeton IoT Inspector). An overview of the conceptual design behind the first experiment can be seen in Figure 1.

The first experiment served two main purposes:

a) Explore how successful Princeton IoT Inspector is at recognising suspicious traffic in regards to privacy and IoT.

b) Contribute to the scientific community by sharing the data generated by the devices in our testbed with the researchers at Princeton.

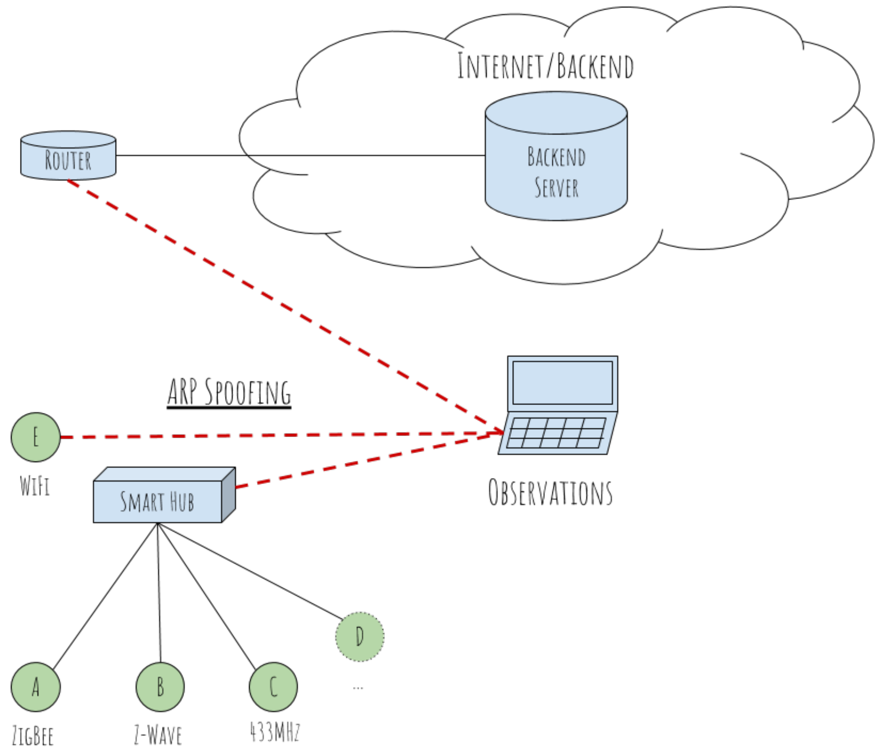


Figure 1: The conceptual design of the first experiment. The red dashed lines in the figure indicates the affected flow of traffic.

5.1.3 Experiment II - Local IPS

The goal of the second experiment was partly to investigate another commercially available software, based on a different security concept than the one from the first experiment, in order to see how well it performed compared to the first solution. For that reason, the concept of an IPS was used in the second experiment. By placing an IPS (inline) on the network, the traffic was forced to pass through the IPS and could thus be observed/logged/prevented. For the conceptual design of the second experiment, see Figure 2.

There were several reasons why the second experiment used an IPS as the underlying concept. First, its ability to block traffic (even though that particular functionality was not used in the second experiment) is an added benefit for the regular user. Secondly, an IPS is a concept quite different from ARP spoofing. Thirdly, we researched a few different IPS and IDS solutions and concluded that they are typically more customizable and expandable compared to Princeton IoT Inspector, which was important for the third and final experiment. Thus, the second experiment served two main purposes:

a) Explore how successful a modern IPS signature-based software is at detecting (and theoretically preventing) suspicious traffic in regards to privacy and IoT.

b) Work as a platform that can be improved in the third experiment (fetch rules from a third party to improve its signatures).

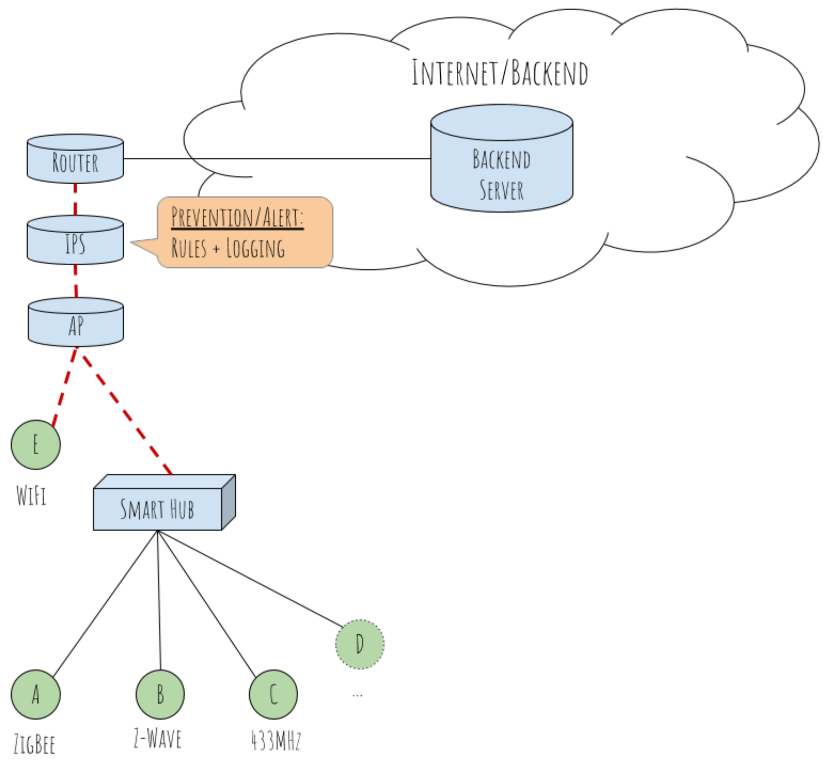


Figure 2: The conceptual design of the second experiment. Affected flow compared to the first experiment is marked by red dashed lines.

5.1.4 Experiment III - Third Party Rules

The third experiment was designed as a proof of concept to try to emulate the role that a centralized/cloud-based IDS solution might play for improving the effectiveness of a local IPS such as the one from the second experiment. By pushing metadata of the data generated by the smart home devices on the local network to the cloud, we simulated a user supplying a service provider with their own user data. Next, rules could be developed on the cloud and pulled on a regular basis by the local IPS. This emulated a user subscribing to a service while at the same time supplying it with data. See Figure 3 for an overview of the conceptual design of the third experiment.

The reason why only metadata were sent had to do with scalability. If actual payload were to be included, it would have created a vast amount of traffic. That would have been particularly troublesome from a scalability perspective, given the predicted growth rate of traffic generated by smart home devices mentioned in Chapter 1. Nevertheless, the metadata did include timestamps as well as information about the size of the payload which made it possible to draw conclusions on the shape of the traffic.

Worth noting, the rules developed on the cloud IDS were based on data from a single network. In a commercial environment, the rules would instead be based on multiple users' data. This is of importance to highlight, as the rules might otherwise just as well have been implemented on the local machine directly. Nevertheless, the advantage of using a centralized/cloud-based solution is twofold:

a) Rules can be developed based on data from a large number of users compared to a single user (rules could be developed using statistical methods)

b) Rules could be developed by security/privacy experts that might otherwise be hard, if not impossible, for a regular user to match (rules could be developed using expertise)

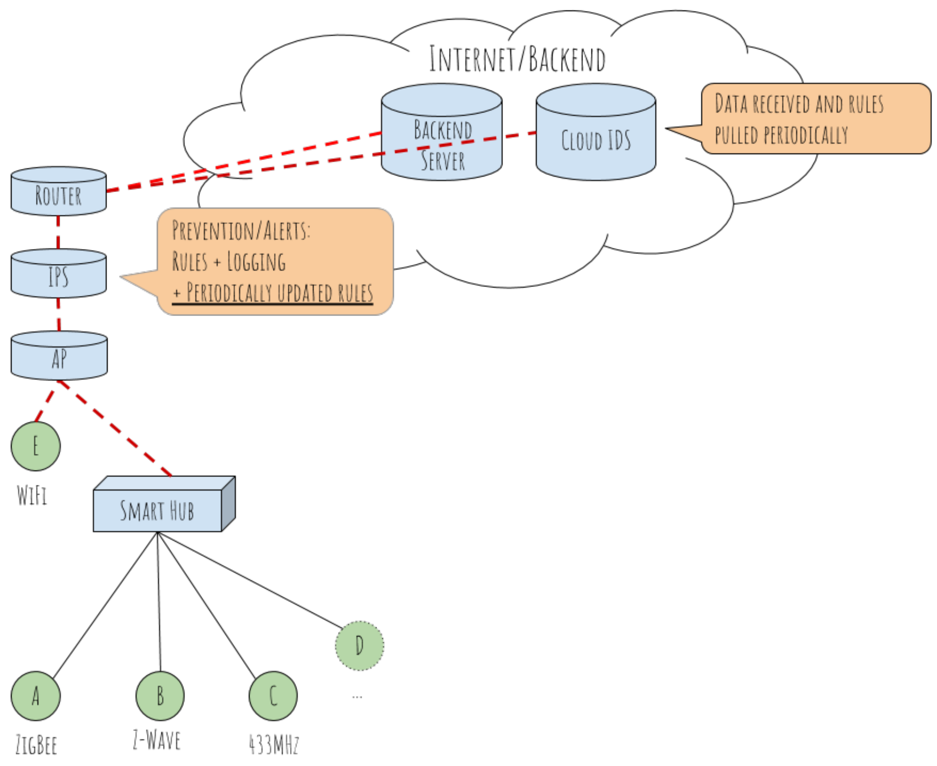

Internet/Backend
Backend Server
Cloud IDS
Data received and rules pulled periodically
Router
IPS
AP
Prevention/Alerts: Rules + Logging + Periodically updated rules
E
WiFi
Smart Hub
A
B
C
D
...
ZigBee
Z-Wave
433MHz

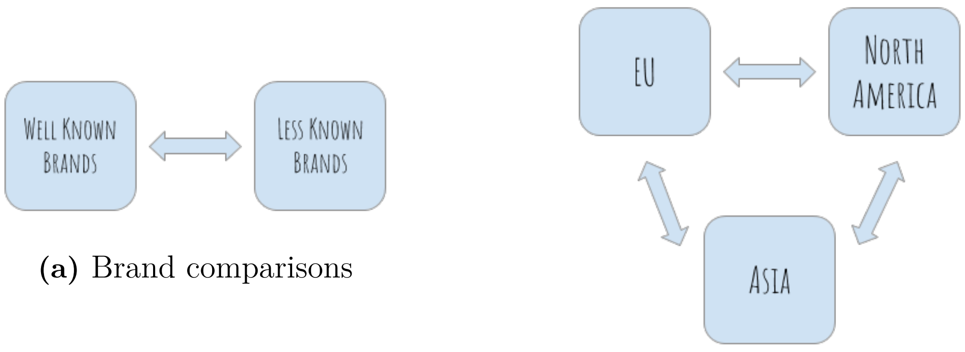
Figure 3: The conceptual design of the third experiment. Affected routes compared to the second experiment are highlighted by red dashed lines.

5.2 Segmentation

In order to be able to more easily reason about privacy differences in regards to brands and regions, a conceptual segmentation of the different devices was beneficial to develop. The segments and comparisons in the different experiments were partly based on brands, where one segment consisted of well-known and another one of lesser-known brands aimed for the EU[1] market. Furthermore, we also divided the

[1]We chose to include UK devices in the EU segment. The reason for this is that we assume that those devices behave similarly to other EU devices as the UK relatively recently left the EU and that their market still is strongly intertwined with other EU countries' markets.

devices into three regional segments: EU, North America and Asia. The North American segment consisted of products from the United States, the EU segment of products from Sweden, the UK and Germany and the Asian of products from China. The reason why these particular countries were selected had to do with the fact that the United States and China can be considered to be the most influential countries in their respective regions. Sweden, the UK and Germany were chosen for reasons of easy access (either easy market access or already owned devices). For an overview of the different segments, see Figure 4.



(a) Brand comparisons

(b) Region comparisons

Figure 4: The segmentation in regards to well known and less known devices within the EU (left) as well as to regions (right).

As previous research has indicated, there can sometimes be differences in behaviour between the same product aimed for different markets. For this reason, the products were primarily bought from marketplaces/websites located in the respective region in order to try to account for this.

5.3 Testbed Devices

As mentioned in Chapter 2, the smart home is highly heterogeneous, and there are many different device categories to choose from. The chosen categories and how they correlate to the different segments described in Section 5.2 can be seen in Table 5.2. For a detailed list of all devices and hardware used throughout the experiments, see Appendix A.

The main reasons for why these particular categories were chosen instead of others were either or both of the following:

a) They can be considered to be some of the most intrusive categories in regards to privacy since they do contain microphones and/or optics/cameras.

Segments \ Categories	Streaming devices	IP-Cameras	Home automation	Smart speakers
EU: Well-known Brand	X	X	X	X
EU: Less known Brand	-	X	X	-
EU	X	X	X	X
North America	X	-	-	X
Asia	-	X	X	-
Brand Comparison	-	X	X	-
Regional Comparison	X	X	X	X

Table 5.2: The device categories being used throughout the experiments, which segments they belonged to and which comparisons were done.

b) They have a wide user base or stood out in previous research (see Section 2.1).

5.4 Software

There is a wide variety of information security programs to choose from and given the nature of the different experiments, several aspects needed to be considered before deciding on which ones to use. In this section, we will present the software solutions that were chosen, give an overview to how they work and explain for what reason those particular solutions were chosen.

5.4.1 Princeton IoT Inspector

For the first experiment, we needed a software that was able to capture all the traffic on the network while still being easy to use. Initially, we looked at Snort in packet sniffer mode since our literature review resulted in many papers that used Snort as a tool for packet sniffing. However, while it is well known, documented and has all the functionalities needed for this task, it is not the most intuitive software for the inexperienced user. Since our first experiment was not solely focused on analysing the problem, but also doing so in a user-friendly way, we needed a solution that was more simplistic and included a graphical user interface (GUI), which Snort did not.

While looking for other options we came across Princeton IoT Inspector, which met the requirements above. We perceived it as being user friendly and suitable to use even for users with little to no experience in network security.

Princeton IoT Inspector is an open-source network traffic analyser software developed by researchers in the United States and Europe [18] that uses ARP spoofing (see Section 4.2) to monitor network traffic [19]. However, a requirement for using the software is to allow it to collect the traffic from the participants' smart home devices for research purposes. It collects a variety of data like traffic statistics, scrambled MAC-addresses, etc. Nevertheless, no payload data is being collected, except for DNS, DHCP Request, and TLS Client Hello [19]. It also allows the user to download all the collected data in JSON format.

The software offers the user the option to get a detailed look into which endpoints the devices connect to. It also helps the user to visualise the shape of the traffic. For instance, it shows how frequently certain devices are active as well as the size of the packets in their communications. Moreover, the software also offers the user to see if the communication was sent encrypted or not and if any device connects to known trackers or advertisement servers.

With the software it is easy to get an overview of how the IoT devices behave on the network. By doing so, anomalies in network traffic can be detected. For example, a significant amount of traffic to third parties could be detected but also signs of devices being hijacked when there is activity when none is expected.

Nevertheless, what Princeton IoT Inspector does not offer is a means to get information about what data is actually being sent. As an example, if a microphone transmits a non-initiated recording, whether the message simply consists of a simple keep-alive message or not could be considered significant in regards to privacy. However, since it displays the shape of the traffic, one could likely figure this out by looking at the size of the packets.

5.4.2 Snort

For the second experiment we required an IPS software. This made the selection straightforward since we already considered using Snort for the first experiment. While we had picked Princeton IoT Inspector because of its user-friendliness and the fact that it was based on the concept of ARP Spoofing, we chose Snort for its dissimilarity to Princeton IoT Inspector. Snort is a bit more complex than Princeton IoT inspector but instead allows for a higher degree of customisation.

Snort is a well known and free open source IDS and IPS software [44]. These days it is developed by Cisco [45]. There is a lot of unpaid community support from the user base and a set of community rules that the user can download so they do not have to set it up themselves [46]. Since it is developed by Cisco, it has the full support of Cisco's different data centres and file reputation system. For example, the Cisco Talos Intelligence Group supplies Snort with a large database of traffic signatures that are common network attacks or other forms of malicious activity and compares all incoming traffic to that database automatically. This team are the authors behind both the official Snort Rule Set [47] and the Snort Subscriber Rule Set [48]. Both require licenses, but the latter is a paid service where your Snort rules get updated by those written by the Talos group as soon as they are added. There is also a free community ruleset available, which is a subset of the subscriber ruleset, that one can use without the need of any license [49]. This is the ruleset that is used in this thesis and referred to as *community rules* throughout the thesis.

Snort can perform real-time traffic analysis and packet logging on IP networks and has the ability to perform protocol analysis, content searching and matching. It can be configured in different modes to act as either a sniffer, packet logger or to perform network intrusion detection/prevention. In the latter mode, Snort can also perform a specific action based on what has been configured in its rule file.

In our experiments we used the third configuration mode to allow it to monitor network traffic, analyse it against rule sets defined by both us and the community and to (theoretically) block undesired traffic. Moreover, there are also different versions of the software available. Although in beta (spring 2020), we chose to use Snort version 3 since it has native support for JSON as an output format. This proved to be a convenient format when using Snort in combination with other software solutions.

Snort also offers plugins focused on the application layer such as the OpenAppID. It is an open, application-focused detection language and processing module designed specifically for Snort by Cisco, and it enables users to create, share, and implement application detection.

When it comes to the Snort rule language, it is highly flexible. The rules are evaluated in a predetermined order in regards to actions, rather than in the order that they appear in the snort rule files [50].

By default, the order of the actions are:

- Pass rules: Snort ignores the packet.
- Alert rules: Snort generates an alert using the alert method.
- Log rules: After generating an alert, Snort then logs the packet.

Though this predetermined order only determines order between rule groups (pass or alert, for example). Within those rule groups, the rules are read in order and executed accordingly [50].

In our experiment we were conscious of the fact that we needed to look at a large number of packets. However, Snort does not come with any GUI. Furthermore, the default output method writes the capture information to a file and consistently updates the file when something new is detected. Going through this manually afterwards would be cumbersome. This meant that we needed a better way to visualise the logs.

5.4.3 Elasticsearch, Logstash and Kibana (ELK)

In order to determine what tools to use for visualising the logs and alerts from Snort, different articles were read on the topic to see if there was a consensus about how to best visualise the data [51][52]. Based on this research, we decided to use the ELK stack [53].

ELK is an acronym for three open source projects: Elasticsearch [54], Logstash [55] and Kibana [43]. The three of them work together to become a software stack that focuses on log management and data analysis. It can monitor modern applications and IT infrastructures which is useful since these are often highly distributed, dynamic and noisy environments and is not limited to Snort logs exclusively.

Elasticsearch is the scalable search and analytic engine that stores the data. It is based on the Apache Lucene search engine [54]. Logstash is a server-side data processing pipeline, or log aggregator, that collects data from various input sources, executes different operations on the data and then sends it to various supported destinations. In the case of ELK, this means Elasticsearch. Finally, Kibana is a visualisation layer that works on top of Elasticsearch, providing users with the ability to visualise and analyse the data.

Initially, we were mainly interested in the visualisation part of ELK, meaning Kibana. However, after looking into Elasticsearch and Logstash we realised they would also be beneficial to our log management. Logstash assists in determining how the data looks and can be used for further queries like querying for geodata and domain name resolutions based on the IP addresses in the logs. Moreover, Elasticsearch indexes the data which enables quick searches in the database, this is increasingly important as the number of logs increase.

One of the main reasons to why Kibana was the choice of preference was related to its many ways to visualise the data. Moreover, another good trait of Kibana is its ability to filter the data by different metrics, for example, by time. As long as the logs contain timestamps, one can choose to look at smaller or larger time intervals.

Finally, another reason for why Kibana is useful when studying Snort logs is that it has an easy way of looking at individual logs. Given that one of the threats in the threat model was to investigate unexpected device behaviour, we also needed to be able to inspect the traffic in more detail. Being able to find and isolate specific entries proved to be useful.

5.4.4 Cron

For the last experiment it was necessary to automate the process of sending parsed logs to the cloud, fetching the latest cloud rules and restarting Snort on a regular basis. In UNIX there is a utility called Cron that is designed for this purpose, i.e. to automate repetitive tasks, which was used in the third experiment.

CHAPTER 6

Implementation of the Experiments

This chapter features the implementation of the three experiments whose designs were discussed in the previous chapter. It begins with a description of the usage of Princeton IoT Inspector in the first experiment. This is followed by a section discussing the use of Snort and the ELK Stack in the second experiment. The chapter is concluded with a section dedicated to the third experiment, which presents the different Python scripts that were developed and automated in order to improve the setup from the second experiment.

6.1 Experiment I - Princeton IoT Inspector

The first experiment consisted of capturing packets by using the Princeton IoT Inspector on a Windows 10 machine. As mentioned in the previous chapter, this software utilises the concept of ARP spoofing to capture the packets. By doing so, all traffic to and from the devices passed through Princeton IoT Inspector before it reached its target. However, the software does not modify the data in any way, which means that the process is transparent to the devices. The packets that were captured could then be visualised through a web GUI that was provided with the Princeton IoT Software. An overview of how the first experiment was setup can be seen in Figure 1.

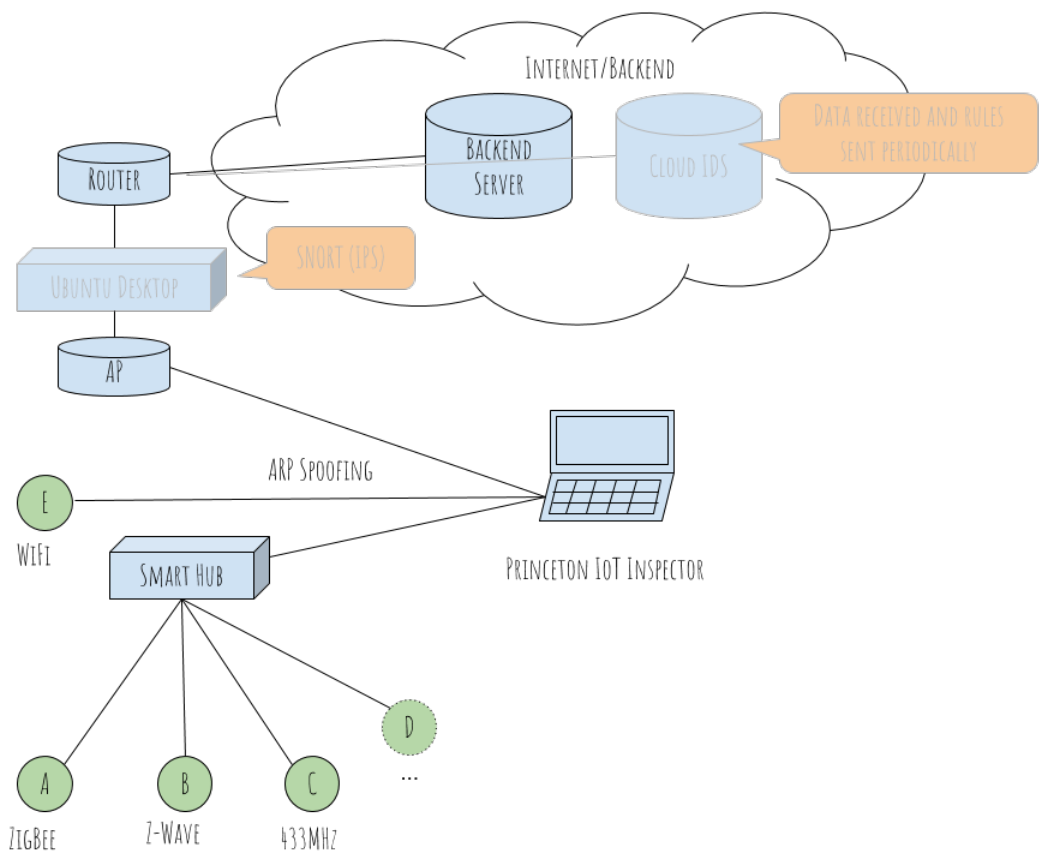


Figure 1: The testbed was designed to support running all three experiments in parallel. Non-essential equipment in regards to the first experiment have been greyed out in the figure.

6.2 Experiment II - Snort & The ELK Stack

The second experiment used Snort in IPS mode to capture the packets. However, its blocking capabilities were not used in this experiment. Instead, it rather ran as if it was an IDS. The IPS mode was chosen primarily to support the third experiment (cloud rules) as well as for the fact that it can be considered a more viable option for regular users. For an overview of the setup for the second experiment, see Figure 2.

Thanks to the way Snort is designed, it is possible to include multiple rule files, which meant that we could separate and use both local rules (written by us) and community rules (downloaded from the Snort community) as well as cloud rules (written by us, simulating a service provider). The community rules are constantly evolving; however, the rules used throughout this experiment were downloaded 2020-05-07. The local rules were designed to ignore traffic from non-IoT devices such as laptops, desktops and smartphones. Furthermore, most common local traffic such as IPv4 and IPv6 multicast and link-local addresses were filtered as well. All other traffic, that was not already being tagged as alerts by Snort, was then logged. See Appendix B.2 for the local Snort rules developed for the second experiment.

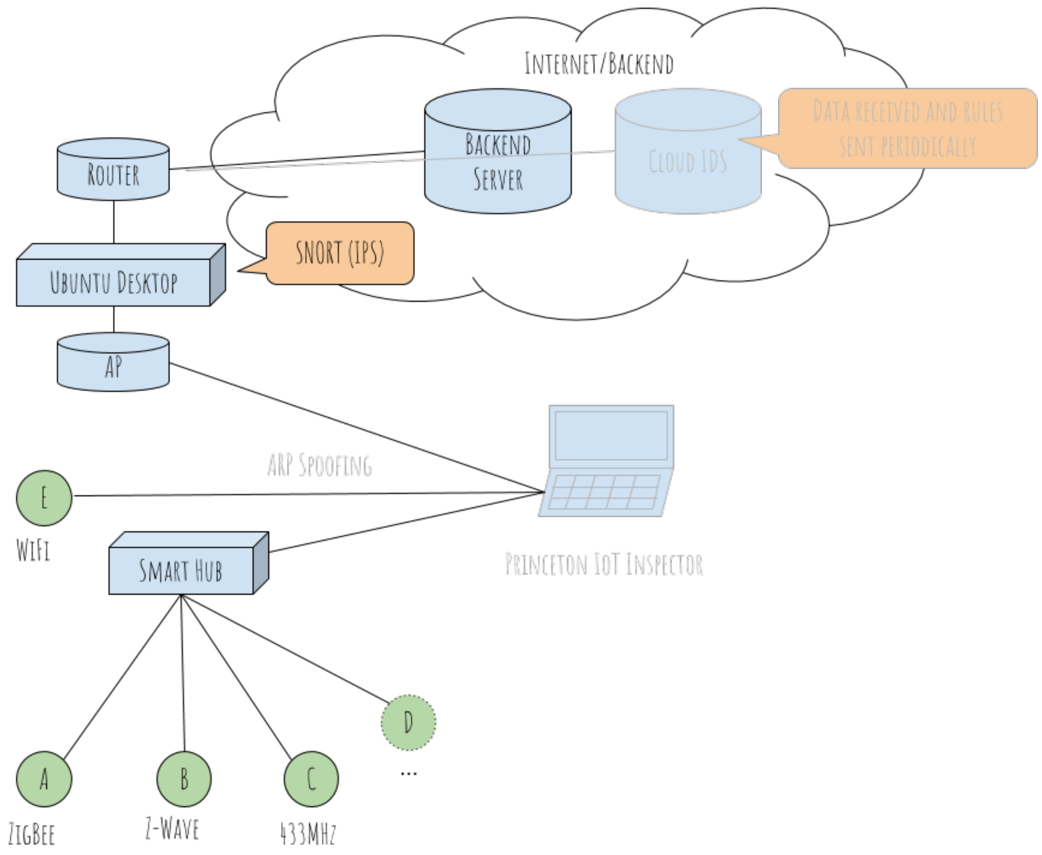


Figure 2: An overview of the second experiment. Non-essential equipment in regards to the second experiment have been greyed out in the figure.

However, as we decided to run Snort in IPS mode, and not as a simple packet sniffer, we somehow needed to separate alerts from logs in order to be able to reason about how well the detection features of Snort works in regards to privacy and IoT. There are different ways to approach this, but we decided to mark all packets as alerts but attach a label, "LOG-MSG", in the message field of each packet that was not an alert generated by Snort itself to distinguish them. Although this might not be the intended way to log traffic in Snort, we believe this did not affect the performance of the IPS given the order the different actions are processed (see subsection 5.4.2 for more details on the ordering of actions in Snort). Moreover, this solution also made it easy to forward all data to Kibana for visualisation and to tell them apart in that interface. All alerts with the message content of "LOG-MSG" will from now on be referred to as *logs* in the rest of the thesis.

As mentioned, in order to visualise the logs and alerts generated by Snort, Kibana was used. This meant that the output from Snort somehow needed to be parsed in such a way that Elasticsearch would be able to index the logs correctly for Kibana to display. Indexing the files in an efficient manner is also important for performance given the share amount of packets. By doing so, searches and general lookup are performed much quicker. The software used for parsing the Snort output in a format that Elasticsearch could process was Logstash. For a graphical representation of how these different software were set up, see Figure 3.

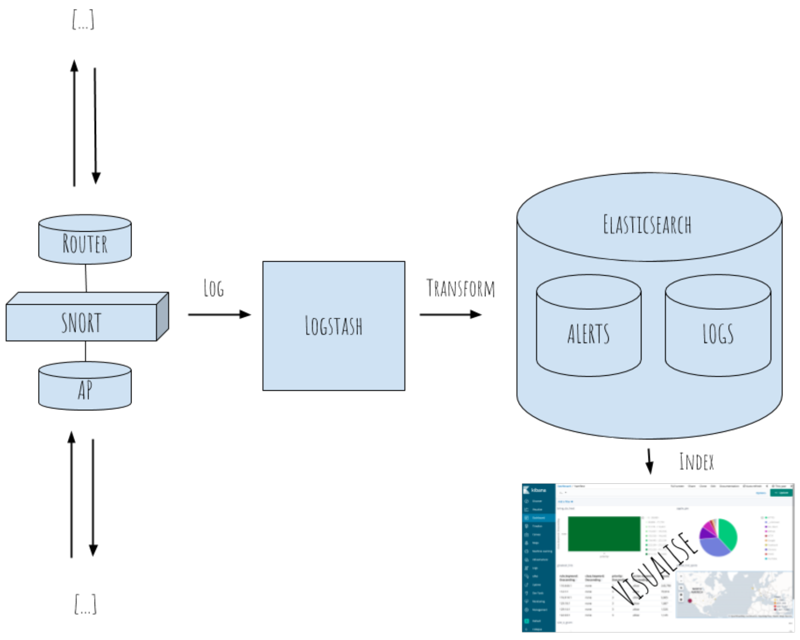


Figure 3: The pipeline that was used in the second experiment.

Given that the logs from Snort were outputted in JSON format, not much configuration had to be done in Logstash regarding parsing the original content of the alerts and logs. Instead, the majority of time spent on configuring Logstash was invested into finding useful plugins. There exists a vast diversity of plugins for Logstash, but we decided to primarily focus on the DNS [56] and Geoip [57] filter. The former adds the functionality to do a reverse IP lookup, which proved to be useful in order to better visualize the endpoints the different devices contacted. The latter, the Geoip filter, instead translates IP to geographical location related data, which means that the IP address can be associated with, for instance, a specific region like the EU, Asia and so on. It is also able to provide more specific location related details such as which country and city that IP address is associated with. Moreover, Geoip also stores the translation as geographic coordinates (longitude and latitude) which can be plotted in Kibana on a world map in order to draw conclusions on the flow of traffic from a regional perspective.

6.3 Experiment III - Cloud Rules

Since the third experiment was a proof of concept, it was not exposed to the same restrictions as the other two experiments. In the other two experiments, traffic was analysed during the same time span and with the same traffic in order to be able to fairly compare the two solutions to one another. In the third experiment, however, no such considerations had to be made. Instead, the data was sent in the later stages of the experiments (when we had a better idea of the size of the logs), but also restricted in terms of how many log files we chose to upload, which meant that the third experiment did not run in complete parallel with the other two. See Figure 4 for an overview of which parts of the testbed were affected by the third experiment.

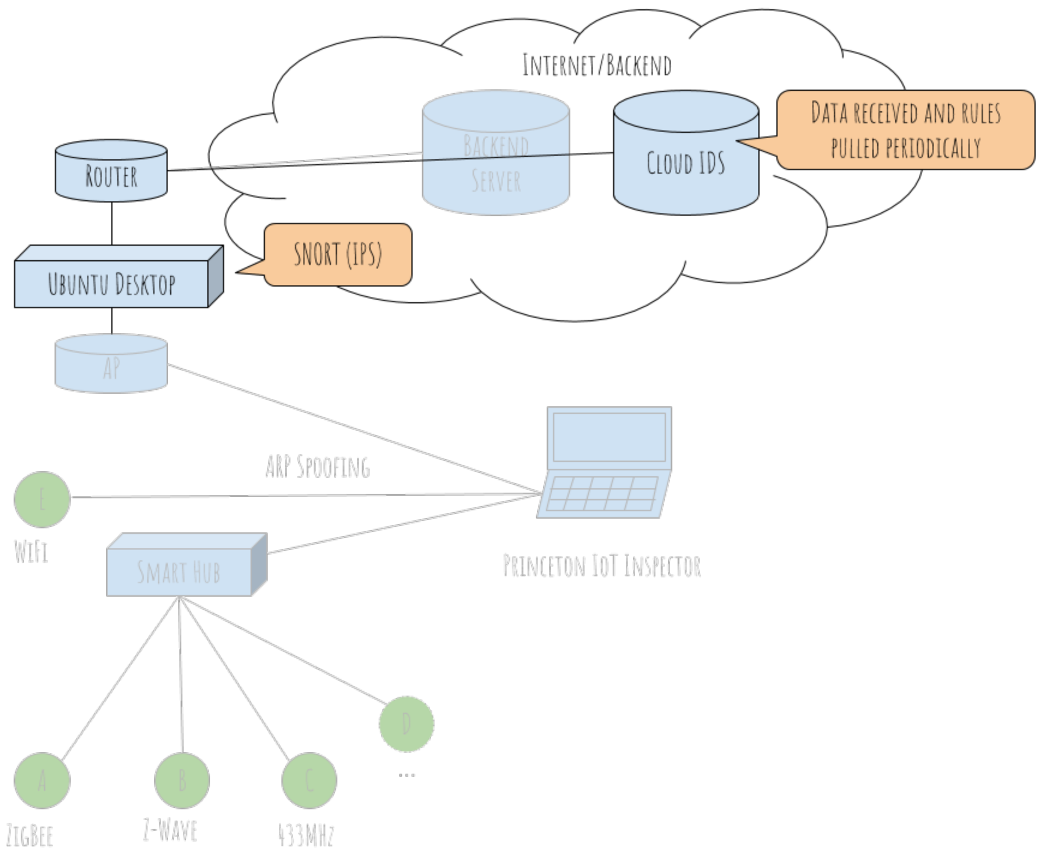


Figure 4: An overview of the third experiment. Non-essential equipment in regards to the third experiment have been greyed out in the figure.

The idea behind this experiment, as mentioned in the previous chapter, was to simulate a user subscribing to a third party service which they supply with data and from which they pull updated rules. Ideally, this would be done on a regular interval (which probably would be the case for a commercial solution), instead this was done in the later stages of the experiments when we had a better idea of the size of the data.

The first part of the experiment was to construct a parser that could parse the Snort logs/alerts and extract only the metadata from the packets. This was done in order to reduce the size of the data sent to the cloud. The parser was constructed in Python, and it took a Snort log file as input, filtered out the payload portion of the log, and piped the remaining parts of the log (the metadata) to a separate file. Since we had defined a 10 Megabytes limit on the log files in Snort in order to keep the log file size manageable, and Snort always processes the current packets in a file called alerts_json.txt (already finished logs had the format of alert_json.txt.xxxxxxxxxx), it was easy to distinguish a file that was being processed from one that had been completed. Nevertheless, since the parser did not run continuously (not essential and thus not a good use of system resources) it needed to keep track of its state (which logs it had processed already and which it had not). This was done by writing the state information (name of the latest processed log file and the number of last parsed log file) in a separate .txt file right before the parser finished its execution.

The second part was to create yet another Python script that was responsible for shipping the parsed logs to the AWS server at regular intervals. This script took all the available parsed logs and sent them via a HTTP POST request to the web server located on AWS. Before deleting the parsed log files, the shipper checked to make sure that the HTTP response matched a success response. For the actual code of the two Python scripts, see Appendix C.

The next part was to create cron jobs to automate the two scripts and managing the restarting of Snort. Since there is no way to update rules dynamically in Snort, it needed to be restarted in order to include the new rules. Thus, the two Python scripts were automated to run some time before the restart of Snort. All of the cron jobs were set to run during hours that would have less of an impact on a typical user.

The Snort rules developed on the cloud were written manually and only served as mockup rules, i.e. they had no other purpose than to confirm that the concept worked as expected. These rules were fetched automatically in the same fashion, using a cron job. They included a timestamp of the date and time of when the rules were downloaded in order to more easily be able to detect if there were any issues with the automation process.

Finally, the cloud rules were tested and confirmed working as expected by manual observation in Kibana.

CHAPTER 7

Results

This chapter presents the findings from the experiments described in previous chapters. It does so for both the different device segments and the cloud solution. For the device segments, the results are based on the observation of the threats found in the threat model.

The chapter begins with three sections dedicated to the results from the different device segments. The first deals with the threats related to concentration of geographical endpoints and cloud providers as well as third-party tracking. The second section primarily focuses on the threats related to the shape of the traffic. The third digs into the aspect of encryption, or lack thereof, as well as generally unexpected device behaviour detected during the experiments.

The final section of the chapter presents the results from the cloud solution. It describes how it could be confirmed working as expected as well show some observations and metrics that are important for evaluating the solution's viability.

7.1 Endpoints

This section will present the results from the device segments in regards to the threats in the threat model that relates to endpoints. That is: concentration of geographical endpoints and cloud providers but also third party tracking.

It contains two subsections where the first presents the results from the well-known and less-known brands segments. The latter instead focuses on these threats in relation to the regional segments, that is the EU, North American and Asian segments.

7.1.1 Brand Comparison

Looking at the traffic from a geographical perspective, most of the traffic from the well-known brands segment terminated in either the UK or the US as can be seen in Figure 1. In the Figure, red colour coding indicates that a significant share of the traffic terminated in that country. Correspondingly, orange indicates relatively high and yellow comparatively low termination rates.

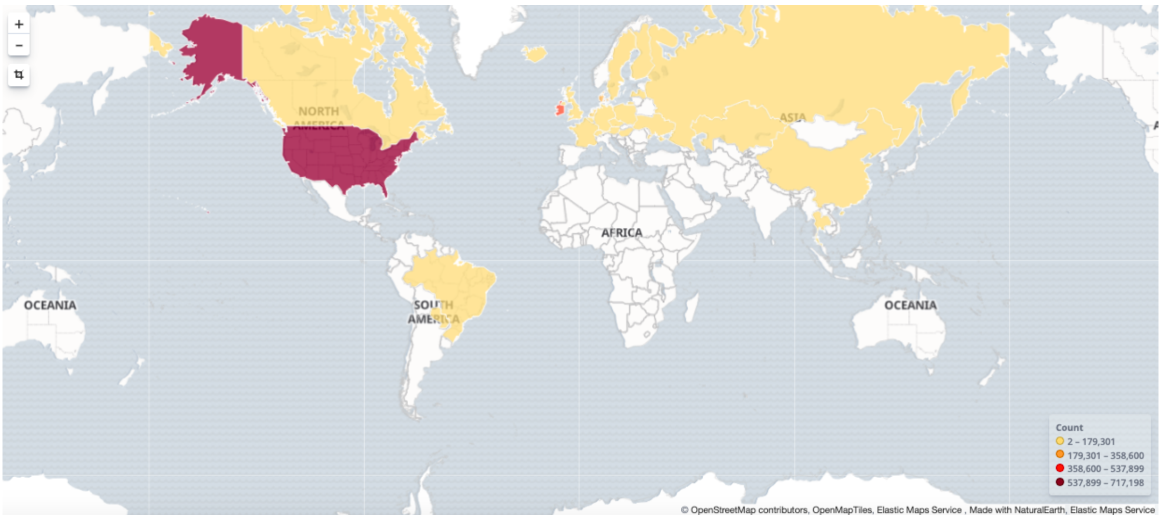
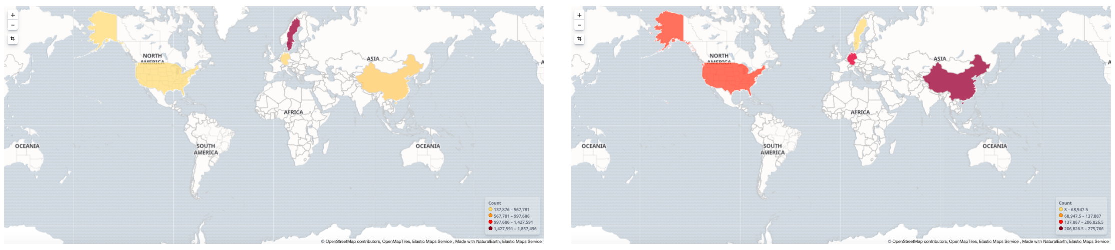


Figure 1: The figure shows where in the world the traffic from the well-known brands segment terminated.

Worth mentioning, regarding the traffic from the devices in the less-known brands segment, much of it was sent to a network-attached storage (NAS) unit owned by one of the authors of the thesis. This happened despite the fact that the NAS resided on a network separated from the testbed. This behaviour was not exclusive to the devices in the less-known brands segment but also occurred for some of the other segments, although to a lesser extent. One possible explanation to how the devices knew about the existence of the NAS in the first place could be related to application privileges, i.e. that they got the information from one of the Android phones used during the experiments. For this reason, there are two different maps (see Figure 2) for depicting how the traffic from the devices from the less-known brands segment terminated in regards to geographical locations. Nevertheless, from now on, all maps, graphs and tables in this chapter will include the traffic to the NAS unless stated otherwise, as this was what was observed.

The traffic from the less-known brand segment mainly terminated in either the EU, China or the US. This can be observed in Figure 2.



(a) Including traffic to the NAS. **(b)** Excluding traffic to the NAS.

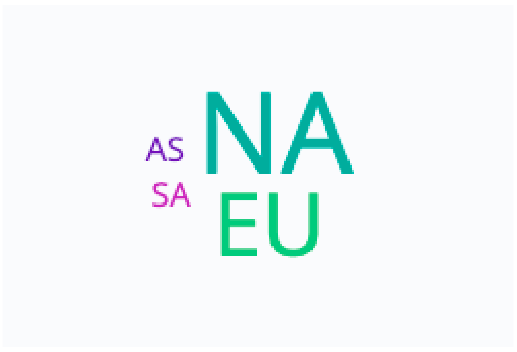
Figure 2: The figure shows where in the world the traffic from the less-known brands segment terminated.

Based on the knowledge of in which parts of the world the traffic from the different segments terminated, it was relevant to take a closer look into which specific regions and cities the traffic terminated in. Figure 3 shows this.

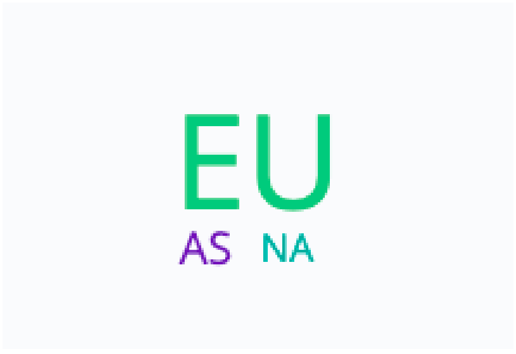
Worth noting, the Geoip filter in Logstash was not always able to translate the coordinates to all levels of details, as seen in Figure 3. For instance, by looking at the traffic from the less-known brands segment, much traffic terminated in Hesse (in line with the observations from the world map). However, looking at the same traffic from a city perspective, the traffic to German cities is too low. This probably indicates that the filter is able to resolve the coordinates on a regional level but not on a city level. Despite these limitations in the filter, this geographical representation still gives a relatively clear view of how the two segments differ in regards to geographical endpoints.

Taking a closer look at Figure 3, one can observe that Mountain View and Dublin stands out as the termination points for the devices in the well-known brand segment. The global headquarters of Google is located in Mountain View [58] and their European headquarter is in Ireland [59]. Given that multiple Google devices were represented in this segment, it might explain why so much traffic terminated in these specific regions.

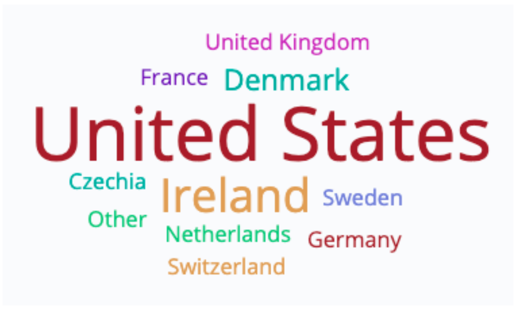
For the less-known brands segment, when excluding the traffic to the NAS, one can see that a significant potion of the traffic terminated in China, both in Hangzhou and to a lesser extent in Shenzhen. Although a device might be aimed for a certain market, many are produced in China. However, that does not necessarily mean that the vendor relies on back-end services in China. Nevertheless, for devices in this segment, this seemed to be the case to a large degree.



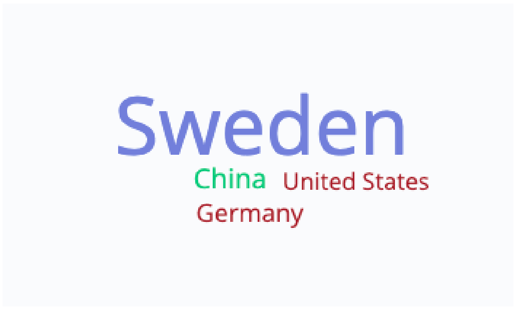
(a) S1: Destination Continents.



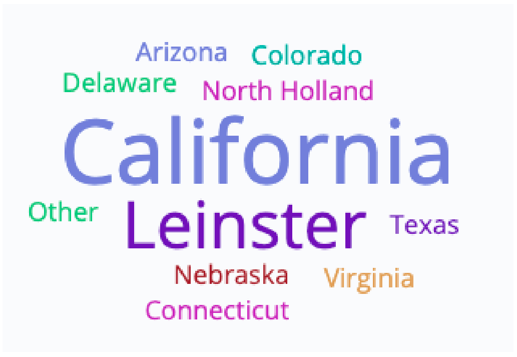
(b) S2: Destination Continents.



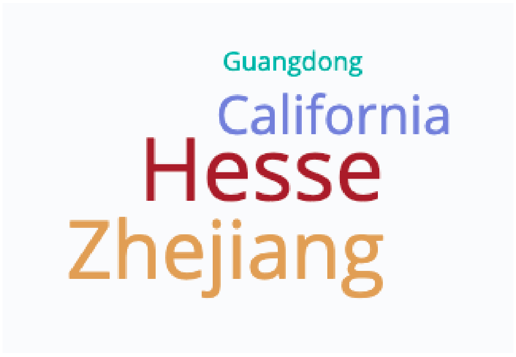
(c) S1: Destination Countries.



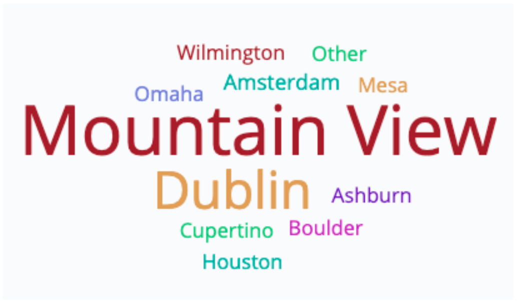
(d) S2: Destination Countries.



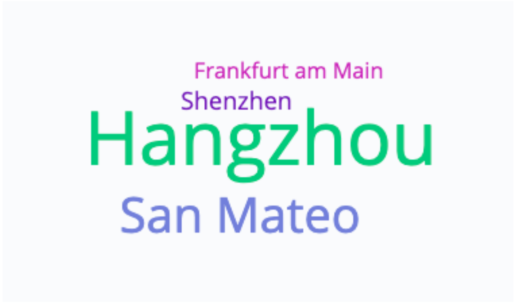
(e) S1: Destination Regions.



(f) S2: Destination Regions.



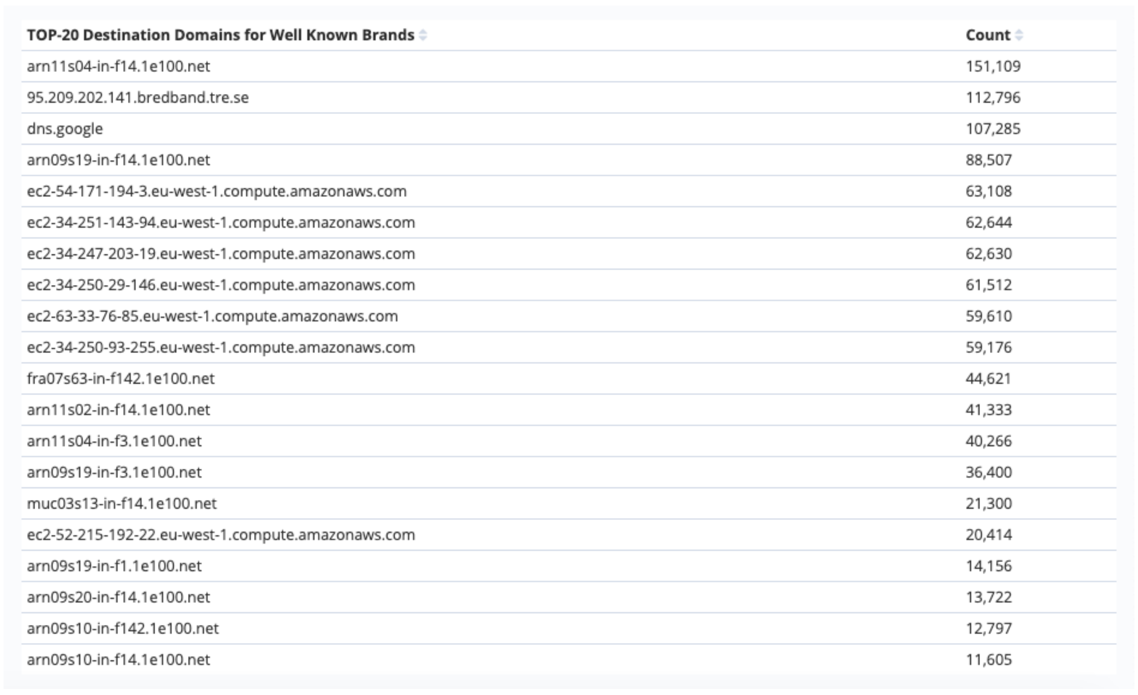
(g) S1: Destination Cities.



(h) S2: Destination Cities.

Figure 3: Different level of details regarding where the traffic terminated, ranging from continent level down to city level. S1 corresponds to the well-known and S2 to the less-known brands segment.

When analysing the threat related to the concentration of cloud providers, it was necessary to take a closer look at the specific domains that the devices contacted. While word clouds give a good bird's eye on the traffic, Figure 4 and 5 list the top 20 domains for the two segments as well as show how many packets were sent to each of those endpoints. Worth noting though, all of these graphs and tables says little to nothing about the volume of the traffic, only the frequency. For a thorough look on the actual shape of the traffic and the amount of data generated by the different devices and segments, see Section 7.2.



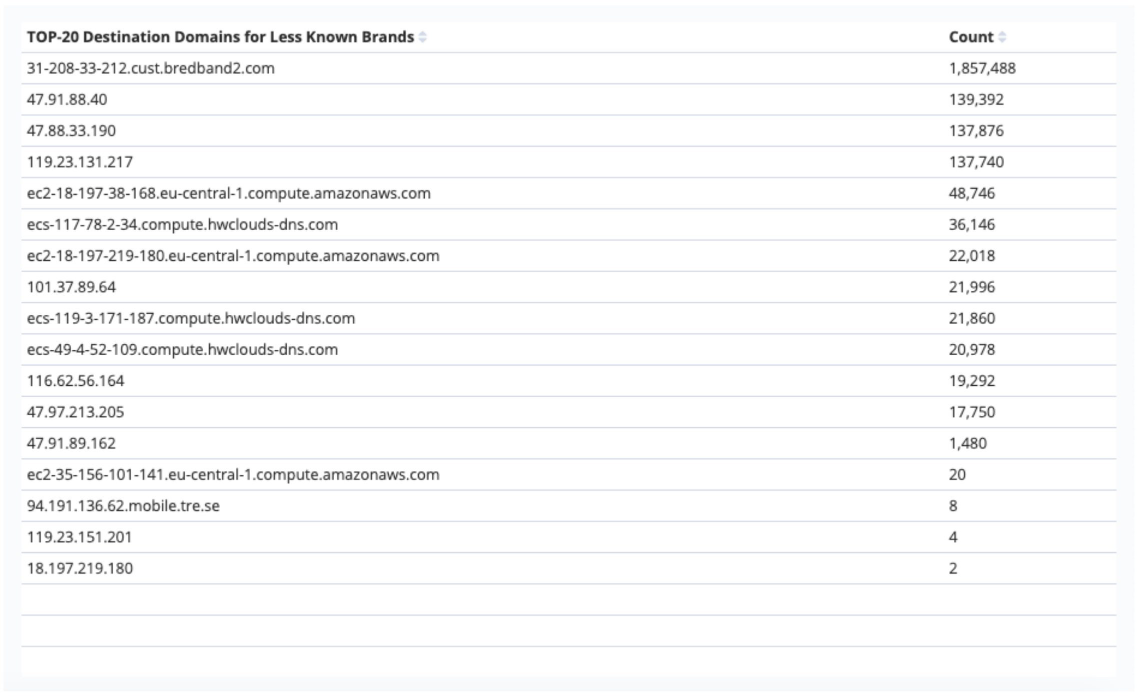
TOP-20 Destination Domains for Well Known Brands	Count
arn11s04-in-f14.1e100.net	151,109
95.209.202.141.bredband.tre.se	112,796
dns.google	107,285
arn09s19-in-f14.1e100.net	88,507
ec2-54-171-194-3.eu-west-1.compute.amazonaws.com	63,108
ec2-34-251-143-94.eu-west-1.compute.amazonaws.com	62,644
ec2-34-247-203-19.eu-west-1.compute.amazonaws.com	62,630
ec2-34-250-29-146.eu-west-1.compute.amazonaws.com	61,512
ec2-63-33-76-85.eu-west-1.compute.amazonaws.com	59,610
ec2-34-250-93-255.eu-west-1.compute.amazonaws.com	59,176
fra07s63-in-f142.1e100.net	44,621
arn11s02-in-f14.1e100.net	41,333
arn11s04-in-f3.1e100.net	40,266
arn09s19-in-f3.1e100.net	36,400
muc03s13-in-f14.1e100.net	21,300
ec2-52-215-192-22.eu-west-1.compute.amazonaws.com	20,414
arn09s19-in-f1.1e100.net	14,156
arn09s20-in-f14.1e100.net	13,722
arn09s10-in-f142.1e100.net	12,797
arn09s10-in-f14.1e100.net	11,605

Figure 4: The top 20 destination domains for the devices in the well-known brands segment.

From Figure 4, showing the top domains of the devices in the well-known brands segment, a few things are apparent. Most of the domains contain either .1e100.net, owned by Google [60], or .amazonaws.com, owned by Amazon, where the prefix right before .amazonaws.com reveals to which region that endpoint belongs [61]. These two companies are unsurprisingly dominant amongst the branded products, although, interestingly enough, no Amazon products were used in that segment.

Regarding the less-known brands segment (see Figure 5), we see a different picture to a large extent. First and foremost, the cust.bredband2.com domain points to the NAS mentioned earlier and thus explains the vast amount of packets to that domain from this segment. Moreover, although Amazon can be found amongst the destination domains for this segment as well, many of the devices from this segment contact domains most likely less familiar to a typical user. For instance, 47.91.88.40 as well as 47.91.89.162 belongs to Alibaba.com, according to whatismyip.com [62]. Similarly, 47.88.33.190, 119.23.131.217, 101.37.89.64, 116.62.56.164, 47.97.213.205 and

119.23.151.201 belongs to Aliyun Computing Co. Ltd [62], also known as Alibaba Cloud [63]. Others, like hwclouds-dns.com, are DNS servers owned by Huawei [64], another big Chinese technology company. Although many of these companies are Chinese, the location of the servers they use might vary. Simply put, that means that looking at the geographical locations of the endpoints alone might not reveal who the underlying cloud providers is and vice versa.



TOP-20 Destination Domains for Less Known Brands	Count
31-208-33-212.cust.bredband2.com	1,857,488
47.91.88.40	139,392
47.88.33.190	137,876
119.23.131.217	137,740
ec2-18-197-38-168.eu-central-1.compute.amazonaws.com	48,746
ecs-117-78-2-34.compute.hwclouds-dns.com	36,146
ec2-18-197-219-180.eu-central-1.compute.amazonaws.com	22,018
101.37.89.64	21,996
ecs-119-3-171-187.compute.hwclouds-dns.com	21,860
ecs-49-4-52-109.compute.hwclouds-dns.com	20,978
116.62.56.164	19,292
47.97.213.205	17,750
47.91.89.162	1,480
ec2-35-156-101-141.eu-central-1.compute.amazonaws.com	20
94.191.136.62.mobile.tre.se	8
119.23.151.201	4
18.197.219.180	2

Figure 5: The top 20 destination domains for the devices in the less-known brands segment.

For the curious readers, a more detailed description of which device contacted which endpoints the most can be found in Appendix F. Moreover, for details about which IP address is associated with what device, see Appendix E.

Finally, moving on to the last threat in the threat model regarding endpoints, we take a closer look into reported tracker and/or advertisement domain contacted by the different devices. Princeton IoT Inspector has a built in tool for visualising this. As seen in Figure 6, some of the devices from the well-known brands segment do contact known ads or tracker domains, more specifically the Google Home Mini and the Google Chromecast (192.168.0.107 and 192.168.0.109 respectively). Somewhat surprisingly, no such behaviour was reported by Princeton IoT Inspector regarding devices from the less-known brands segment.

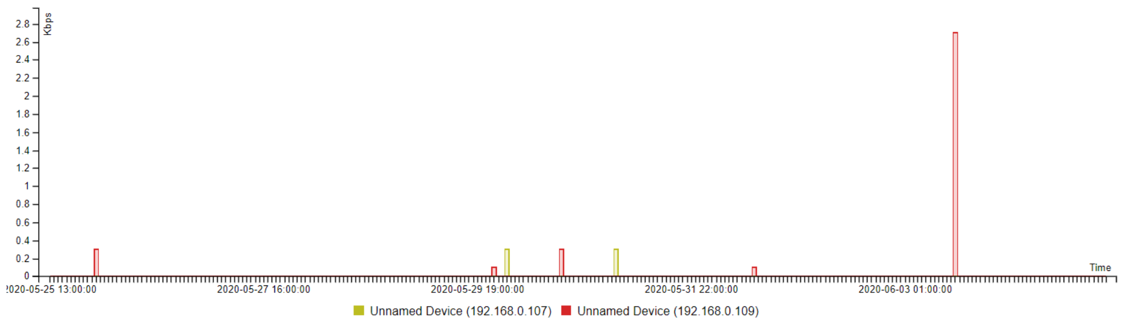
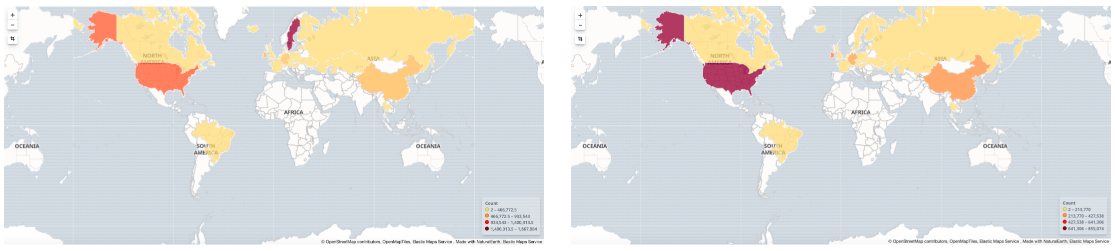


Figure 6: Traffic from the well-known brands segment that Princeton IoT Inspector recognised as being associated with advertisements servers and/or trackers.

7.1.2 Region Comparison

From a geographical perspective, much of the traffic from the EU segment terminated in either the US, the UK, the EU or China, as seen in the previous section. However, some traffic also terminated in other parts of the world such as in Canada, Russia or South America, although to a much lower extent. This can be seen in Figure 7, where traffic from the EU segment, both including and excluding the NAS, is presented.



(a) Including traffic to the NAS. **(b)** Excluding traffic to the NAS.

Figure 7: The figure shows where in the world the traffic from the EU segment terminated.

In contrast, the devices from the North American segment terminated predominantly in the US as well to some extent in the EU. Moreover, unlike the traffic from the devices from the EU segment, the endpoints were highly concentrated, i.e. not much traffic terminated in any other part of the world. See Figure 8.

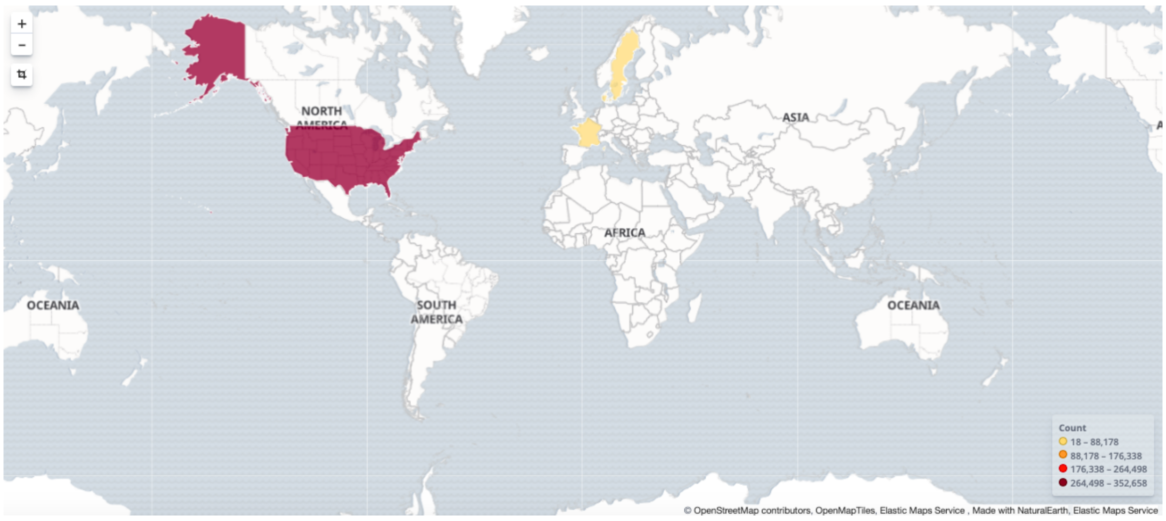


Figure 8: The figure shows where in the world the traffic from the North American segment terminated.

Similar to the traffic from the devices in the EU segment, the traffic from the devices in the Asian segment terminated primarily in China, the EU and the US. Furthermore, some traffic also terminated in countries such as Russia and Canada as can be seen in Figure 9.

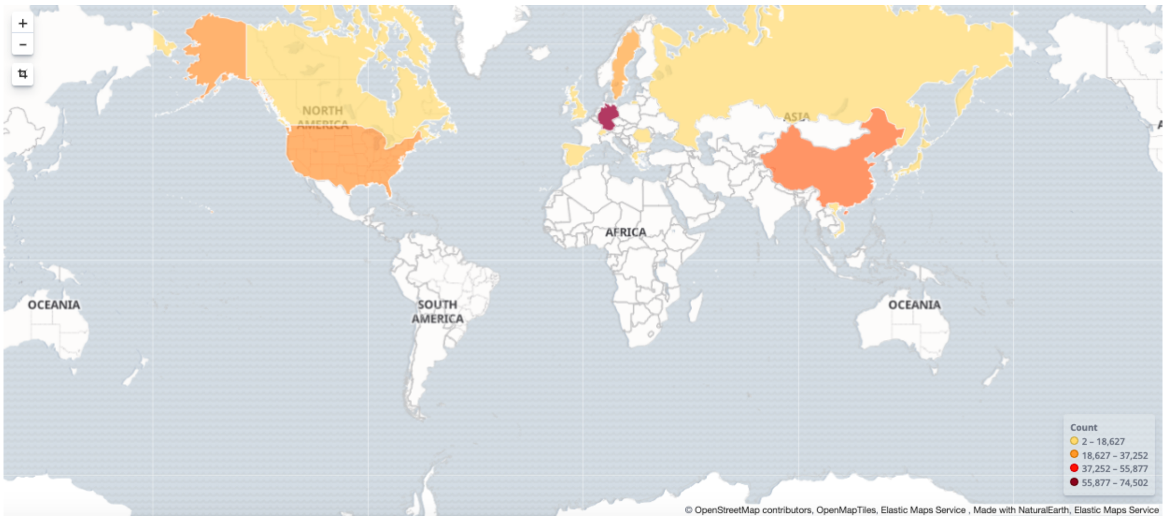
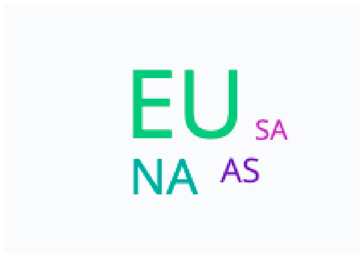
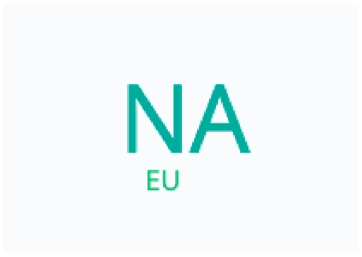

NORTH AMERICA
ASIA
AFRICA
OCEANIA
SOUTH AMERICA
OCEANIA
Count
2 – 18,627
18,627 – 37,252
37,252 – 55,877
55,877 – 74,502
© OpenStreetMap contributors, OpenMapTiles, Elastic Maps Service , Made with NaturalEarth, Elastic Maps Service

Figure 9: The figure shows where in the world the traffic from the Asian segment terminated.

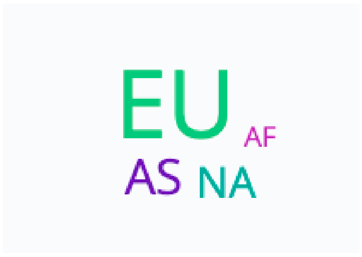
Figure 10 gives a complementary and more detailed look into which specific parts of the world the traffic from the different regional segments terminated in. Although it gives a good bird's view on the traffic, this representation sufferers from the same drawbacks regarding the Geoip translation as mentioned in the previous section.



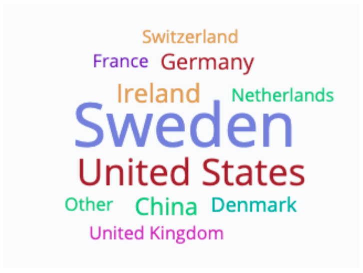
(a) S3: Destination Continents.



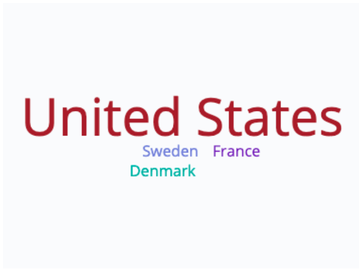
(b) S4: Destination Continents.



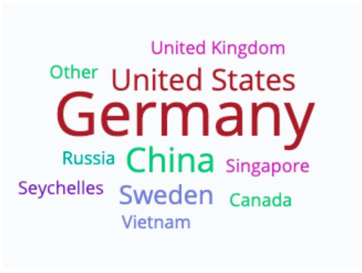
(c) S5: Destination Continents.



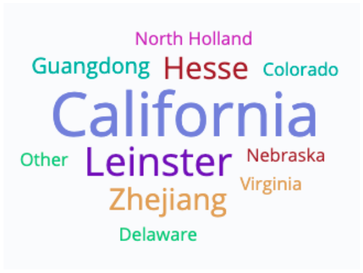
(d) S3: Destination Countries.



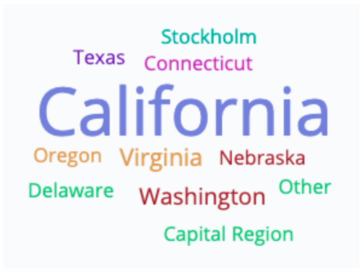
(e) S4: Destination Countries.



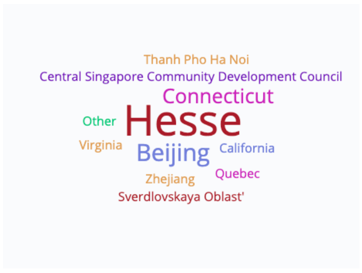
(f) S5: Destination Countries.



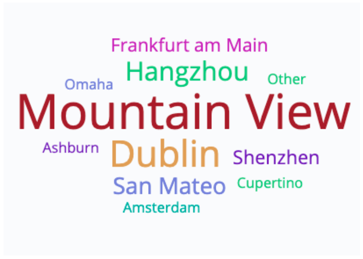
(g) S3: Destination Regions.



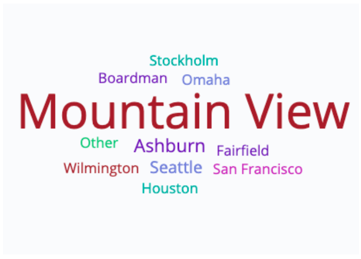
(h) S4: Destination Regions.



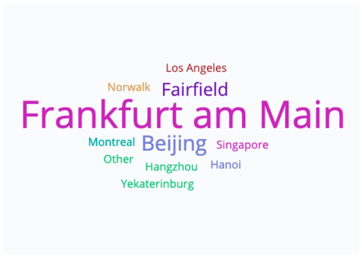
(i) S5: Destination Regions.



(j) S3: Destination Cities.



(k) S4: Destination Cities.



(l) S5: Destination Cities.

Figure 10: Different level of details regarding where the traffic terminated, ranging from continent down to city level. S3 corresponds to the EU segment, S4 the North American segment and S5 the Asian segment.

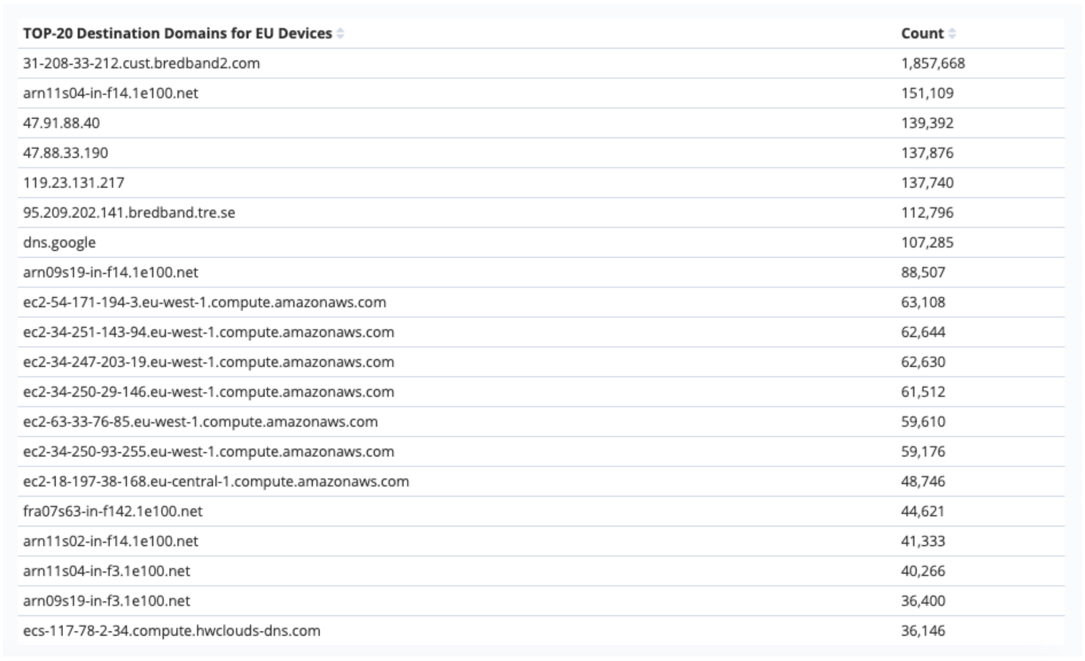
The geographical location of the endpoints from the devices in the EU segment have already been presented in detail in the previous section and will thus not be discussed in this section. Instead, looking at the traffic from the devices in the North American segment, it primarily terminated in the US and dominantly so in Mountain View in California. Similar to the well-known brands segment, multiple Google devices were present in North American segment, which probably explains why so much traffic terminated in this region.

Looking at the traffic from the devices in the Asian segment, most of it terminated in Germany, China and the US. More specifically, much traffic terminated in Frankfurt am Main in Hesse but also Beijing and multiple other less frequent endpoints in China and the US.

Transitioning to the threat related to the concentration of cloud providers, Figure 11 shows the top 20 domains for the EU segment. Corresponding figures for the North American and Asian segments are shown in Figure 12 and 13. For details on the top 10 endpoints for each individual device from the different segments, see Appendix F.

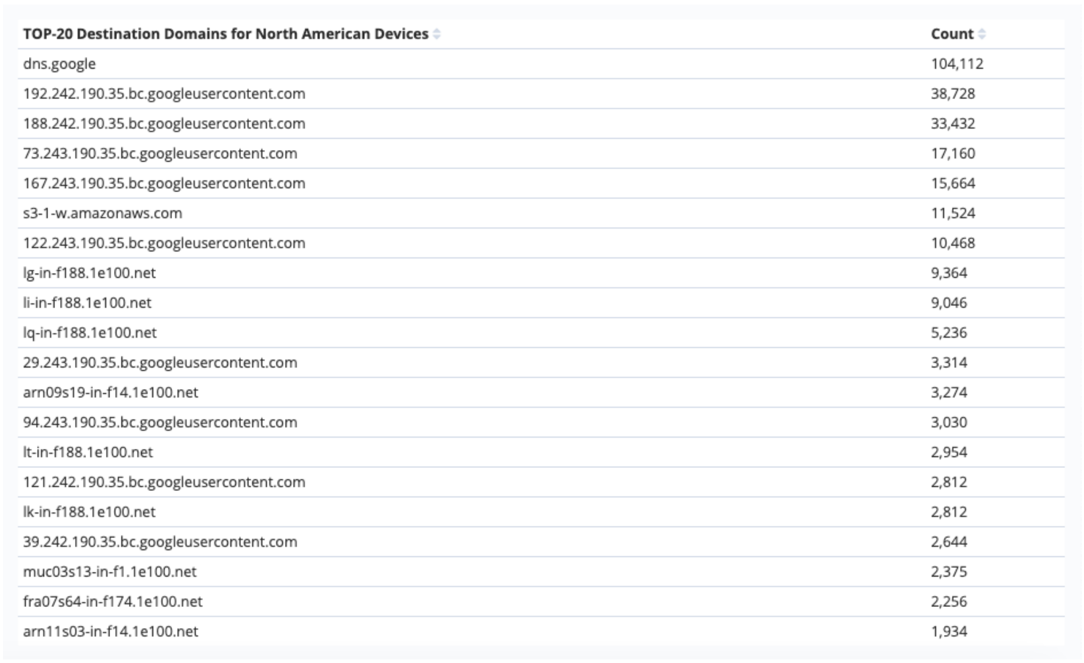
The top destination domains from the EU segment have already been discussed and for details about these, see Section 7.1.1. Regarding the North American segment, all but one of the top 20 domains are google domains such as 8.8.8.8 (dns.google), bc.googleusercontent.com or .1e100.net. The remaining domain is instead owned by Amazon (.amazonaws.com). This was not that surprising, as the North American segment consisted exclusively of Google and Amazon devices.

For the devices in the Asian segment we also see a concentration of cloud providers, although not of the same companies. For this segment, most traffic was directed



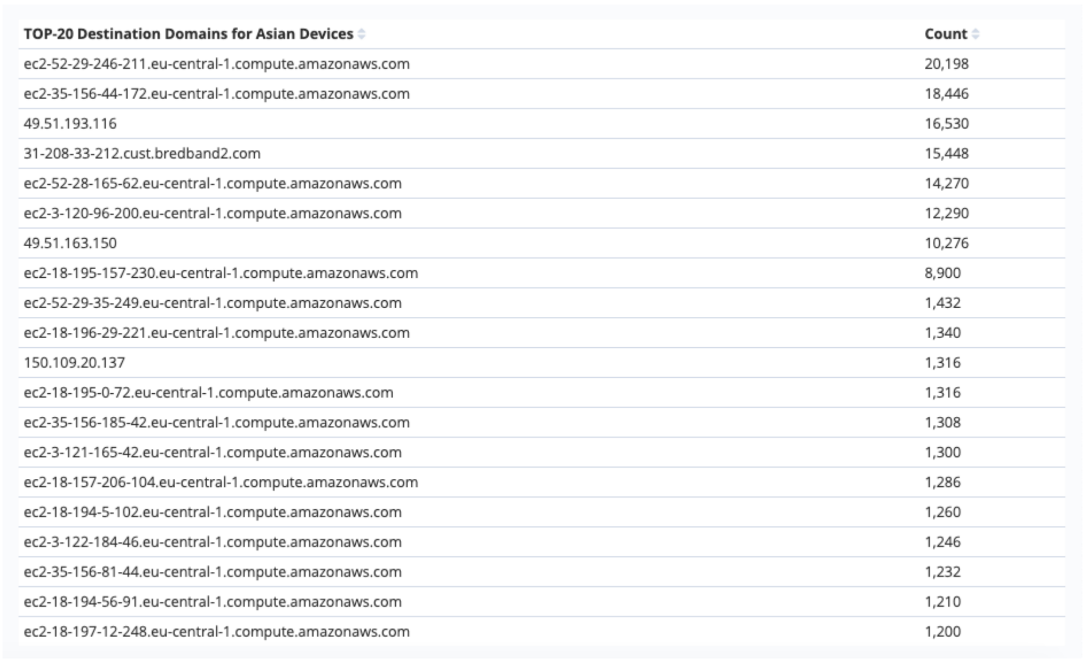
TOP-20 Destination Domains for EU Devices	Count
31-208-33-212.cust.bredband2.com	1,857,668
arn11s04-in-f14.1e100.net	151,109
47.91.88.40	139,392
47.88.33.190	137,876
119.23.131.217	137,740
95.209.202.141.bredband.tre.se	112,796
dns.google	107,285
arn09s19-in-f14.1e100.net	88,507
ec2-54-171-194-3.eu-west-1.compute.amazonaws.com	63,108
ec2-34-251-143-94.eu-west-1.compute.amazonaws.com	62,644
ec2-34-247-203-19.eu-west-1.compute.amazonaws.com	62,630
ec2-34-250-29-146.eu-west-1.compute.amazonaws.com	61,512
ec2-63-33-76-85.eu-west-1.compute.amazonaws.com	59,610
ec2-34-250-93-255.eu-west-1.compute.amazonaws.com	59,176
ec2-18-197-38-168.eu-central-1.compute.amazonaws.com	48,746
fra07s63-in-f142.1e100.net	44,621
arn11s02-in-f14.1e100.net	41,333
arn11s04-in-f3.1e100.net	40,266
arn09s19-in-f3.1e100.net	36,400
ecs-117-78-2-34.compute.hwclouds-dns.com	36,146

Figure 11: The top 20 destination domains for the devices in the EU segment.



TOP-20 Destination Domains for North American Devices	Count
dns.google	104,112
192.242.190.35.bc.googleusercontent.com	38,728
188.242.190.35.bc.googleusercontent.com	33,432
73.243.190.35.bc.googleusercontent.com	17,160
167.243.190.35.bc.googleusercontent.com	15,664
s3-1-w.amazonaws.com	11,524
122.243.190.35.bc.googleusercontent.com	10,468
lg-in-f188.1e100.net	9,364
li-in-f188.1e100.net	9,046
lq-in-f188.1e100.net	5,236
29.243.190.35.bc.googleusercontent.com	3,314
arn09s19-in-f14.1e100.net	3,274
94.243.190.35.bc.googleusercontent.com	3,030
lt-in-f188.1e100.net	2,954
121.242.190.35.bc.googleusercontent.com	2,812
lk-in-f188.1e100.net	2,812
39.242.190.35.bc.googleusercontent.com	2,644
muc03s13-in-f1.1e100.net	2,375
fra07s64-in-f174.1e100.net	2,256
arn11s03-in-f14.1e100.net	1,934

Figure 12: The top 20 destination domains for the devices in the North American segment.



TOP-20 Destination Domains for Asian Devices	Count
ec2-52-29-246-211.eu-central-1.compute.amazonaws.com	20,198
ec2-35-156-44-172.eu-central-1.compute.amazonaws.com	18,446
49.51.193.116	16,530
31-208-33-212.cust.bredband2.com	15,448
ec2-52-28-165-62.eu-central-1.compute.amazonaws.com	14,270
ec2-3-120-96-200.eu-central-1.compute.amazonaws.com	12,290
49.51.163.150	10,276
ec2-18-195-157-230.eu-central-1.compute.amazonaws.com	8,900
ec2-52-29-35-249.eu-central-1.compute.amazonaws.com	1,432
ec2-18-196-29-221.eu-central-1.compute.amazonaws.com	1,340
150.109.20.137	1,316
ec2-18-195-0-72.eu-central-1.compute.amazonaws.com	1,316
ec2-35-156-185-42.eu-central-1.compute.amazonaws.com	1,308
ec2-3-121-165-42.eu-central-1.compute.amazonaws.com	1,300
ec2-18-157-206-104.eu-central-1.compute.amazonaws.com	1,286
ec2-18-194-5-102.eu-central-1.compute.amazonaws.com	1,260
ec2-3-122-184-46.eu-central-1.compute.amazonaws.com	1,246
ec2-35-156-81-44.eu-central-1.compute.amazonaws.com	1,232
ec2-18-194-56-91.eu-central-1.compute.amazonaws.com	1,210
ec2-18-197-12-248.eu-central-1.compute.amazonaws.com	1,200

Figure 13: The top 20 destination domains for the devices in the Asian segment.

towards domains owned by either Amazon (compute.amazonaws.com) or domains that WhatIsMyIP.com recognized as belonging to Tencent Cloud Computing Co. Ltd. [62]. Similar to the traffic from the devices in the less-known brands segment, one can see that the NAS mentioned in the previous section is also present in the top 20 domains for the devices in the Asian segment (cust.bredband2.com).

Finally, moving on to the threat related to third-party tracking, Princeton IoT Inspector only recognised Google devices from the North American segment as contacting known trackers and/or advertisement servers from the North American and Asian segments, as can be seen in Figure 14. This might indicate that other devices indeed do not contact trackers or ad servers but might just as well mean that Google is the only manufacturer in the testbed whose devices contacts trackers and/or ad domains known to Princeton IoT Inspector.

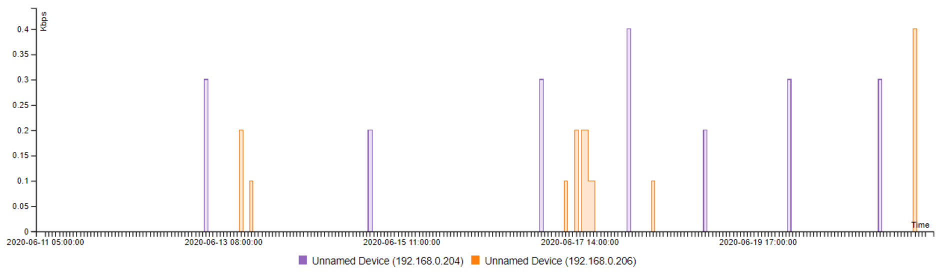


Figure 14: Traffic from devices in the North American segment that Princeton IoT Inspector recognised as being associated with ad servers and/or trackers.

7.2 Shape of the Traffic

This section will present the results from the different device segments in regards to the threat in the threat model that relates to the shape of traffic. It means that it will focus on characteristics of the traffic that might reveal what type of device the traffic originated from. Examples of properties that are of interest are: the amount of traffic sent, with what frequency and using which protocol. Similar to the previous section, the focus in this section is on outgoing traffic, although incoming traffic will briefly be presented in the end of the section as well.

The first subsection presents the results from the well-known and less-known brands segments. This is followed by a subsection dedicated to the results from the three regional segments, meaning the EU, the North American and the Asian segments.

Both subsections use a mix of device names, product categories and IP addresses when referring to the results from the different devices. To find out which device is associated with what IP address, see Appendix E.

7.2.1 Brand Comparison

The device that sent the most data of all devices in the EU segment was the IP camera from the less-known brands segment (192.168.0.103), which can be seen in Figure 15.

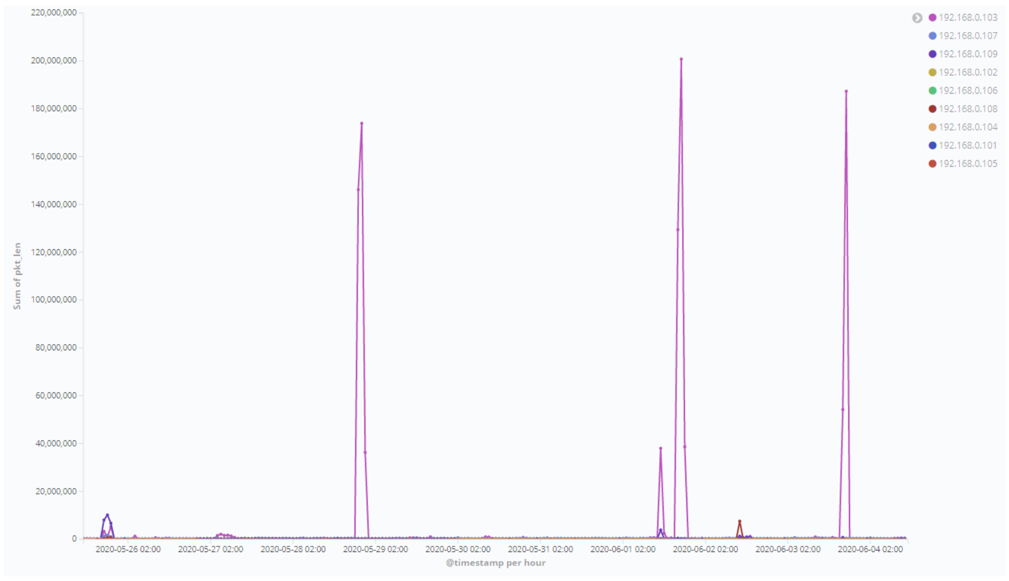


Figure 15: The sum of the packet lengths (i.e. total bytes) per hour for all the devices in the EU segment. The purple line in the figure corresponds to the IP camera from the less-known brands segment.

Looking specifically at the devices from the well-known brands segment, two devices were more active than the others. One of these was the Google Chromecast (192.168.0.109), which shows a spike of around 26 kB, while the other one, the Google Home Mini (192.168.0.107), consistently sent traffic of around 4 kB, as can be seen in Figure 16.

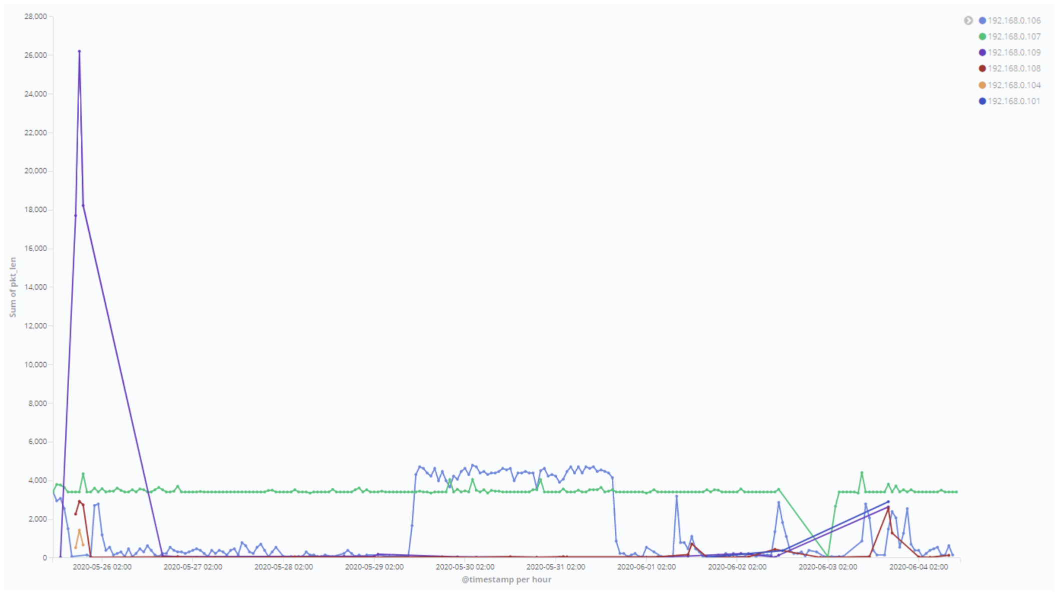


Figure 16: The sum of packet lengths (i.e. total bytes) per hour for all the devices in the well-known brands segment. The purple line corresponds to the Google Chromecast (192.168.0.109) and the green line the Google Home Mini (192.168.0.107).

By looking at the size of the data sent from both segments, the total data sent favours the IP camera from the less-known brands segment (192.168.0.103) which sent around 1 GB of data. Out of the devices from well-known brands segment, the devices that sent the most data was the Google Home Mini (192.168.0.107) that sent 83.2 MB, followed by the Google Chromecast (192.168.0.109) that sent around 48.9 MB. Details on this can be found in Appendix G.

Comparing the two segments in regards to device categories, the two cameras (192.168.0.101 and 192.168.0.103) both communicated with servers outside the network as can be seen in Figure 17, although the camera from the less-known brands segment sent noticeably more.

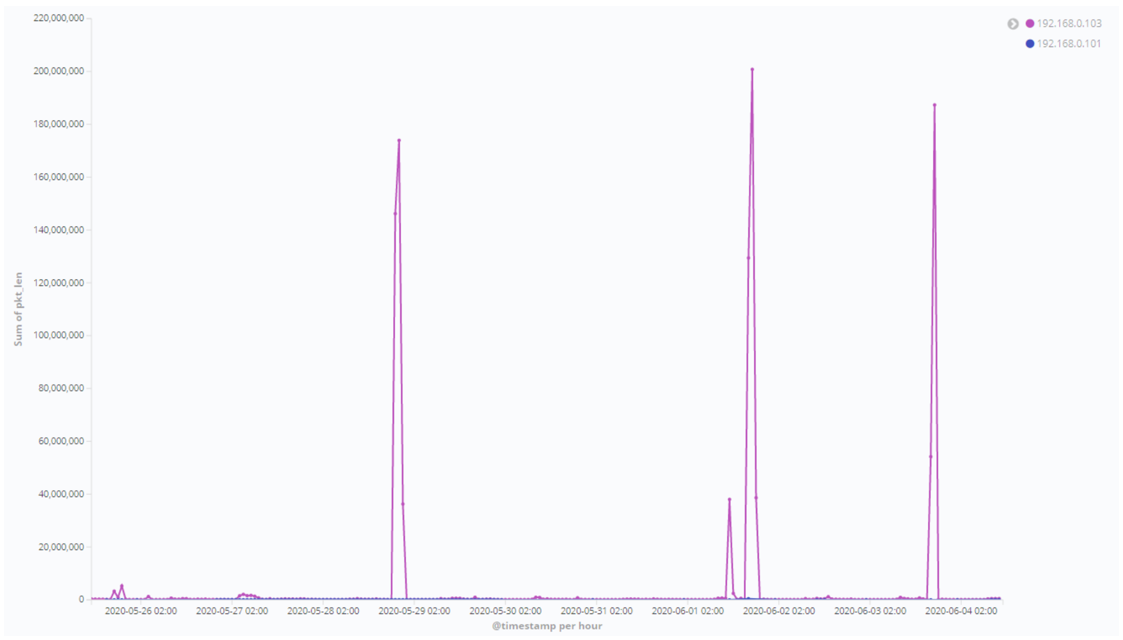


Figure 17: The traffic from the IP cameras from the well-known (192.168.0.101) and less-known (192.168.0.103) brands segments.

From Figure 17, it is clear that the IP camera from the less-known brands segment sent much more data than the camera from the well-known brands segment, approximately 200 times as much. Furthermore, it is interesting to note that the vast majority of the traffic sent by the camera from the less-known brands segment was sent using UDP while the camera from the well-known brands segment instead sent the majority of its data using TCP which can be observed in Appendix G.

Another device category that could be compared across the two segments was home automation devices. In Figure 18, one can see that the smart plug from the well-known brands segment sent little traffic after its initial configuration in comparison to the one from a less-known brands segment, which instead had a baseline of sending about 20 KB and sometimes spiking up to 60 KB per hour.

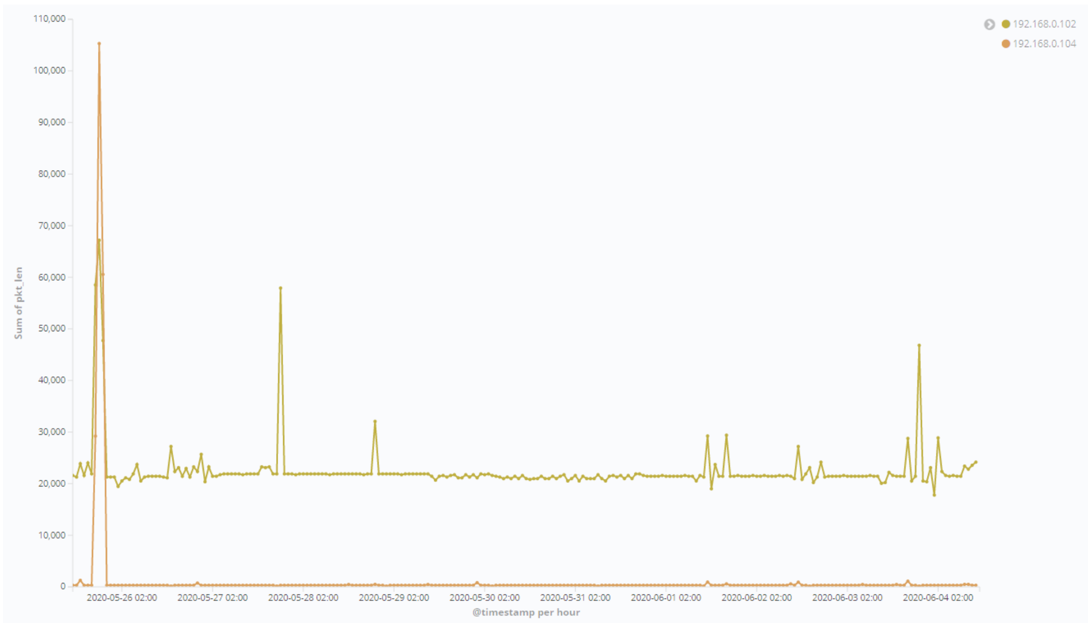


Figure 18: The traffic from the smart plugs from the well-known (192.168.0.104) and less-known (192.168.0.102) brands segments.

Besides the smart plugs, we also chose to include the Samsung SmartThings hub and the IKEA gateway from the well-known brands segment for comparisons of traffic volumes between different home automation devices, which can be seen in Figure 19.

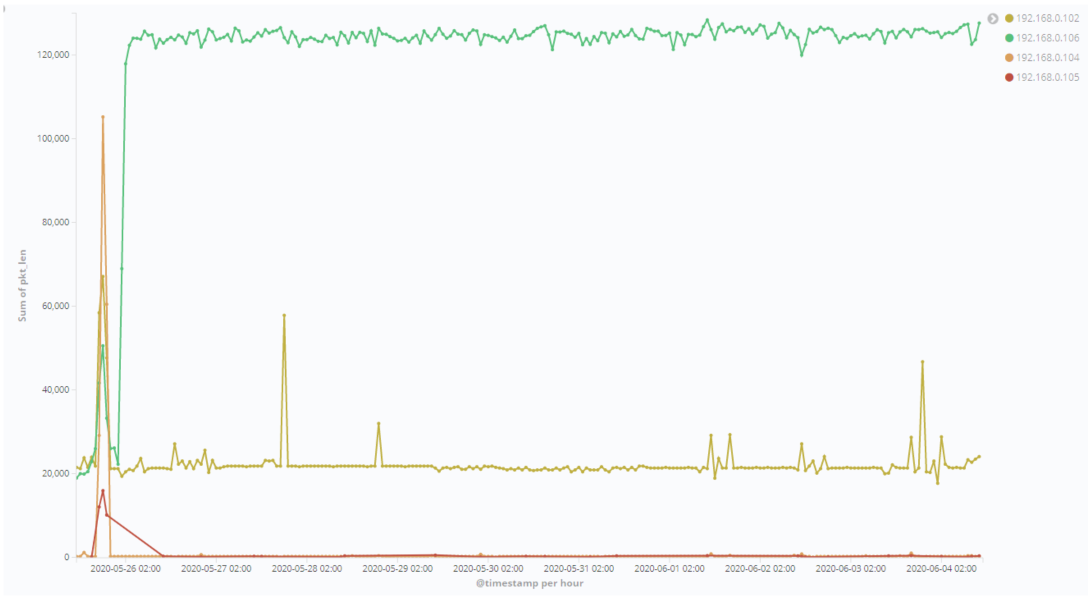


Figure 19: The traffic from the home automation devices from the well-known and less-known brands segments: the smart plug from the less-known brands segment (192.168.0.102), the smart plug from the well-known brands segment (192.168.0.104), the IKEA "trådfri" gateway (192.168.0.105) and Samsung SmartThings Hub (192.168.0.106).

For a more in-depth comparison of the total amount of data sent by these devices, see Appendix G.

7.2.2 Region Comparison

When comparing the traffic from the different regional segments, it is important to clarify that the traffic was captured during different time periods for the different segments. The traffic from the EU segment was captured earlier and will thus always be to the left whereas the traffic from the North American and Asian segments were captured later, meaning that their traffic will be further to the right when presented in the same graphs.

Nevertheless, when comparing the different regional segments, one can see that the IP camera from a less-known brands segment stands out, as can be seen in Figure 20. It sent large spikes of data compared to even the most talkative device from the other two regional segments. Interestingly though, the most talkative device from the latter two segments was also an IP camera (192.168.0.201).

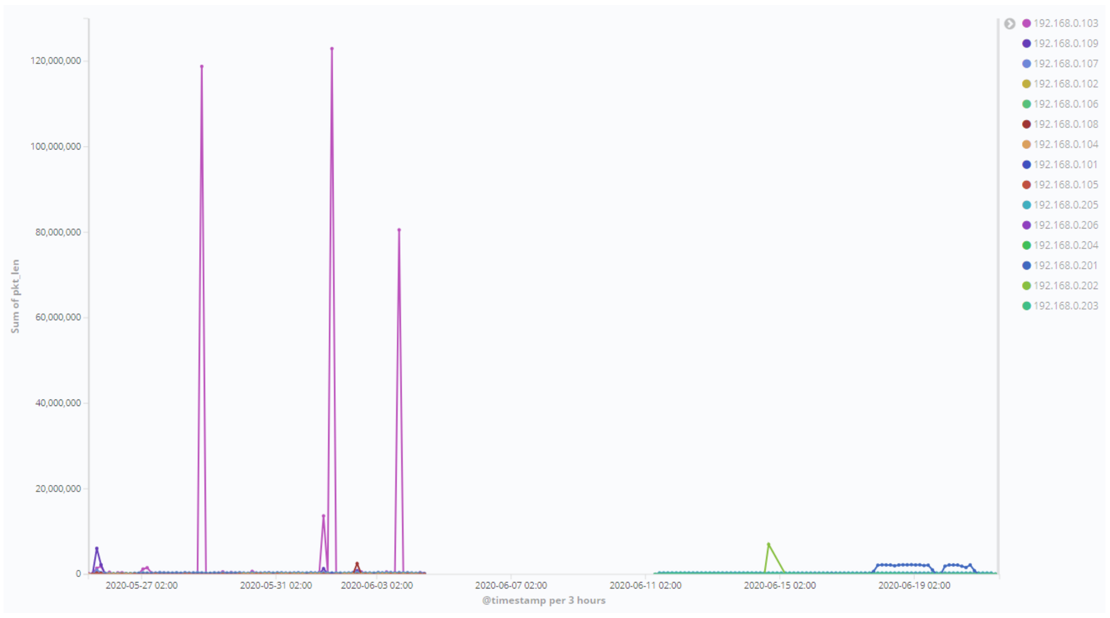


Figure 20: The traffic from the different regional segments side by side. The traffic from the EU segment is to the left and the traffic from the North American and Asian segments is to the right in the graph.

In order to be able to get a better look at the shape of the outgoing traffic of the devices in the North American and Asian segments, see Figure 21. In this picture, the IP camera from a less-known brand was removed due to how it scaled the graph.

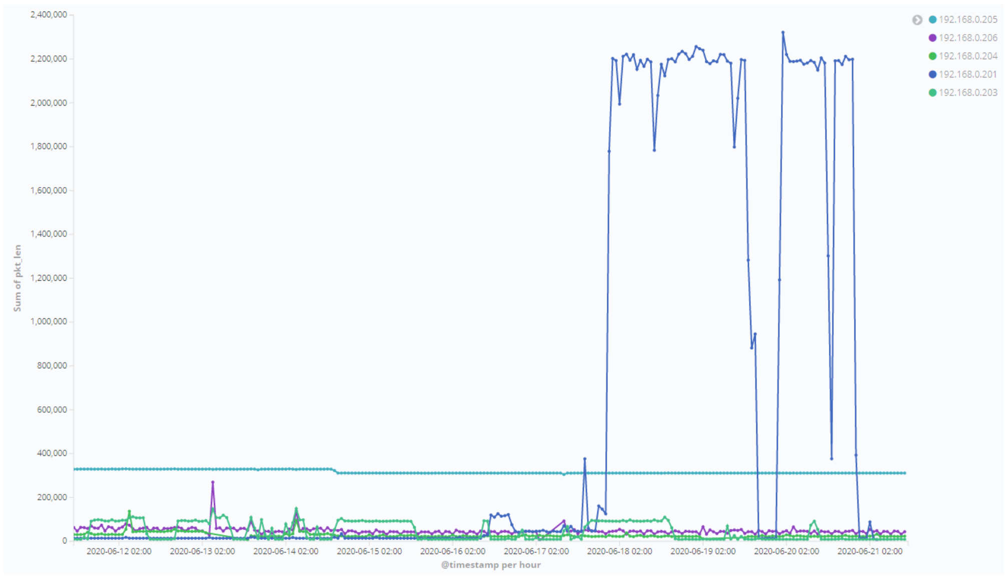


Figure 21: The shape of the outgoing traffic from the devices from the North American and Asian segments.

By a closer inspection of the traffic from the devices in the North American and Asian segments, it is clear that the IP camera from the Asian segment (192.168.0.201) sent almost twice the amount of data compared to the second most talkative device from these two segments, the smart plug from the Asian segment (192.168.0.205). See Appendix G for details.

Moving on to comparing devices from the same product category between the regional segments, the traffic from the different IP cameras can be seen in Figure 22.

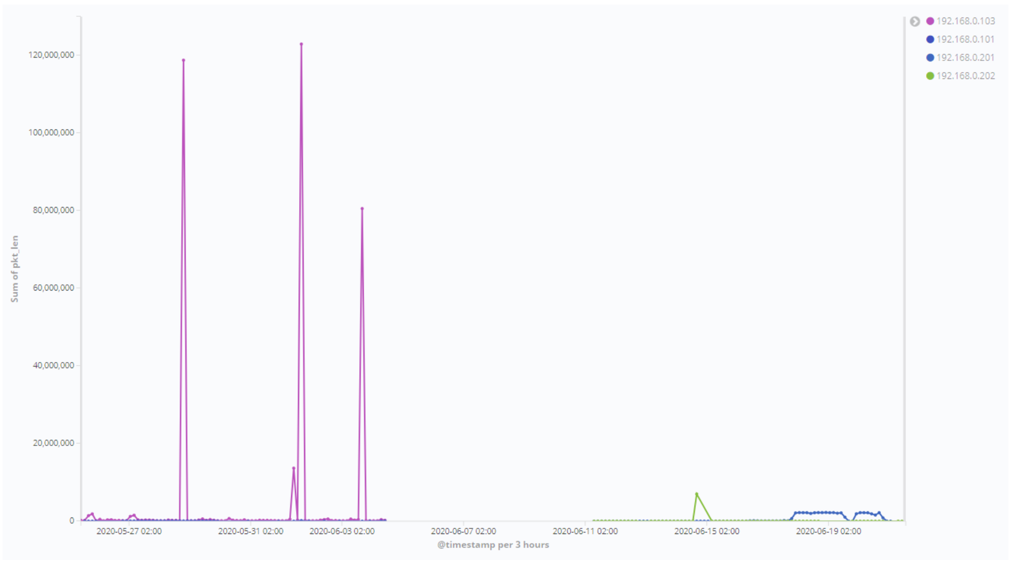


Figure 22: The traffic from the IP cameras from the EU (192.168.0.101 and 192.168.0.103) and the Asian (192.168.0.201 and 192.168.0.202) segments.

Similarly, the difference between the shape of the outgoing traffic from the home automation devices from the EU and Asian segments can be seen in Figure 23. As can be seen, the difference is significant. The smart plug from the Asian segment sent 76 MB across ten days, while the smart plugs from the EU segment sent 5.4 MB and 271 kB, respectively. See Appendix G for details.

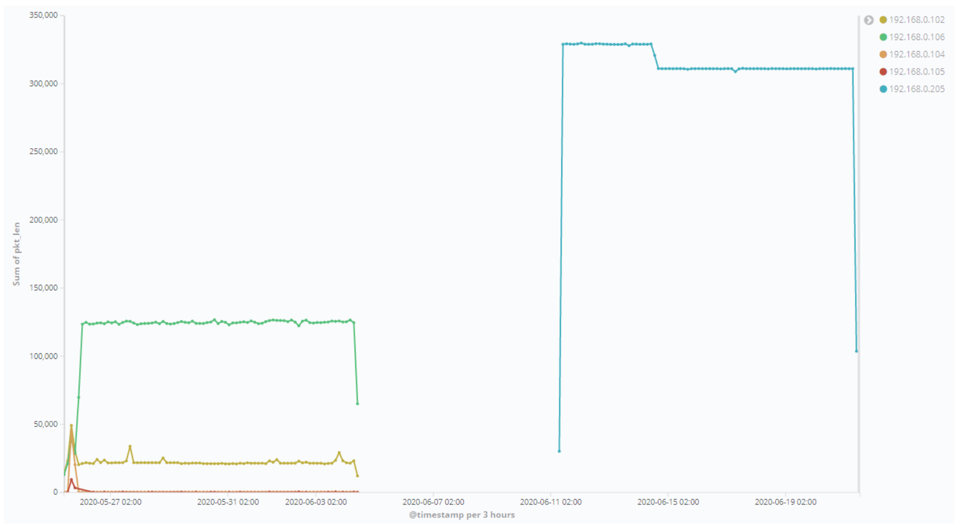


Figure 23: The traffic from the different home automation devices from the EU (192.168.0.102, 192.168.0.104, 192.168.0.105 and 192.168.0.106) and the Asian (192.168.0.205) segments.

The next category of devices that was compared on a regional basis was the smart speakers category. From Figure 24, it is noticeable that the smart speaker from the EU segment (192.168.0.107) sent more than its counterparts.

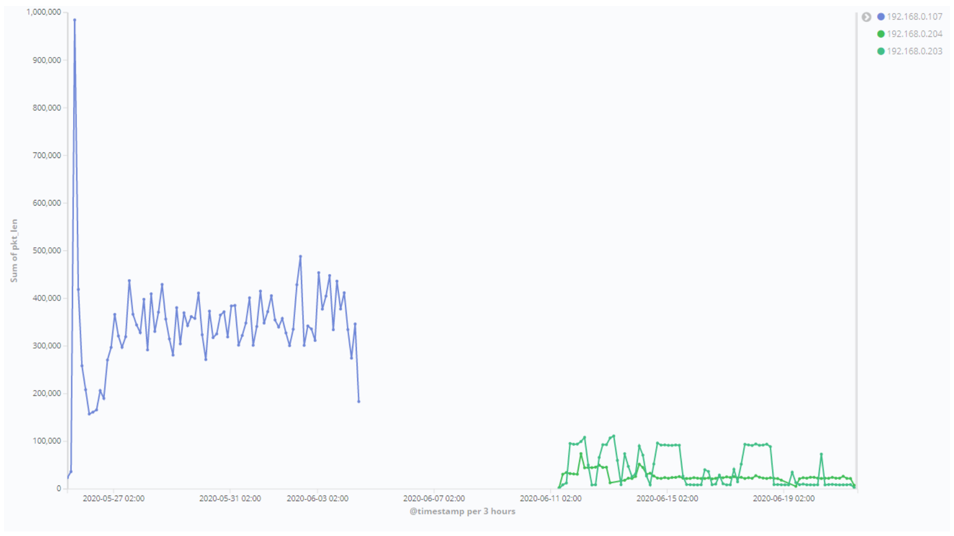


Figure 24: The traffic from the different smart speakers from the EU (192.168.0.107) and the North American (192.168.203 and 192.168.0.204) segments.

Finally, the last device category that was compared between the regional segments was streaming devices. The result can be seen in Figure 25.

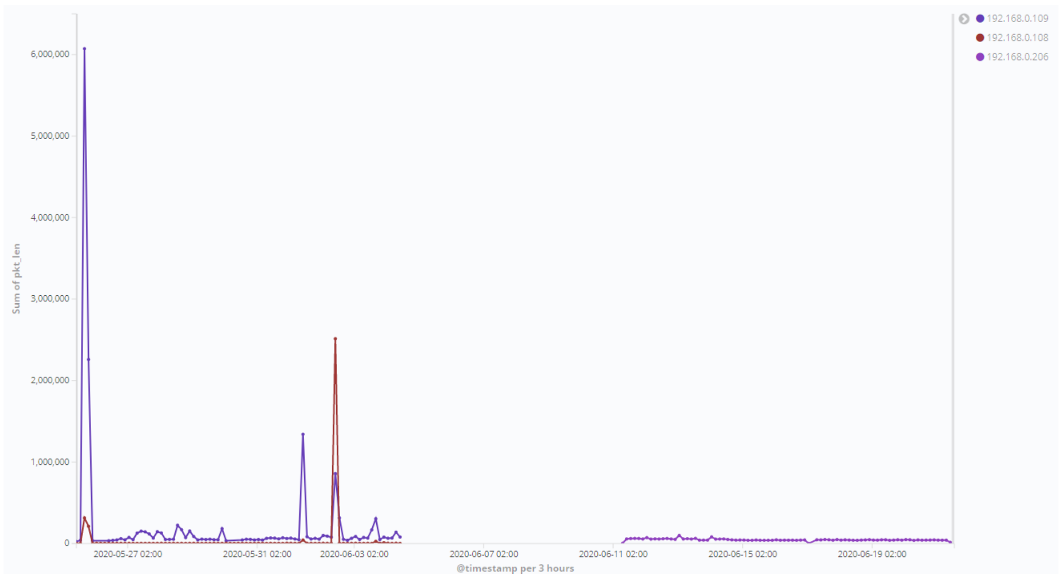


Figure 25: The traffic from the different streaming devices from the EU (192.168.108 and 192.168.0.109) and the North American (192.168.0.206) segments.

Up until this point, we have exclusively only shown traffic that was sent from the devices in the testbed. However, given that some device categories such as streaming devices are expected to receive much more data than they send, we also looked at incoming traffic for all devices, which can be seen in Figure 26.

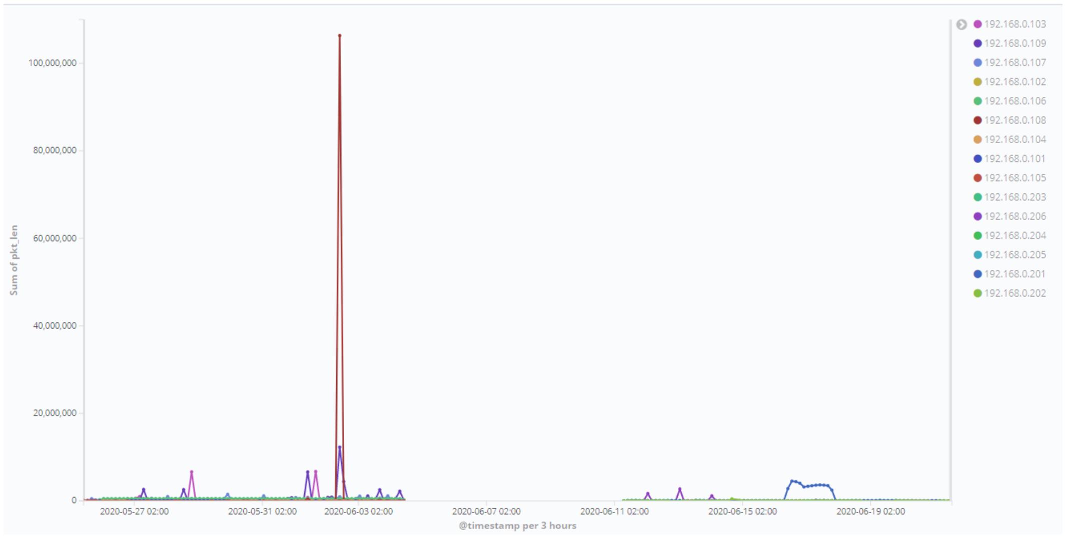


Figure 26: The incoming traffic to all the devices in the testbed.

Since the incoming traffic to the Apple TV (192.168.0.108) was significantly larger than for any other device, and in order to be able to observe the other devices' incoming traffic more easily, Figure 27 shows the results excluding the Apple TV.

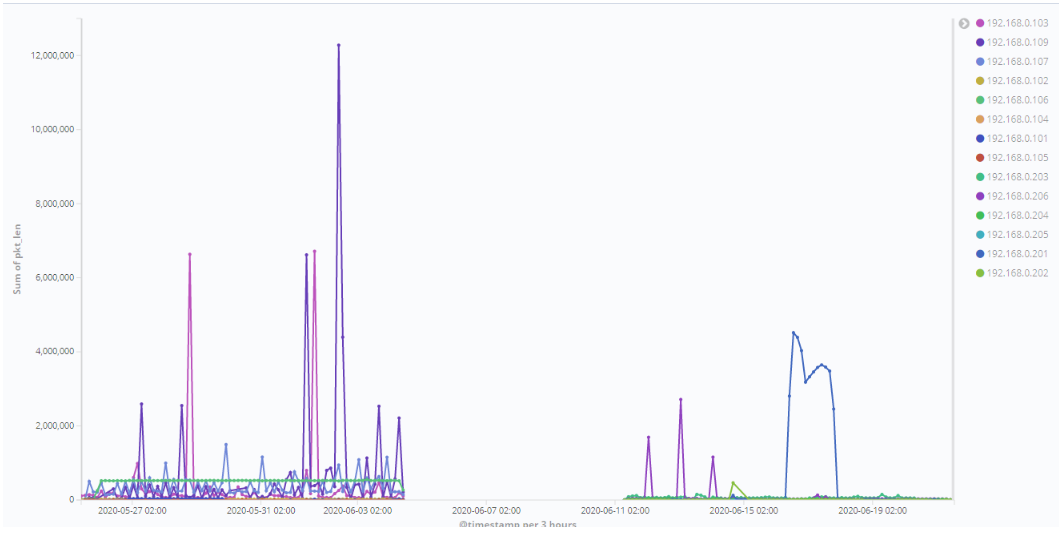


Figure 27: The incoming traffic to all devices in the testbed excluding the Apple TV.

For details about the incoming traffic to the different devices, see Appendix G.

7.3 Unexpected Device Behaviour

This section will present the results related to unexpected device behaviour observed during the experiments as well as lack of encryption.

It begins with a section dedicated to comparing the devices from the well-known to the less-known brands segment and concludes with a section comparing the three regional segments.

7.3.1 Brand Comparison

Regarding unexpected device behaviour, the first thing we noticed was the behaviour of the different smart plugs in the home automation category. The smart plug from the less-known brands segment sent far more data than the one from the well-known brands segment. However, when we looked at the top ten endpoints the smart plug from the less-known brands segment contacted the most, all of them were to aws.amazon, as can be seen in Appendix F. In contrast, the smart plug from the well-known brands segment sent significantly less data, although to more interesting endpoints, which can also be seen in Appendix F. For instance, it sent traffic to privately-owned IP addresses.

When it comes to the cameras, both contacted our home network cust.bredband2.com and sent traffic to a NAS residing on that network despite the fact that that network was separated from the test bed.

Regarding the threat related to lack of encryption, one of the IP cameras sent 621.4 kbs unencrypted data on 2020-06-01 around 13.00, as can be seen in Figure 28.

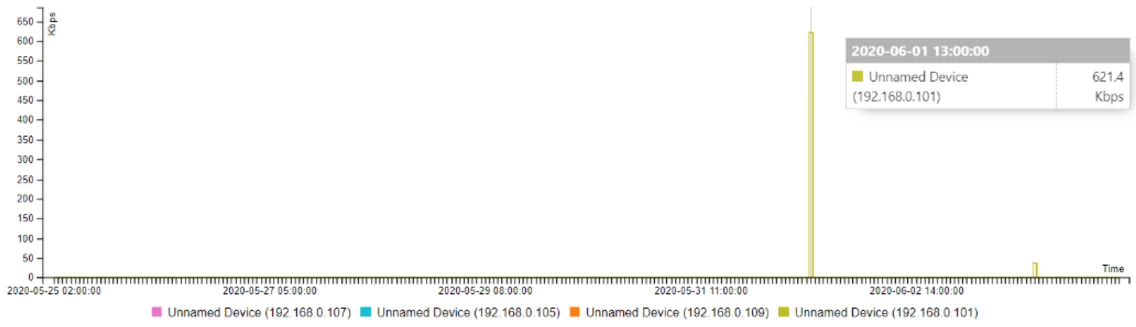


Figure 28: Unencrypted traffic from the devices in the EU testbed that was observed by Princeton IoT Inspector.

Besides that, there were no significant spikes of unencrypted data according to Princeton IoT Inspector for the EU segment.

7.3.2 Region Comparison

When it comes to the devices in the North American and Asian segments, a strange behaviour from one of the IP cameras (192.168.0.201) from the Asian segment was detected. It had a problem maintaining the reserved IP address that the DHCP server had given it. This resulted in it dropping its IP address twice. Being able to assign dedicated IP addresses are important in order to be able to track the device behaviour. For that reason, this behaviour was problematic in regards to privacy.

Secondly, it was clear that the smart plug from the Asian segment sent far more data than expected, especially compared to its EU counterparts. It behaved in a similar way to the smart plug from the less-known brands segment, i.e. it continuously sent data and sometimes dipped, but never down to 0 bytes. The difference between the two was the amount of data sent though. The smart plug from the Asian segment sent 76 MB while the most talkative smart plug from the EU segment sent 5.4 MB.

Another unexpected observation was that one of the cameras from the Asian segment continuously sent data over long time periods. For example, it would start sending data at three in the morning local time and stop ten hours later. This could arguably be considered unexpected. Similarly, sending data over a long time spans was something we observed for the Chromecast from the North American segment as well. It sent around 250 kB per hour on average for up to 10 hours at a time.

In regards to lack of encryption, among the North American and Asian devices, a few sent unencrypted data, as shown in Figure 29.

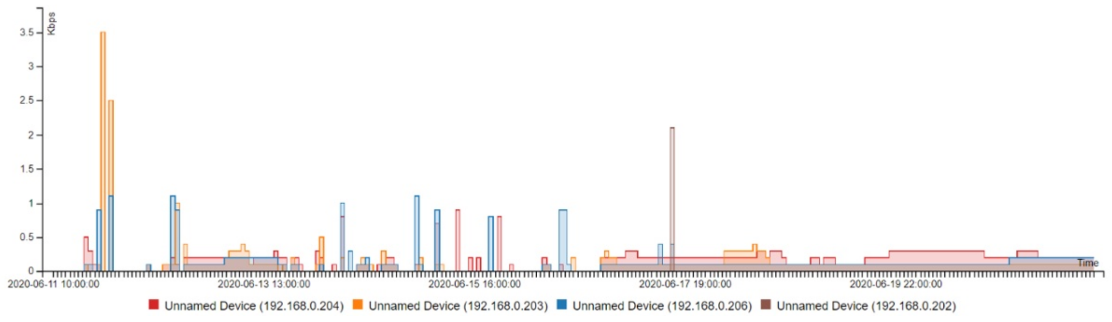


Figure 29: Unencrypted traffic from the North American and Asian devices that was observed by Princeton IoT Inspector.

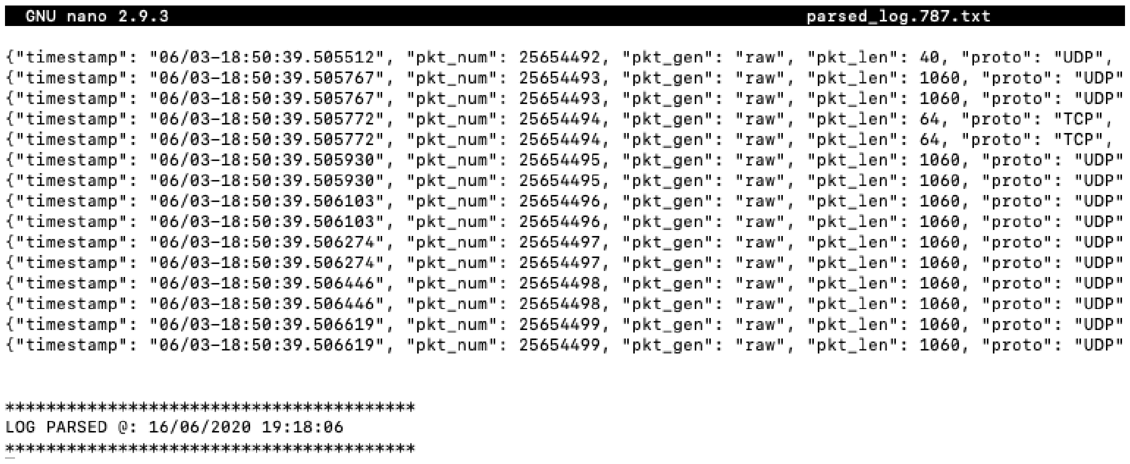
7.4 Cloud Solution

This section presents the results from the cloud solution. It begins with showcasing how the code could be verified and moves on to discuss the ratio between payload and headers/metadata of the Snort logs. Finally, we include a screenshot confirming the integration of the cloud rules in Snort and that they indeed could be observed in Kibana. For details on the code and cron jobs used to obtain the results, see Appendix C and D.

7.4.1 Observations from the Cloud Solution

The cloud-related infrastructure (code and cron jobs) performed much on par with the specification. The idea was to parse the Snort logs to extract only the metadata, send these parsed logs to a web server at AWS and then to fetch the rules developed on the cloud. All of this should also be automated.

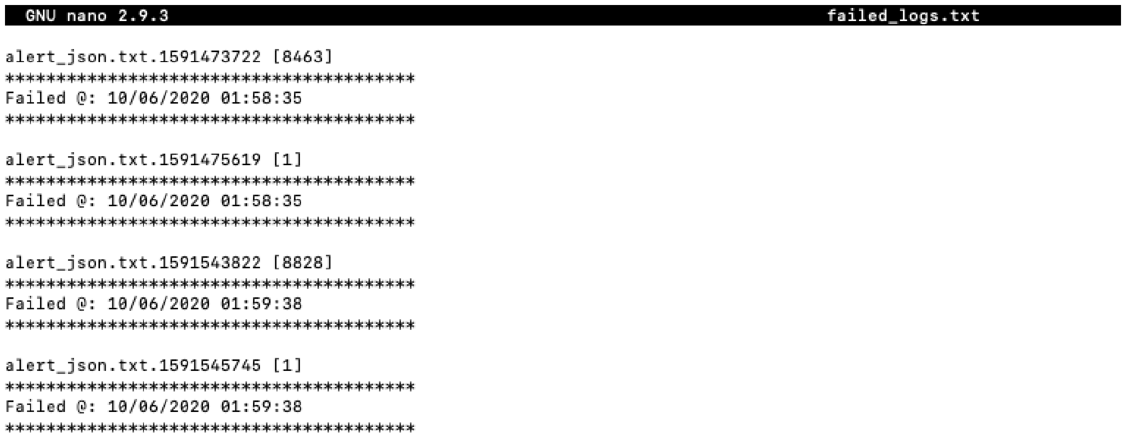
The code responsible for parsing the logs was *parser.py*. The parser added a timestamp at the end of the parsed log file in order to be able to keep track of when the log was parsed, see Figure 30.



```
  GNU nano 2.9.3                                                    parsed_log.787.txt

{"timestamp": "06/03-18:50:39.505512", "pkt_num": 25654492, "pkt_gen": "raw", "pkt_len": 40, "proto": "UDP",
{"timestamp": "06/03-18:50:39.505767", "pkt_num": 25654493, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.505767", "pkt_num": 25654493, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.505772", "pkt_num": 25654494, "pkt_gen": "raw", "pkt_len": 64, "proto": "TCP",
{"timestamp": "06/03-18:50:39.505772", "pkt_num": 25654494, "pkt_gen": "raw", "pkt_len": 64, "proto": "TCP",
{"timestamp": "06/03-18:50:39.505930", "pkt_num": 25654495, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.505930", "pkt_num": 25654495, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506103", "pkt_num": 25654496, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506103", "pkt_num": 25654496, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506274", "pkt_num": 25654497, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506274", "pkt_num": 25654497, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506446", "pkt_num": 25654498, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506446", "pkt_num": 25654498, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506619", "pkt_num": 25654499, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"
{"timestamp": "06/03-18:50:39.506619", "pkt_num": 25654499, "pkt_gen": "raw", "pkt_len": 1060, "proto": "UDP"


****************************************
LOG PARSED @: 16/06/2020 19:18:06
****************************************
```

Figure 30: A parsed log file. Each row in the file corresponds to a network packet.

However, since Snort writes the packet capture information in real time to the log file it sometimes makes mistakes, at least given the way we set it up. This was especially true for the end and the beginning of a new log file (we restricted the log files to a size of ten megabytes). For that reason, the parsing portion of the code in *parser.py* was wrapped in a try-catch statement in order to catch the lines/packets that failed to be parsed. These errors were then saved to a file called *failed_logs.txt* which contained information about which log files and row numbers caused the errors as well as timestamps of when these errors occurred. This made it easy to backtrack and see what packet caused the errors. See Figure 31 for details on how these errors were presented to the user.



```
  GNU nano 2.9.3                                                      failed_logs.txt

alert_json.txt.1591473722 [8463]
****************************************
Failed @: 10/06/2020 01:58:35
****************************************

alert_json.txt.1591475619 [1]
****************************************
Failed @: 10/06/2020 01:58:35
****************************************

alert_json.txt.1591543822 [8828]
****************************************
Failed @: 10/06/2020 01:59:38
****************************************

alert_json.txt.1591545745 [1]
****************************************
Failed @: 10/06/2020 01:59:38
****************************************
```

Figure 31: The *failed_logs.txt* kept track of which packets were not sent to the cloud. It revealed which log file and packet [row] failed to parse and at what time.

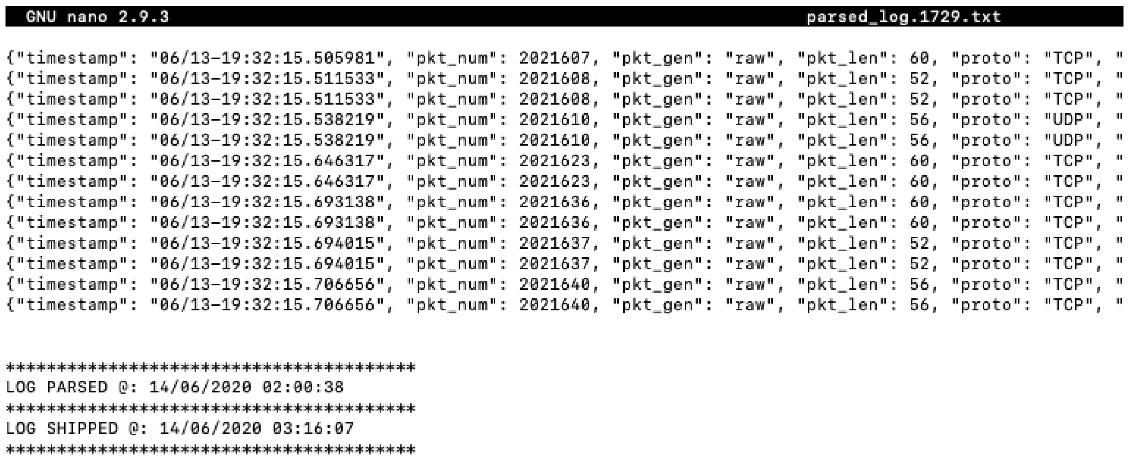
Both the EU logs as well as the North American and Asian logs had comparatively low fail rates ($\approx$ 0.0001% of the packets), see Table 7.1. This indicated that the parser was robust and that Snort rarely produced mistakes.

	Number of packets	Failed packets
EU	$\approx$ 7.3M	0
North America & Asia	$\approx$ 6.5M	6

Table 7.1: The number of packets captured for the different segments and how many of them failed to parse.

Much in the same way as the *parser.py*, the *shipper.py*, responsible for shipping the logs to the server at AWS, added a timestamp to the log right before it was shipped. Once the log was confirmed to be delivered by the back-end server at AWS, the shipper deleted the parsed log. The decision of whether to delete the log or not was based on the status code of the response from the web server. If the status code was

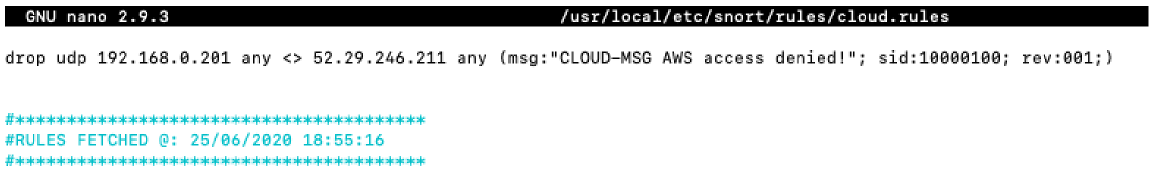
200 or 201 (success), the shipper was allowed to delete the log. Instead, if more than three subsequent attempts failed, the shipper stopped the execution of the script, removed the timestamp and ignored the deletion of the parsed log. See Figure 32 for how a parsed log might look after it was received at the AWS server.



```
 GNU nano 2.9.3                                               parsed_log.1729.txt

{"timestamp": "06/13-19:32:15.505981", "pkt_num": 2021607, "pkt_gen": "raw", "pkt_len": 60, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.511533", "pkt_num": 2021608, "pkt_gen": "raw", "pkt_len": 52, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.511533", "pkt_num": 2021608, "pkt_gen": "raw", "pkt_len": 52, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.538219", "pkt_num": 2021610, "pkt_gen": "raw", "pkt_len": 56, "proto": "UDP", "
{"timestamp": "06/13-19:32:15.538219", "pkt_num": 2021610, "pkt_gen": "raw", "pkt_len": 56, "proto": "UDP", "
{"timestamp": "06/13-19:32:15.646317", "pkt_num": 2021623, "pkt_gen": "raw", "pkt_len": 60, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.646317", "pkt_num": 2021623, "pkt_gen": "raw", "pkt_len": 60, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.693138", "pkt_num": 2021636, "pkt_gen": "raw", "pkt_len": 60, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.693138", "pkt_num": 2021636, "pkt_gen": "raw", "pkt_len": 60, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.694015", "pkt_num": 2021637, "pkt_gen": "raw", "pkt_len": 52, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.694015", "pkt_num": 2021637, "pkt_gen": "raw", "pkt_len": 52, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.706656", "pkt_num": 2021640, "pkt_gen": "raw", "pkt_len": 56, "proto": "TCP", "
{"timestamp": "06/13-19:32:15.706656", "pkt_num": 2021640, "pkt_gen": "raw", "pkt_len": 56, "proto": "TCP", "


****************************************
LOG PARSED @: 14/06/2020 02:00:38
****************************************
LOG SHIPPED @: 14/06/2020 03:16:07
****************************************
```

Figure 32: How a log file might look after it has arrived at the AWS web server.

A few hours after the latest logs had been parsed and shipped, the cloud rules were fetched by the local Ubuntu server. The fact that the rules were the latest available could be confirmed by the timestamp (that it contained the current date). See Figure 33 for details on how the cloud rules looked after they had been fetched by the local server. Worth noting though, since these rules solely served the purpose of a proof of concept, the actual rule in the rule file served little to no purpose.



```
 GNU nano 2.9.3                                  /usr/local/etc/snort/rules/cloud.rules

drop udp 192.168.0.201 any <> 52.29.246.211 any (msg:"CLOUD-MSG AWS access denied!"; sid:10000100; rev:001;)


#****************************************
#RULES FETCHED @: 25/06/2020 18:55:16
#****************************************
```

Figure 33: How the fetched cloud rules file might look. The timestamp tells when the rules were downloaded.

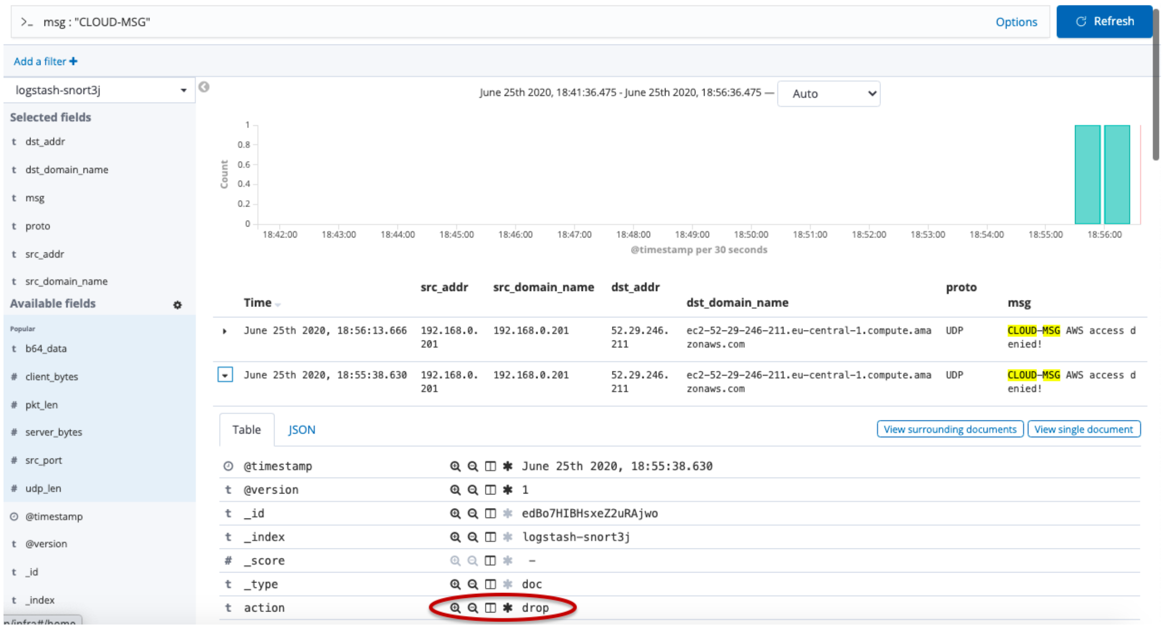
Since one of the arguments to why a parser was developed in the first place was to reduce the size of the data sent to the cloud, it was important to investigate the size drop after the payload had been parsed out. Moreover, in order to be able to reason about the solution's viability, it was necessary to take a more detailed look into the ratio between the actual logs and the parsed logs. Table 7.2 shows the size of all the logs before and after the payload had been parsed out. Worth noting, since we observed many ICMP messages, especially to Google's public DNS servers, we checked the ratio for two cases, both including and excluding ICMP packets.

	Logs	Parsed logs	Payload
EU	≈ 8.5GB	≈ 5.6GB	≈ 34%
North America & Asia	≈ 8.3GB	≈ 5.2GB	≈ 37%
EU (Excluding ICMP packets)	≈ 8.0GB	≈ 5.4GB	≈ 33%
North America & Asia (Excluding ICMP packets)	≈ 7.9GB	≈ 5.0GB	≈ 37%

Table 7.2: The accumulated size of all logs before and after the payload portion of the packets had been parsed out.

As one can see, the payload was roughly one-third of the total size of the packets for both the EU as well as North American and Asian segment (the North American and Asian segments were measured together given the way the experiment was set up). Surprisingly, by removing the ICMP packets, the ration of the payload was more or less unchanged (for one of the cases it even decreased). This indicated that the logs contained many empty, or at least very small, packets like TCP Keepalive packets.

Finally, the rules were manually fetched and confirmed to be successfully integrated into Snort by observing the output in Kibana. Given that the cloud rules were tagged with *CLOUD-MSG* in the msg field, one can see in Figure 34 that they indeed were incorporated into the ruleset used by Snort.



```
>_ msg : "CLOUD-MSG"

logstash-snort3j

June 25th 2020, 18:41:36.475 - June 25th 2020, 18:56:36.475 —

Selected fields
t dst_addr
t dst_domain_name
t msg
t proto
t src_addr
t src_domain_name
Available fields
Popular
t b64_data
# client_bytes
# pkt_len
# server_bytes
# src_port
# udp_len
@timestamp
t @version
t _id
t _index

Time                         src_addr       src_domain_name  dst_addr      dst_domain_name                                proto  msg
June 25th 2020, 18:56:13.666 192.168.0.201  192.168.0.201    52.29.246.211 ec2-52-29-246-211.eu-central-1.compute.amazonaws.com UDP CLOUD-MSG AWS access denied!
June 25th 2020, 18:55:38.630 192.168.0.201  192.168.0.201    52.29.246.211 ec2-52-29-246-211.eu-central-1.compute.amazonaws.com UDP CLOUD-MSG AWS access denied!

@timestamp    June 25th 2020, 18:55:38.630
t @version    1
t _id         edBo7HIBHsxeZ2uRAjwo
t _index      logstash-snort3j
# _score      -
t _type       doc
t action      drop
```

Figure 34: The successfully integration of the cloud rules into Snort's ruleset could be observed and verified in Kibana.

CHAPTER 8

Discussion

This chapter will discuss the results from the previous chapter as well as the different software solutions used in the testbed in order to answer the aims of the thesis. Moreover, technical limitations which to a varying degree affected decisions and outcomes are discussed as are ethical issues, sustainability aspects and recommendations on future work.

The chapter begins with a section dedicated to the results from both the different device segments and cloud solution presented in Chapter 7. Endpoints, the shape of traffic and unexpected device behaviour are addressed in this section as is the viability of a cloud solution.

The second section discusses the different software solutions used throughout the experiments. That is, how useful they are in detecting and/or understanding privacy threats to smart home as well as their applicability to different users.

This is followed by three separate sections reasoning about technical limitations, ethical issues and sustainability aspects.

The chapter concludes with a section dedicated to future work, in which we discuss how the scientific community might move forward in regards to privacy and consumer IoT.

8.1 Interpreting the Results

This section is dedicated to discussing the results presented in Chapter 7. It begins with three subsections comparing the different device segments in relation to the threats in the threat model from three different aspects: endpoints, the shape of the traffic and unexpected device behaviour. The final subsection discusses the results of the cloud solution and its viability as a future approach to improving privacy in the smart home.

8.1.1 Endpoints

Across all segments, the traffic mainly terminated in one of three regions: Europe, North America or Asia. One explanation to this is most likely that the devices in the testbed were aimed for these markets. Nevertheless, it does not explain why some devices from the less-known brands segment to a large extent terminated in China or some devices from the well-known brands segment in the US and so on. Nevertheless, the results show that there arguably is a concentration of geographical endpoints.

From the results, one can observe that there is also a high concentration of cloud providers. What device relies on which provider depends to a large degree on what market the device is aimed for but also its brand recognition as many of the top endpoints for the devices from the less-known brands segment contacted Chinese cloud providers. This is not apparent by simply applying a geographical perspective on the traffic, since many of these providers do have servers across the world. Moreover, just looking at which company owns the domain might not either be sufficient as different regions might be subject to different privacy legislation. This shows the importance of applying multiple perspectives when analysing endpoints.

When it comes to endpoints associated with advertisement or tracking, only the Google devices showed up as sending data to third-party ad servers or trackers. This is most likely because they use well-known advertisement servers. For the other devices, we could not find any such behaviour. This might indicate that they are not subject to third party tracking but might just as well mean that Princeton IoT Inspector did not know about the domains used for tracking by those devices.

8.1.2 The Shape of the Traffic

In regards to the shape of traffic, there were some noticeable trends. For example, certain cameras and streaming devices had large spikes in both their outgoing and incoming traffic. Since cameras also received relatively large volumes of data compared to other device categories, it was hard to determine whether or not a device was a camera or a streaming device based on the shape of the traffic. The streaming devices also used different transport layer protocols, adding an additional layer of obfuscation. For example, the Apple TV received data primarily sent using UDP, whereas other streaming devices mostly received TCP packets. Though if the devices are compared to each other, it is noticeable that an IP camera sends more than it receives while streaming devices receive more data. Which means that if a smart home has both device categories, it would be easier to determine which one is which.

Similarly, it was hard to determine if a device was a home automation device or not, though some devices shared similarities. The Asian smart plug had a similar shape to that of the smart plug in the less-known brands segment and the Samsung smart hub from the well-known brands segment. However, the smart plug and the IKEA gateway from the well-known brands segment stayed mostly idle, meaning that their device type could not be determined based on the shape of their traffic.

As for the last device category, the smart speakers, they acted quite differently from one another. The Google Home Mini from the EU segment was prone to sending larger volumes of data in a pattern with many spikes and drops. In comparison, the North American Google Home Mini and the Amazon Echo Dot acted similarly to each other. The main difference between the two geographical segments was that the North American devices sent a consistent volume of data over larger time spans and much less than their European counterpart. While they have a unique behaviour compared to other devices, they all behave quite differently from one another, which makes it hard to identify all of them as belonging to a specific device category.

In conclusion, it was hard to determine precisely what kind of device something was based on the shape of its traffic. For instance, a smart speaker and a smart plug look similar while idle and an IP camera could behave similarly to a streaming device, depending on the manufacturer. However, while it might be difficult to determine precisely what type of device something is based on the shape of traffic, it is possible to determine what it is not. For example, it would be hard to tell if Device A is a camera or a streaming device, but it is possible to say it is not a smart plug.

8.1.3 Unexpected Device Behaviour

The main observation regarding strange behaviour was that most of the devices that exhibited what we called strange behaviour sent more data than expected. This was especially true for the smart plugs from the Asian and the less-known brands segments. The Asian device, in particular, was the second most talkative device in its segment. This is concerning since we do not believe that anyone expects a smart plug to send that much data.

For the other device categories, the Asian devices and the devices in the less-known brands segment were by far the most talkative. These devices also showed interesting behaviour when it comes to what times they were most active. They tended to upload data during late evening or early morning local time, meaning outside Swedish waking hours.

As for endpoints, there were two conspicuous observations. One was that the NAS was a popular target for multiple devices and accessible although it resided on a separate network. We believe this was related to the devices exploiting application privileges in the Android operating system. The second was how the smart plug from the well-known brands segment contacted domains we interpreted as belonging to individuals. We assume this was due to us forgetting to reset the plug before connecting it to the testbed. That is, it might have been a resold product.

Finally, regarding encryption, a majority of the devices sent unencrypted traffic as reported by Princeton IoT inspector although to a small extent. This was much on par with observations from other studies.

8.1.4 The Viability of a Cloud Solution

Cloud-developed rules is an intriguing idea, using the expertise and resources of a third party rather than solely relying on user or community know-how. From a strictly technical perspective, the results from the cloud solution experiment show that it is feasible to parse and ship the logs to the cloud and at the same time automate this process to make it transparent for the user. Furthermore, given the amount of log entries passed through the parser, only a small portion of the packets failed to parse, which indicates that Snort rarely makes mistakes when capturing the packets. Although a proof of concept, we also showed that one quite easily can incorporate fault tolerance such as keeping track of failed parsed logs or having safety mechanisms in place for the shipping process in case any issues arise. Both can be considered important if the solution is intended to be run without the need for user intervention.

However, if it is to be implemented in a commercial setting, the size quickly becomes an issue. The logged packets, stored as JSON, were sometimes multiple times larger than the actual packet size and the ratio between headers and payload was problematic. This raises the question: how can one improve upon it? We can think of a few approaches.

First, allowing the cloud rules provider to specify which packets to exclude would significantly reduce the amount of logs that need to be sent. As an example, by excluding ICMP and/or TCP Keepalive packets alone this would remove a large amount of the sent packets. As seen in Chapter 7, just from our testbed, million of packets would have been sent if we would have had the automation running alongside the other experiments the whole time without any such considerations.

Secondly, for those packets that are to be included, one could allow the service provider to specify which headers they want to keep in the parsed log. This would reduce the size of each entry and change the ratio between headers and payload. In our setup, we chose to include many headers which for obvious reasons did affect the ratio between the two.

Thirdly, by sending the logs less regularly as well as compressing them before shipping them to the cloud would impact both the volume and frequency of the traffic to the service provider and make the solution more scalable. Moreover, this would also reduce the stress on Snort by not having to be restarted every night in order to incorporate the new rules.

Finally, other formats than JSON might be used. One approach would be to send the logs in more space efficient formats that Snort natively supports such as unified2. However, despite the gain from a size reduction, this might require the receiving part to translate it back to a format that can be used in their systems. Nevertheless, as a general rule, and to support the involvement of multiple parties, it is important to avoid proprietary formats when designing the solution.

8.2 Software Solutions

This section will discuss the pros and cons of the software solutions used throughout the experiments. They will be compared in regards to four main aspects: clarity, customisability, expandability and user-friendliness.

8.2.1 Which is the Better Software?

When it comes to clarity, both Princeton IoT Inspector and Snort in combination with the ELK Stack allow the user to observe and visualise the traffic from the smart home devices in multiple ways. One neat trait of Princeton IoT Inspector though is that it for some domains can give hints on what kind of service that domain is associated with. Another useful privacy feature it has is that it helps the user to identify which devices contact known trackers or advertisement servers as well as send unencrypted traffic. These threats can be observed in regards to both frequency and volume in the software. This is something neither Snort nor the ELK stack offer by default.

Princeton IoT Inspector is also able to both filter the data on a per device basis as well as supply the user with a good bird's view on the overall traffic from the smart home network from its summary tables. However, this was also one of its major drawbacks, that is, its lack of ability to tailor the analysis to the user's preferences. More specifically, not allowing the user to specify fixed time intervals for all visualisations turned out to be troublesome for experiments such as the ones conducted for this thesis. However, it did partly support it for the graphs, by allowing zoom operations in the graphs. Nevertheless, that was not sufficient for all use cases. Sometimes tables are more suitable than graphs to visualise certain aspects of the traffic. For instance, by not being able to filter endpoints by time in the endpoints lists made Princeton IoT Inspector's endpoints analysis tool much less useful compared to solutions such as Kibana.

In contrast, Snort supports a wide range of data fields that one can choose to include in the capture. Furthermore, given that Kibana can provide a detailed and easily accessible way to look at specific packets, this proved to be a formidable combination, especially when looking for anomalies or unexpected device behaviour. The usefulness of Kibana's wide selection of visualisations was to a high degree dependent on the quality of the data/logs fed into Elasticsearch. Snort's detailed captures were one of the reasons for why Kibana was a good choice for visualising the logs; however another was some of the many filters that Logstash offered. By allowing filters such as the DNS filter, Kibana could provide much of the same functionality as Princeton IoT Inspector did, while at the same time allowing for additional ones. One such example was the use of the Geoip filter, adding geolocation data to the logs.

For experiments such as the ones in this thesis, or for the scientific community in large, a high level of customisability might be needed, which makes Princeton IoT Inspector the less suitable choice, at least in its current state. However, if only the endpoints panel would allow for user-defined time intervals, that would be a significant improvement. Instead, Snort and the ELK Stack supplied everything that we needed for the experiments. Despite the high level of customisability of the ELK stack, it does not require that much configuration in order for it to be useful. For instance, although Kibana offers formats such as Vega to create custom visualisations, one might just as well choose to rely on only the included visualisations as we did. Similarly, the idea of Logstash to separate input, filters and output enables a highly modular design where different developers can develop new filters/plugins, while it, at the same time, allows the end-users to tailor the log management to their needs. The same argument can be said for Snort, where the level of customisability is high, although it is not that complicated to get started with since there is a lot of documentation available on how to get started.

Although security aspects are the focus for Snort, thanks to its modular design and the ELK stack's excellent visualisations, it proved to be highly useful for analysing privacy threats as well. Though, with the standard configuration and using the default community rules, it offered little to no extra insights into privacy threats to the network alone.

Although we might prefer Snort and ELK solution for log management, we realise that for the typical smart home user the shortcomings we experienced with Princeton IoT Inspector might not be as significant. Instead, the high level of customisability provided by Snort and the ELK stack comes with a price - user-friendliness. Although Princeton IoT Inspector during the spring of 2020 repeatedly changed which version they offered to the user, it probably is still the best option for the average smart home user. Moreover, when Princeton IoT Inspector is available for Unix (and hopefully supported on ARM CPU:s) more people will be able to use it. Given the widespread adoption of the raspberry pi, support for the Raspbian operating system would be a welcome development.

There is often a trade-off between user-friendliness and customisability, and depending on the use case, one solution might be better suited than the other. Nevertheless, for experiments such as the ones in this thesis, Snort in combination with the ELK stack was the better choice.

8.3 Technical Limitations

Doing our best to represent the different segments as fairly as possible was a challenge. Once we had selected our devices, we could not test all of our selections for two different reasons. The first was COVID-19 causing long delays in our orders for any devices outside the EU. Any device we did not receive in time for the experiment was not included. Another issue we found was related to a US version of a Samsung SmartThings hub. For some reason, we did not manage to configure it and had to exclude it from the test, eliminating cross-segment smart hub comparisons.

Another problem we faced was that Princeton IoT Inspector was initially only available for Mac OS, though this stopped being supported shortly before we started the experiments. This meant we needed to use the officially supported Windows client instead. The source code was there for Linux and Mac OS, but we could not get it to work. The Windows client was buggy and prone to crash, so there are gaps in the monitoring when the software was not running. These gaps mainly occurred over the night since it would crash in the evening, and we would not notice it until the following morning.

The lack of documentation on Snort version 3 was another limiting factor. There are quite a lot of changes from the previous version and some aspect of it would benefit from being more thoroughly explained in the official documentation/manual. As a consequence, we decided to go with a logging approach that most likely is not the intended way to log the packets.

Finally, we faced some setbacks related to writing to and reading of the same log file in real-time. We set up Logstash to read the current file Snort was writing to in order to get quick access to the logs in Kibana. However, perhaps due to the slow storage medium, many packets were not integrated into Elasticsearch by doing it this way. Instead, most of the packets with larger payloads were ignored by Logstash. This actually made the problem harder to detect as most of the smaller packets were read correctly (Snort probably managed to write them quickly enough). Thus, we decided to wait for Snort to log all the packets before integrating them into Elasticsearch. By doing so, from what we could observe, no packets were ignored by Logstash.

8.4 Ethical Issues

The main ethical concern throughout this thesis has been the importance of protecting user privacy. In our experiments we used our own data, generated on a closed network, meaning that we only sacrificed our own privacy during the data collection.

As for our proposed solution, both Princeton IoT Inspector and the cloud solution raise some privacy concerns. In the first case, Princeton IoT Inspector collects certain data about the smart home that is not encrypted: device manufacturers, DNS requests and responses, destination IP addresses and ports and the names of devices on the network. This data is collected from all IoT devices on the network currently being analysed. It is stored at the Department of Computer Science in Princeton University. While they claim that the data collected is confidential, sharing data is always a risk, though, no one but the research team has access to the data, and they are unable to infer what IoT devices any one user owns.

Similarly, in the case of the suggested third party cloud subscription service, the privacy concern is sharing the data with a third party. We believe that we applied sound security and privacy measures such as using https and certificates for validating authenticity, only sent the metadata without the actual payload as well as password protected the ability to upload to the AWS server. However, it is never without risk sharing data with another party.

8.5 Sustainability

The future smart home might have even more connected devices than we used in our test bed. This means that their individual power consumption needs to be considered. In a setup similar to ours, the calculation for their power consumption also needs to consider the software that is used to monitor their network activity.

Besides the individual power consumption for the private home, the growth of consumer IoT will also increase the need for backed services. This means large server banks that are constantly running, needing to be cooled. While we believe this development is inevitable given the rapid growth of internet-based services, having a third party cloud solution protect smart homes will certainly increase the need for cloud storage, which means even more data servers.

8.6 Future Work

In our work we found Princeton IoT Inspector to be useful for getting a bird's eye on the network activity. It is likely a suitable tool for an everyday user since most people will probably only worry about the current activities and want to get a general sense of how often a specific endpoint is contacted. Though, for a more in-depth view, it is somewhat lacking. If it is to be used in other research, it would benefit greatly from allowing a higher degree of customisability, for instance, by letting the user specify a time interval in all of the different visualisations. Moreover, supporting more operating systems such as Raspbian and Ubuntu would widen the user base, which in the end would make the software more usable for both research purposes and end-users.

Besides further development of software solutions directed at the end-user, it would also be interesting to see further research on initiatives such as the Manufacturer Usage Description mentioned in Chapter 2. It provides a way for devices to inform the network what sort of access and network functionality they need to properly function. This makes sure that the device is getting the appropriate access while allowing the network to have control over it. This makes the system more secure and bridges the gap between the manufacturer and the user. It also facilitates a level of trust and security that benefits network and security administrators.

Finally, in our thesis, we have focused on signature-based detection on the local network. However, given the trend of machine learning, we hope to see significant improvements for anomaly detection in the years to come. For instance, the cloud solution proposed in this thesis assumes there is an advantage of having a centralised solution with large computational capabilities that can make use of statistical approaches on a vast amount of user-generated metadata. Whether this is the future solution or not remains to be seen. Before then, further research into such a solution's capabilities would be beneficial.

CHAPTER 9

Conclusion

Consumer IoT is on the rise and is expected to constitute a significant share of the overall internet traffic the coming years. Considering the nature of these devices, that they are located in our homes and often contain microphones and/or cameras, as well as the fact that they tend to be produced by or dependent on infrastructure from some of the world's largest IT companies (which are usually present on other digital platforms as well), this raises privacy concerns.

Given this background, this thesis has investigated privacy in the smart home with a focus on solutions applicable at different levels (user, community and third parties). The aims of the thesis were designed to cover different aspects of privacy. For instance, we observed and compared traffic from devices from different regions, manufacturers and categories, aimed for different markets, to see if they behave differently from one another in regards to privacy. Moreover, we also explored different software solutions that can be used for privacy protection and evaluated their applicability for different contexts. Furthermore, we also looked into the feasibility of third-party developed rules and what role they could play for improving privacy protection for smart homes in the future.

In order to be meet these aims, a testbed supporting three different experiments was constructed. The results from the experiments were evaluated in relation to a threat model that was developed based on some of the most prominent privacy threats found in the literature.

The results from the device comparisons showed some concerning patterns. For example, there was a relatively strong concentration of geographical regions in which the traffic terminated. Furthermore, a high concentration of cloud providers was also observed. From this, we noticed that it is problematic to only apply a geographical perspective on the traffic since many of these companies have servers across the world. Similarly, only looking at destination domains might not either be sufficient as different regions might be subject to different legislation.

When it comes to tracking and use of encryption, we found that only a few devices were reported to contact known trackers or advertisement servers which was positive. However, most devices did send unencrypted traffic, although to a small extent.

Regarding the shape of the traffic, the results were mixed. Some device categories were clearly distinguishable from others, whereas others had a higher resemblance. We also observed some unexpected device behaviour, such as cameras contacting storage devices on private networks or home automation devices being surprisingly chatty.

As for the cloud solution, it proved satisfactory. However, the size of the logs quickly became an issue and if this is to be done on a commercial level, considerations need to be taken regarding the design to allow the rule developer to specify both which logs to ignore and which headers to include in the logs.

When it comes to the software solutions, both Princeton IoT Inspector and Snort in combination with the ELK Stack offered good visualisations to observe the traffic. However, the two solutions are very different. The former is much more user friendly and requires less know-how at the expense of customisability and expandability. For us, Snort and the ELK stack was the preferred choice, although for the regular smart home owner, Princeton IoT Inspector is most likely the better option.

This thesis has shown that smart home devices typically are talkative. The need for users to get a better understanding of what they are sharing and to whom they are doing so with will likely become increasingly important in the future. For this reason, we need to be able to better map the devices' behaviour. This requires collaboration on multiple levels. For instance, rule-based detection systems such as Snort is a good approach that can be extended with both community support and cloud rules. However, it is most likely not accessible to the wider audience. Other improvements would be enforcement of MUD as well as further development of software solutions such as Princeton IoT Inspector. Nevertheless, before one can fix the problem, one needs to understand it. As it stands today, the gap between what is being collected and what we know about it needs to narrow, and for that to happen we need further study of privacy threats to the smart home.

Bibliography

[1] M. H. Mazhar and Z. Shafiq. *Characterizing Smart Home IoT Traffic in the Wild*. 2020. arXiv: 2001.08288 [cs.NI].

[2] N. King. *Smart Home - A Definition.* https://www.housinglin.org.uk/_assets/Resources/Housing/Housing_advice/Smart_Home_-_A_definition_September_2003.pdf. Accessed: 2020-05-12.

[3] *List of Public Corporations by Market Capitalization.* https://en.wikipedia.org/wiki/List_of_public_corporations_by_market_capitalization. Accessed: 2020-02-27. Wikipedia.

[4] J. D. Rey. *Why Congress's Antitrust Investigation Should Make Big Tech Nervous.* https://www.vox.com/recode/2020/2/6/21125026/big-tech-congress-antitrust-investigation-amazon-apple-google-facebook. Accessed: 2020-02-27.

[5] B. Auxier, L. Rainie, M. Anderson, A. Perrin, M. Kumar, and E. Turner. *Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information.* https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/. Accessed: 2019-12-13.

[6] Ø. H. Kaldestad. *Fitness Wristbands Violate European Law.* https://www.forbrukerradet.no/siste-nytt/fitness-wristbands-violate-european-law. Accessed: 2019-12-13.

[7] H. Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Huang, N. Feamster, E. Felten, P. Mittal, and A. Narayanan. "Watching You Watch: The Tracking Ecosystem of Over-the-Top TV Streaming Devices". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* Nov. 2019, pages 131–147. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3354198.

[8] J. Ren, D. Dubois, D. Choffnes, A. Mandalari, R. Kolcun, and H. Haddadi. "Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach". In: *Proc. of the Internet Measurement Conference (IMC).* Oct. 2019, pages 267–279. ISBN: 978-1-4503-6948-0. DOI: 10.1145/3355369.3355577.

[9] I. Sanchez-Rola, M. Dell'Amico, P. Kotzias, D. Balzarotti, L. Bilge, P.-A. Vervier, and I. Santos. "Can I Opt Out Yet?: GDPR and the Global Illusion of Cookie Control". In: *Proceedings of the 2019 ACM Asia Conference on*

Computer and Communications Security. July 2019, pages 340–351. DOI: 10.1145/3321705.3329806.

[10] B. Wolford. *What is GDPR, the EU's New Data Protection Law?* https://gdpr.eu/what-is-gdpr/. Accessed: 2020-02-27.

[11] *What Does the General Data Protection Regulation (GDPR) govern?* https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-does-general-data-protection-regulation-gdpr-govern_en. Accessed: 2020-02-27. European Commission.

[12] *What is Personal Data?* https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_en. Accessed: 2020-02-27. European Commission.

[13] *GDPR in Numbers*. https://ec.europa.eu/info/sites/info/files/infographic-gdpr_in_numbers.pdf. Accessed: 2020-02-27. European Commission.

[14] *Facebook–Cambridge Analytica Data Scandal*. https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal. Accessed: 2020-05-30. Wikipedia.

[15] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric. "All Things Considered: An Analysis of IoT Devices on Home Networks". In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pages 1169–1185. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak.

[16] *Pi-hole Documentation*. https://docs.pi-hole.net/. Accessed: 2020-05-23. Pi-hole LLC.

[17] E. Lear, R. Droms, and D. Romascanu. *Manufacturer Usage Description Specification*. RFC 8520. Mar. 2019. DOI: 10.17487/RFC8520. URL: https://rfc-editor.org/rfc/rfc8520.txt.

[18] D. Huang Y., D. Kumar, N. Apthorpe, G. Acar, F. Li, A. Narayanan, and N. Feamster. *What is Princeton IoT Inspector?* https://iot-inspector.princeton.edu/. Accessed: 2020-03-01.

[19] D. Huang Y., D. Kumar, N. Apthorpe, G. Acar, F. Li, A. Narayanan, and N. Feamster. *Frequently Asked Questions*. https://iotinspector.org/blog/post/faq/. Accessed: 2020-05-31.

[20] N. Feamster and D. Y. Huang. *IoT Inspector*. https://iotinspector.org/blog/post/contact-us/. Accessed: 2020-05-12.

[21] M. Roesch. *Snort*. https://www.snort.org/documents. Accessed: 2020-05-12.

[22] P. Cardoso, J. Monteiro, J. Semião, and J. Rodrigues. *Harnessing the Internet of Everything (IoE) for Accelerated Innovation Opportunities*. Advances in Computer and Electrical Engineering (2327-039X). IGI Global, 2019. ISBN: 9781522573333. URL: https://books.google.se/books?id=fTyEDwAAQBAJ.

[23] J. Bugeja, A. Jacobsson, and P. Davidsson. "On Privacy and Security Challenges in Smart Connected Homes". In: *2016 European Intelligence and Security Informatics Conference (EISIC)*. 2016, pages 172–175.

[24] P. Rysavy. *Low Versus High Radio Spectrum.* https://hightechforum.org/low-versus-high-radio-spectrum/. Accessed: 2020-05-12.

[25] *Zigbee.* https://zigbeealliance.org/solution/zigbee/. Accessed: 2020-05-12. Zigbee Alliance.

[26] *Z Wave.* https://www.z-wave.com/learn. Accessed: 2020-05-12. Silicon Labs.

[27] Bughunter. *InternetProtocolStack.png.* https://commons.wikimedia.org/wiki/File:InternetProtocolStack.png. Accessed: 2020-05-30.

[28] N. King. *The Domain Name Industry Brief.* https://www.verisign.com/assets/domain-name-report-Q32019.pdf. Accessed: 2020-05-12.

[29] F. Gont. *Network and Trust.* https://www.internetsociety.org/resources/deploy360/dns-privacy/intro/. Accessed: 2020-05-12.

[30] *Amazon EC2.* https://aws.amazon.com/ec2/. Accessed: 2020-06-17. Amazon Web Services, Inc.

[31] *Cloud Computing Services.* https://azure.microsoft.com/en-us/. Accessed: 2020-05-12. Microsoft.

[32] S. Watts and M. Raza. *SaaS vs PaaS vs IaaS: What's The Difference and How To Choose.* https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/. Accessed: 2020-05-12.

[33] B. Turner. *What is SaaS?* https://www.techradar.com/news/what-is-saas. Accessed: 2020-05-12.

[34] B. Butler. *PaaS Primer: What is Platform as a Service and Why Does It Matter?* https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html. Accessed: 2020-05-12.

[35] *What is IaaS?* https://azure.microsoft.com/en-us/overview/what-is-iaas/. Accessed: 2020-05-12. Microsoft Team.

[36] *Wireshark.* https://www.wireshark.org/. Accessed: 2020-05-12. Wireshark Foundation.

[37] *Promiscuous Monitoring in Ethernet and Wi-Fi Networks.* https://www.tamos.com/docs/monitoring.pdf/. Accessed: 2020-05-21. TamoSoft.

[38] G. Harris. *Ethernet capture setup.* https://wiki.wireshark.org/CaptureSetup/Ethernet/. Accessed: 2020-05-21.

[39] *ARP Spoofing.* https://www.imperva.com/learn/application-security/arp-spoofing/. Accessed: 2020-05-30. Imperva.

[40] *About Suricata.* https://suricata-ids.org/about/. Accessed: 2020-08-17. Open Information Security Foundation.

[41] *tcpdump & libcap.* https://www.tcpdump.org/. Accessed: 2020-05-24. The Tcpdump Group.

[42] *Splunk.* https://www.splunk.com/. Accessed: 2020-05-25. Splunk Inc.

[43] D. Berman. *Kibana Tutorial.* https://logz.io/blog/kibana-tutorial/. Accessed: 2020-05-12.

[44] S. Cooper. *9 best Network-based Intrusion Detection Systems (NIDS) tools.* https://www.comparitech.com/net-admin/nids-tools-software/. Accessed: 2020-05-22.

[45] *What is the Relationship Between Snort and Cisco?* https://www.snort.org/faq/what-is-the-relationship-between-snort-and-cisco. Accessed: 2020-05-12. The Snort Team.

[46] *What are Community Rules?* https://www.snort.org/faq/what-are-community-rules. Accessed: 2020-05-12. The Snort Team.

[47] *What is a Snort rule?* https://www.snort.org/faq/what-is-a-snort-rulet. Accessed: 2020-05-12. The Snort Team.

[48] *What are Snort Subscriber Rule Set?* https://www.snort.org/faq/what-are-snort-subscriber-rule-set. Accessed: 2020-05-12. The Snort Team.

[49] *What are Community Rules?* https://www.snort.org/faq/what-are-community-rules. Accessed: 2020-05-29. The Snort Team.

[50] *Snort Manual.* https://usermanual.wiki/Pdf/snortmanual.1346323497/help. Accessed: 2020-05-12. The Snort Team.

[51] S. Bocetta. *5 Useful Open Source Log Analysis Tools.* https://opensource.com/article/19/4/log-analysis-tools. Accessed: 2020-05-22.

[52] B. Jackson. *Top 10+ Log Analysis Tools - Making Data-Driven Decisions.* https://www.keycdn.com/blog/log-analysis-tools. Accessed: 2020-05-22.

[53] R. Combs. *Snort 3.0 with ElasticSearch, LogStash, and Kibana (ELK).* https://blog.snort.org/2017/11/snort-30-with-elasticsearch-logstash.html. Accessed: 2020-05-22.

[54] D. Berman. *Elasticsearch Tutorial.* https://logz.io/blog/elasticsearch-tutorial/. Accessed: 2020-05-12.

[55] D. Berman. *Logstash Tutorial.* https://logz.io/blog/logstash-tutorial/. Accessed: 2020-05-12.

[56] *DNS Filter Plugin.* https://www.elastic.co/guide/en/logstash/current/plugins-filters-dns.html. Accessed: 2020-05-27. Elasticsearch B.V.

[57] *Geoip Filter Plugin.* https://www.elastic.co/guide/en/logstash/current/plugins-filters-geoip.html. Accessed: 2020-05-27. Elasticsearch B.V.

[58] *Google.* https://en.wikipedia.org/wiki/Google. Accessed: 2020-06-18. Wikipedia.

[59] *Dublin, Ireland.* https://www.google.com/about/datacenters/locations/dublin/. Accessed: 2020-06-21. Google LLC.

[60] *What is 1e100.net?* https://support.google.com/faqs/answer/174717?hl=en. Accessed: 2020-06-18. Google.

[61] *AWS service endpoints.* https://docs.aws.amazon.com/general/latest/gr/rande.html. Accessed: 2020-06-18. Amazon Web Services, Inc.

[62] *IP Address Lookup.* https://www.whatismyip.com/ip-address-lookup/. Accessed: 2020-07-07. WhatIsMyIP.com.

[63] *Alibaba Cloud.* https://en.wikipedia.org/wiki/Alibaba_Cloud. Accessed: 2020-07-08. Wikipedia.

[64] *What Are HUAWEI CLOUD DNS Servers?* https://support.huaweicloud.com/en-us/dns_faq/dns_faq_012.html. Accessed: 2020-06-18. Huawei Cloud.

APPENDIX A

Device Lists

A.1 Brand Segments

A.1.1 Devices in the Well-known Brands Segment

A.1.2 Devices in the Less-known Brands Segment

Categories / Segment	Streaming devices	IP-Cameras	Home automation	Smart speakers
EU: Well-known Brand	Google Chromecast (gen 2) AppleTv (gen 5)	D-Link DCS-932L (H/W v.B2)	IKEA "Tråfri" Gateway (type:E1526) IKEA "Trådfri" On/Off Switch (type:E1743) IKEA "Trådfri" Control Outlet (type:E1603) TP-Link Smart WiFi Plug (Model HS100, v.4.0) Samsung SmartThings Hub (M/N:IM6001-V3P22) Samsung SmartThings Multipurpose Sensor (M/N:STS-MLT-250)	Google Home Mini (Model: H0A)

Table A.1: The devices that were used in the well-known brands segment and the device categories they belonged to.

Segments \ Categories	Streaming devices	IP-Cameras	Home automation	Smart speakers
EU: less-known Brand	-	P-302R (Model:OEM087-EU)	Clas Ohlson WiFi Smart Plug (Model:SP3 EU)	-

Table A.2: The devices that were used in the less-known brands segment and the device categories they belonged to.

A.2 Region Segments

A.2.1 Devices in the EU Segment

A.2.2 Devices in the North American Segment

A.2.3 Devices in the Asian Segment

Categories / Segments	Streaming devices	IP-Cameras	Home automation	Smart speakers
EU	Google Chromecast (gen 2, Model: NC2-6A5) AppleTv (gen 5, Model: A1842)	D-Link DCS-932L (H/W v.B2) P-302R (Model:OEM087-EU)	IKEA "Tråfri" Gateway (type:E1526) IKEA "Trådfri" On/Off Switch (type:E1743) IKEA "Trådfri" Control Outlet (type:E1603) TP-Link Smart WiFi Plug (Model HS100, v.4.0) Samsung SmartThings Hub (M/N:IM6001-V3P22) Samsung SmartThings Multipurpose Sensor (M/N:STS-MLT-250) Clas Ohlson WiFi Smart Plug (Model:SP3 EU)	Google Home Mini (Model: H0A)

Table A.3: The devices that were used in the EU segment and the device categories they belonged to.

Categories / Segments	Streaming devices	IP-Cameras	Home automation	Smart speakers
North America	Google Chromecast Ultra (Model: NC2-6A5-D)	-	-	Google Home Mini (Model: H0A) Amazon Echo Dot (gen 2, Model: RS03QR

Table A.4: The devices that were used in the North American segment and the device categories they belonged to.

Categories / Segments	Streaming devices	IP-Cameras	Home automation	Smart speakers
Asia	-	USAFEQLO (Model: USAN701F-100W) ESCAM (Model: Q6)	Wi-Fi Smart Socket (Model: SWA11)	-

Table A.5: The devices that were used in the Asian segment and the device categories they belonged to.

A.3 Auxiliary Equipment

The following auxiliary equipment was used in the testbed:

I. A computer running Ubuntu 18.04 with:

   a) 4 GB DDR3 RAM (@800MT/s)

   b) Intel(R) Core™2 Duo CPU E8500 @3.16GHz

   c) x3 wired NICs (x2 for inline configuration, x1 for (remote) management)

   d) x2 of the NICs supported promiscuous mode (needed for being able to read packets destined for other machines)

II. TP-Link Archer MR200 (EU) v.4.0 (a 4G router that is able to reserve IP-addresses as part of its DHCP service)

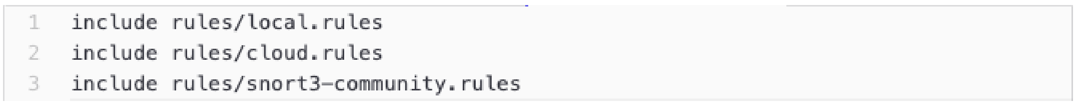
III. D-Link Dir-845L (H/W version A1) configured in "access point mode" (i.e. DHCP turned off and WAN port not used)

IV. A Windows computer for Princeton IoT Inspector (in spring 2020)

APPENDIX B

Snort Rules

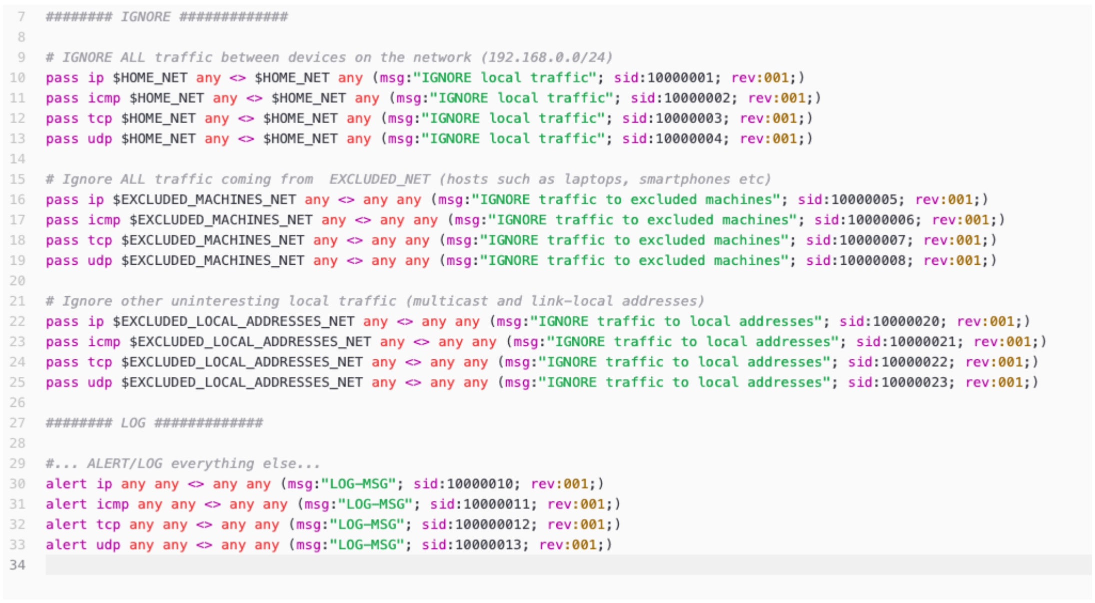
B.1 Included Rule Files



```
include rules/local.rules
include rules/cloud.rules
include rules/snort3-community.rules
```

Figure 1: the ips.include file specifies which rules files to include. Both local, community and cloud rules were used.

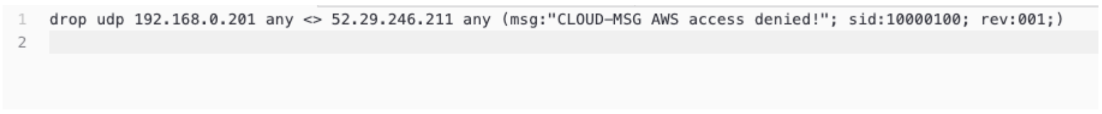
B.2 Local Rules



```
######## IGNORE #############

# IGNORE ALL traffic between devices on the network (192.168.0.0/24)
pass ip $HOME_NET any <> $HOME_NET any (msg:"IGNORE local traffic"; sid:10000001; rev:001;)
pass icmp $HOME_NET any <> $HOME_NET any (msg:"IGNORE local traffic"; sid:10000002; rev:001;)
pass tcp $HOME_NET any <> $HOME_NET any (msg:"IGNORE local traffic"; sid:10000003; rev:001;)
pass udp $HOME_NET any <> $HOME_NET any (msg:"IGNORE local traffic"; sid:10000004; rev:001;)

# Ignore ALL traffic coming from  EXCLUDED_NET (hosts such as laptops, smartphones etc)
pass ip $EXCLUDED_MACHINES_NET any <> any any (msg:"IGNORE traffic to excluded machines"; sid:10000005; rev:001;)
pass icmp $EXCLUDED_MACHINES_NET any <> any any (msg:"IGNORE traffic to excluded machines"; sid:10000006; rev:001;)
pass tcp $EXCLUDED_MACHINES_NET any <> any any (msg:"IGNORE traffic to excluded machines"; sid:10000007; rev:001;)
pass udp $EXCLUDED_MACHINES_NET any <> any any (msg:"IGNORE traffic to excluded machines"; sid:10000008; rev:001;)

# Ignore other uninteresting local traffic (multicast and link-local addresses)
pass ip $EXCLUDED_LOCAL_ADDRESSES_NET any <> any any (msg:"IGNORE traffic to local addresses"; sid:10000020; rev:001;)
pass icmp $EXCLUDED_LOCAL_ADDRESSES_NET any <> any any (msg:"IGNORE traffic to local addresses"; sid:10000021; rev:001;)
pass tcp $EXCLUDED_LOCAL_ADDRESSES_NET any <> any any (msg:"IGNORE traffic to local addresses"; sid:10000022; rev:001;)
pass udp $EXCLUDED_LOCAL_ADDRESSES_NET any <> any any (msg:"IGNORE traffic to local addresses"; sid:10000023; rev:001;)

######## LOG #############

#... ALERT/LOG everything else...
alert ip any any <> any any (msg:"LOG-MSG"; sid:10000010; rev:001;)
alert icmp any any <> any any (msg:"LOG-MSG"; sid:10000011; rev:001;)
alert tcp any any <> any any (msg:"LOG-MSG"; sid:100000012; rev:001;)
alert udp any any <> any any (msg:"LOG-MSG"; sid:10000013; rev:001;)

```

Figure 2: The local rules used.

B.3 Cloud Rules



```
drop udp 192.168.0.201 any <> 52.29.246.211 any (msg:"CLOUD-MSG AWS access denied!"; sid:10000100; rev:001;)
```

Figure 3: The cloud rules used.

APPENDIX C

Python Scripts

C.1 Log Parser: parser.py

```
##### IMPORTS/EXTERNAL LIBS #####
import glob
import json
import os
from timestamper import getTimestamp
#################################


# Message to include in the timestamp
MSG = 'LOG PARSED @'

# Set the prefix of the output files
output_file_prefix = "/home/nylundj/pythonIoT/src/parsed_logs/parsed_log."

# Path to the logs
logs_path = "/var/log/snort/"

# Get the array of logs
logs_list = os.listdir(logs_path)

### CASE SEPCIFIC ###
logs_list.remove('alert_json.txt') # current snort file
logs_list.remove('appid_stats.log') # application statistics
####################

# ... and sort it by name
logs_list.sort()
```

```
# Get the state information about the parsing last time
with open('/home/nylundj/pythonIoT/src/parser_state.txt', 'r+') as state:

    # Fetch the state data
    state_info = state.read().splitlines()

    # Get the last log that was processed and name of the number of
    # the latest parsed log
    latest_processed_file       = state_info[0]
    latest_parsed_log_counter   = int(state_info[1])
    nmbr_of_ignored_packets     = 0 # information purposes

    # Three cases:
    # a) First parsing ever
    # b) New logs available
    # c) No new logs available
    if latest_processed_file == 'null':
        current_index               = 0
        current_file                = logs_list[current_index]
        current_parsed_log_counter  = latest_parsed_log_counter
    elif logs_list.index(latest_processed_file)+1 < len(logs_list):
        latest_proccessed_index     = logs_list.index(latest_processed_file)
        current_index               = latest_proccessed_index +1
        current_file                = logs_list[current_index]
        current_parsed_log_counter  = latest_parsed_log_counter + 1
    else:
        print('There are no logs to process. Quitting')
        state.close()
        quit()

    # While there are still logs to process...
    while current_index < len(logs_list) :

        # Update the paths
        parsed_log_output_path = (output_file_prefix +
        str(current_parsed_log_counter))
        current_file_path      = logs_path + current_file

        # Filter out the b64_data field (i.e. payload) from the Snort logs
        with open(parsed_log_output_path + '.txt', 'w') as destination_file:
            with open(current_file_path, 'r') as source_file:
                # In case of failure, keep a counter to easily track the
                # row (packet) that caused the issue
                row = 0
                for line in source_file:
```

```
                row = row +1
                try:
                    fields = json.loads(line.strip())
                    if 'b64_data' in fields:
                        del fields['b64_data']
                    # Write it to destination_file
                    destination_file.write(json.dumps(fields) + "\n")
                except Exception as e:
                    # Log the failed attempt
                    nmbr_of_ignored_packets = nmbr_of_ignored_packets +1

                    with open('/home/nylundj/pythonIoT/src/failed_logs.txt',
                    'a') as failed_logs_file:
                        failed_log_info = (current_file +
                        ' [' + str(row) + ']' + '\n')
                        error_timestamp = getTimestamp(True, True, False,
                        'Failed @')

                        failed_logs_file.write(failed_log_info)
                        failed_logs_file.write(error_timestamp)
                        failed_logs_file.close()

            source_file.close()

        # Finally, add a timestamp to the parsed log
        timestamp = getTimestamp(True, False, False, MSG)
        spaced_timestamp = '\n' + '\n' + timestamp
        destination_file.write(spaced_timestamp)
        destination_file.close()

    # Move on to next log file
    current_index = current_index + 1

    # Check if any more log files exist. If so, update the state
    if (current_index < len(logs_list)):
        current_file = logs_list[current_index]
        current_parsed_log_counter = current_parsed_log_counter + 1
    else:
        print(('SUCCESS: All logs have been parsed! Number of ignored ' +
        'packets: ' + str(nmbr_of_ignored_packets)))

# Save the state info
new_state = current_file + "\n" + str(current_parsed_log_counter)
state.seek(0)
state.write(new_state)
state.truncate()
```

```
    state.close()
```

C.2 Log Shipper: shipper.py

```
##### IMPORTS/EXTERNAL LIBS #####
import glob
import os
import requests
from timestamper import getTimestamp
#################################


# Message to include in the timestamp
MSG = 'LOG SHIPPED @'

# Keep track if the log was successfully shipped
success = False

# Path to the logs
PARSED_LOGS_PATH = "/home/nylundj/pythonIoT/src/parsed_logs/"

# Url to the AWS server
AWS_SERVER_URL = "https://aws.webredirect.org"

# Credentials
data = {
    'username': 'master.iot.thesis@gmail.com',
    'password': 'serverawspass'
}

# Get the array of logs and sort it by name
parsed_logs_list = os.listdir(PARSED_LOGS_PATH)
parsed_logs_list.sort()

# Create a counter that keeps track of the log index to process
ith_log = 0

# Maximum number of retries/attempts
MAX_ATTEMPTS = 3

# Check if there is no logs to process. If not, break
if len(parsed_logs_list) == 0:
    print('There are no logs to process. Quitting')
    quit()

# Else, while there are logs to ship...
```

```
while parsed_logs_list[ith_log]:

    # Get the ith parsed log and its path
    current_log          = parsed_logs_list[ith_log]
    current_log_path     = PARSED_LOGS_PATH + current_log

    # Set number of attempts
    nmbr_of_attempts     = 0 # none as of yet

    # Append a timestamp of when the log was shipped
    with open(current_log_path, 'a+') as current_log_file:
        timestamp = getTimestamp(False, False, False, MSG)
        current_log_file.write(timestamp)
        current_log_file.close()

    # Ship the log
    with open(current_log_path, 'rb') as current_log_file:

        # For as long as we have not exceeded MAX_ATTEMPTS...
        while nmbr_of_attempts < MAX_ATTEMPTS:

            # New attempt - increase the counter
            nmbr_of_attempts = nmbr_of_attempts + 1

            # Make a HTTP POST request to the web server and catch the response
            response = requests.post(AWS_SERVER_URL, files={
            'file':current_log_file}, data=data)

            # Definition of an OK HTTP POST response
            ok_reponse = (response.status_code == 200 or
            response.status_code == 201)

            # Delete the log if succssessful POST
            if(ok_reponse):
                os.remove(current_log_path)
                success = True
                break

        # We are finished with the file. Close it.
        current_log_file.close()

    # Error check
    # (We have failed to ship the log. Remove the timestamp and stop the
    # shipping of further logs)
    if not success:
        # Open it once more to get the data
        with open(current_log_path, 'r+') as failed_log_file:
            failed_log_rows = failed_log_file.read().splitlines()
            failed_log_file.close()

        # Parse out the timestamp
        data_without_timestamp = "\n".join(failed_log_rows[:-2])

        # Overwrite the old file
        with open(current_log_path, 'w+') as failed_log_file:
            for row in range(len(data_without_timestamp)):
                failed_log_file.write(data_without_timestamp[row])
            failed_log_file.write('\n')
            failed_log_file.close()

        # Inform the user about the issue and exit
        print('FAILURE: Not all of the available logs were shipped.' +
        ' Latest status code was: ' + str(response.status_code) +
        '. Try again later')
        quit()

    # Else move on to the next log
    ith_log = ith_log + 1

    # If there is no next log: exit gracefully
    if(ith_log == len(parsed_logs_list)):
        print('SUCCESS: All logs have been successfully shipped.')
        break
```

C.3 Cloud Rules Fetcher: fetcher.py

```
##### IMPORTS/EXTERNAL LIBS #####
import glob
import requests
from timestamper import getTimestamp
#################################


# Define the path to the rule file (output path)
CLOUD_RULES_DST_PATH = '/usr/local/etc/snort/rules/cloud.rules'

# Message to include in the timestamp
MSG = 'RULES FETCHED @'

# Url to the AWS server
AWS_SERVER_URL = "https://aws.webredirect.org/cloud.rules"
```

```
# Define maximum number of retries/attempts
MAX_ATTEMPTS = 3

# Initial number of tries/attempts
nmbr_of_attempts = 0

# Fetch the updated cloud rules.
# (For as long as we have not exceeded MAX_ATTEMPTS...)
while nmbr_of_attempts < MAX_ATTEMPTS:

    # New attempt - increase the counter
    nmbr_of_attempts = nmbr_of_attempts + 1

    # Make a HTTP GET request to the AWS server and catch the response
    response = requests.get(AWS_SERVER_URL)

    # Definition of an OK HTTP GET response
    ok_reponse = (response.status_code == 200 or response.status_code == 201)

    # Break the loop if successful GET
    if(ok_reponse):

        # Update the old rules
        with open(CLOUD_RULES_DST_PATH, 'wb') as old_cloud_rules:
            old_cloud_rules.seek(0)
            old_cloud_rules.write(response.content)
        old_cloud_rules.close()

        # Add a timestamp of when the new/updated rules were fetched
        with open(CLOUD_RULES_DST_PATH, 'a') as new_cloud_rules_file:

            timestamp = getTimestamp(True, False, True, MSG)
            spaced_timestamp = '\n' + '\n' + timestamp
            new_cloud_rules_file.write(spaced_timestamp)

        # We are finished with the new_cloud_rules_file. Close it
        new_cloud_rules_file.close()

        # Inform the user that the about the update
        print('SUCCESS: The rules have been successfully updated.')

        # We are finished. Break the loop
        break

    # Graceful exit in case repeated failures
    if nmbr_of_attempts == MAX_ATTEMPTS:
        error_message = ('Reached maximum number of attempts without success.' +
        ' Quitting')
        print(error_message)
        break
```

C.4 Timestamper: timestamper.py

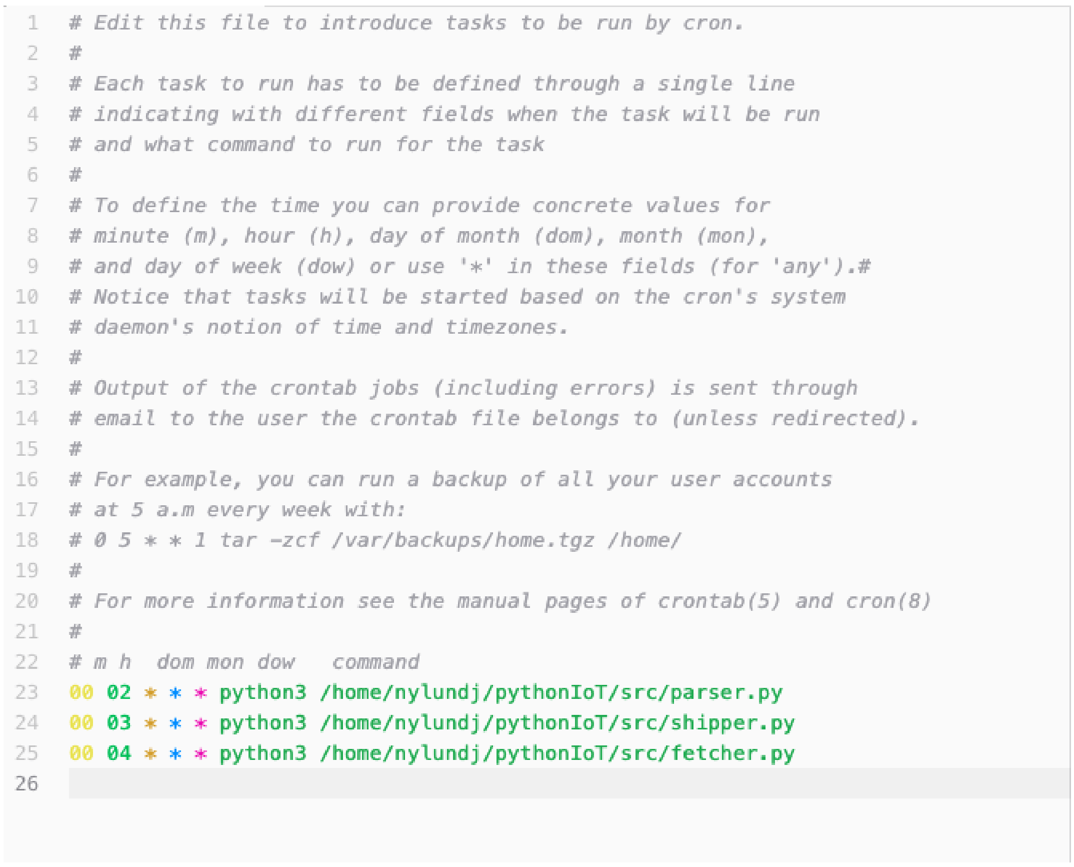
```
##### IMPORTS/EXTERNAL LIBS #####
from datetime import datetime
##################################


# A method for obtaining a timestamp in a format specifiec by the arguments
def getTimestamp(top_delimiter, spacing, isSnortRules, msg):

    # Timestamps in Snort cloud rules need to be commented out.
    # Adapt the syntax
    if isSnortRules:
        snort_adjustment = '#'
    else:
        snort_adjustment = ''

    # Get current date and time from datetime
    current_date_and_time = datetime.now()

    # Convert the date and time on the format: dd/mm/YY H:M:S
    datetime_string = current_date_and_time.strftime("%d/%m/%Y %H:%M:%S")

    # Determine the requested output format of the timestamp
    if top_delimiter and spacing:
        timestamp = (
        snort_adjustment + '****************************************' + '\n' +
        snort_adjustment + msg + ": " + datetime_string + '\n' +
        snort_adjustment + '****************************************' + '\n' +
        '\n')
    elif top_delimiter and not spacing:
        timestamp = (
        snort_adjustment + '****************************************' + '\n' +
        snort_adjustment + msg + ": " + datetime_string + '\n' +
        snort_adjustment + '****************************************' + '\n')
    elif not top_delimiter and spacing:
        timestamp =  (
        snort_adjustment + msg + ": " + datetime_string + '\n' +
        snort_adjustment + '****************************************' + '\n' +
        '\n')
```

```
elif not top_delimiter and not spacing:
    timestamp =  (
    snort_adjustment + msg + ": " + datetime_string + '\n' +
    snort_adjustment + '****************************************' + '\n')

return timestamp
```

APPENDIX D

Cron Jobs

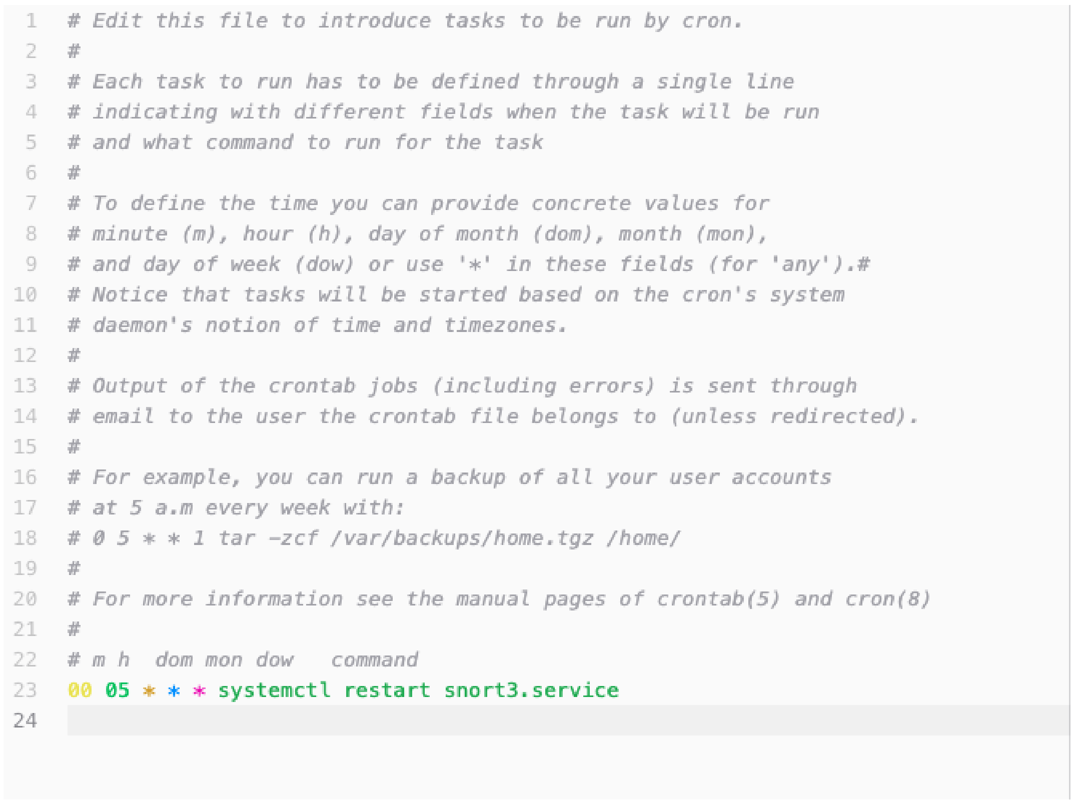
D.1 User Cron Jobs



```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
00 02 * * * python3 /home/nylundj/pythonIoT/src/parser.py
00 03 * * * python3 /home/nylundj/pythonIoT/src/shipper.py
00 04 * * * python3 /home/nylundj/pythonIoT/src/fetcher.py

```

Figure 1: The cron jobs used that did not require root privileges.

D.2 Root Cron Jobs



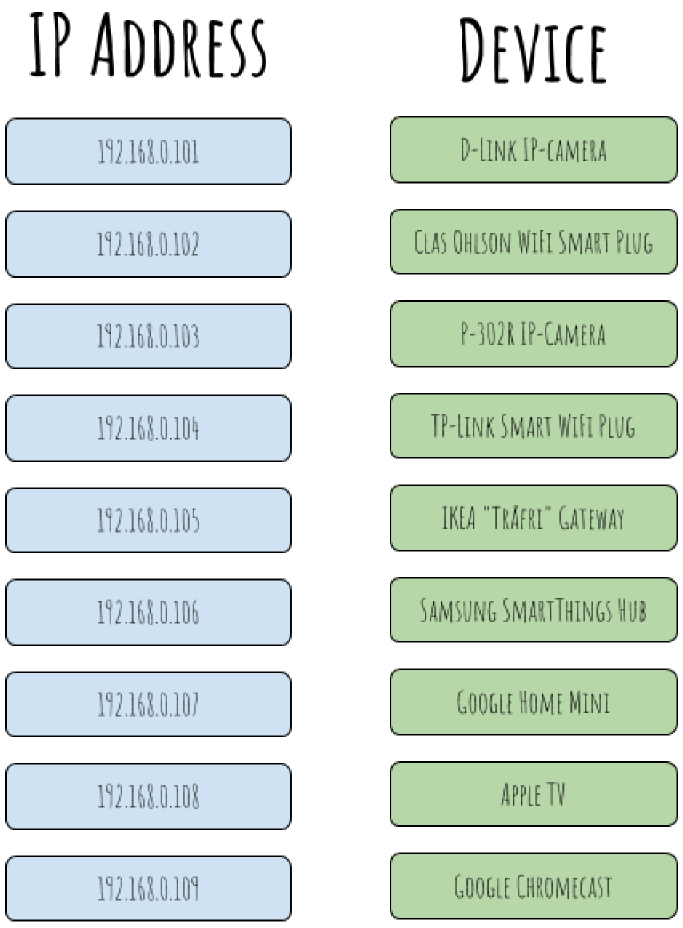
```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
00 05 * * * systemctl restart snort3.service

```

Figure 2: The cron jobs used that required root privileges.

APPENDIX E

IP Addressing

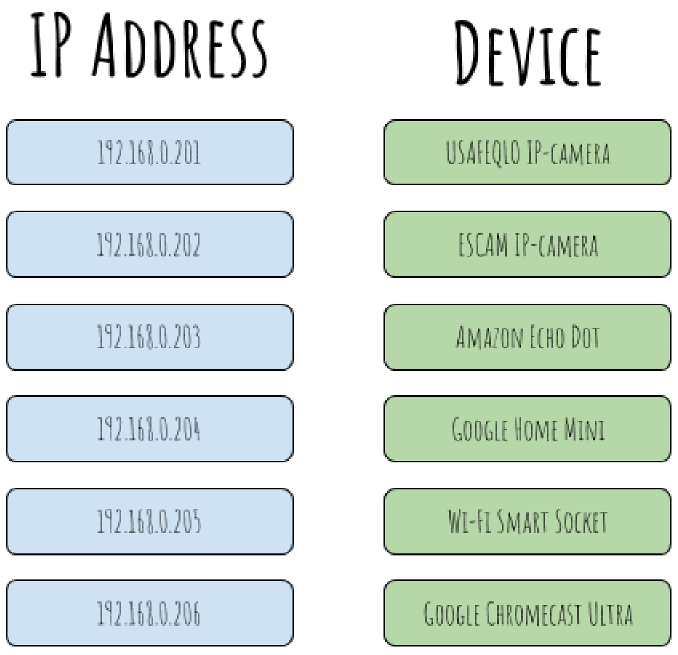
E.1 EU Segment



IP Address	Device
192.168.0.101	D-Link IP-camera
192.168.0.102	Clas Ohlson WiFi Smart Plug
192.168.0.103	P-302R IP-Camera
192.168.0.104	TP-Link Smart WiFi Plug
192.168.0.105	IKEA "TRÅFRI" Gateway
192.168.0.106	Samsung SmartThings Hub
192.168.0.107	Google Home Mini
192.168.0.108	Apple TV
192.168.0.109	Google Chromecast

Figure 1: The reserved IP addresses of the devices in the EU segment.

E.2 North American and Asian Segments



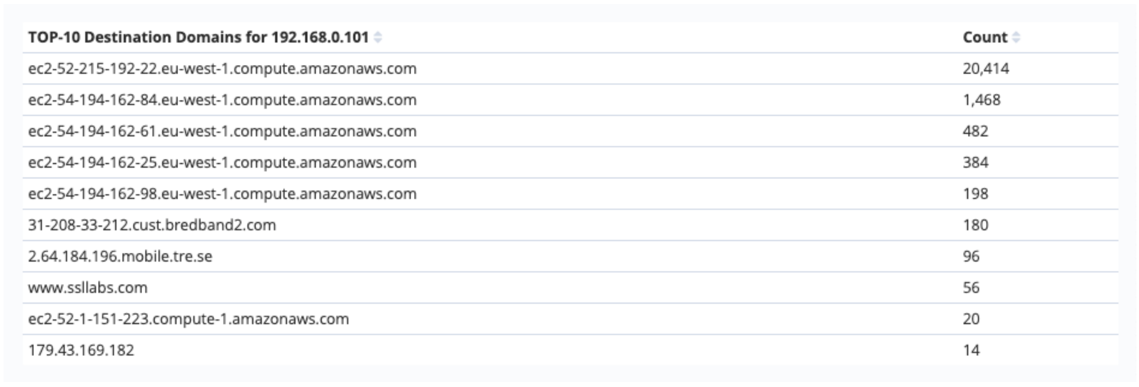
IP Address	Device
192.168.0.201	USAFEQLO IP-camera
192.168.0.202	ESCAM IP-camera
192.168.0.203	Amazon Echo Dot
192.168.0.204	Google Home Mini
192.168.0.205	Wi-Fi Smart Socket
192.168.0.206	Google Chromecast Ultra

Figure 2: The reserved IP addresses of the devices in the North American and Asian segments.

APPENDIX F

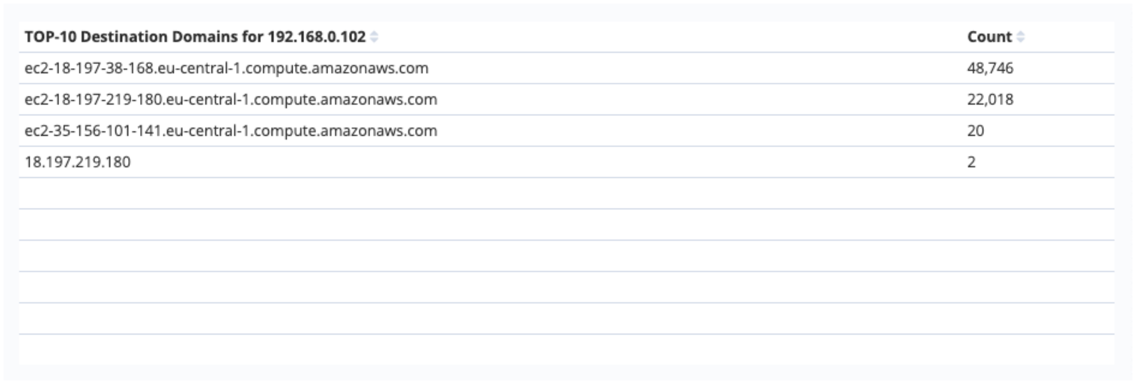
Top Endpoints

F.1 EU Segment: Top Ten Destination Domains per Device



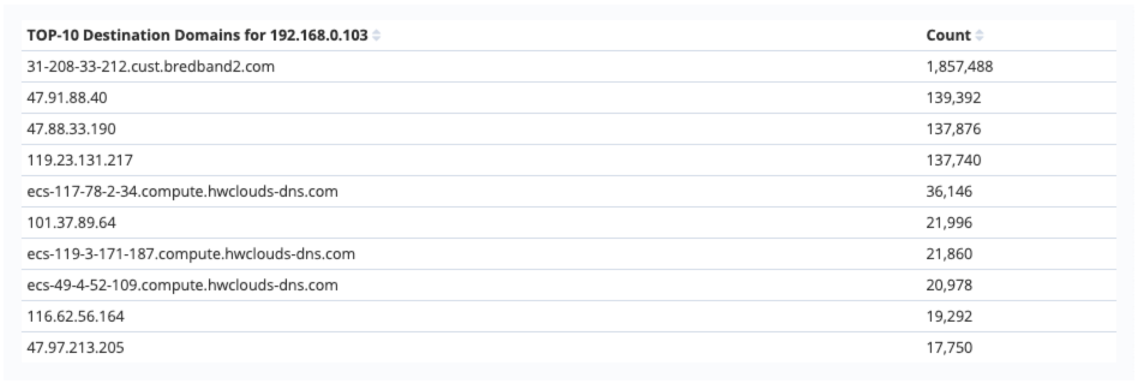
TOP-10 Destination Domains for 192.168.0.101	Count
ec2-52-215-192-22.eu-west-1.compute.amazonaws.com	20,414
ec2-54-194-162-84.eu-west-1.compute.amazonaws.com	1,468
ec2-54-194-162-61.eu-west-1.compute.amazonaws.com	482
ec2-54-194-162-25.eu-west-1.compute.amazonaws.com	384
ec2-54-194-162-98.eu-west-1.compute.amazonaws.com	198
31-208-33-212.cust.bredband2.com	180
2.64.184.196.mobile.tre.se	96
www.ssllabs.com	56
ec2-52-1-151-223.compute-1.amazonaws.com	20
179.43.169.182	14

Figure 1: The top ten most contacted domains for 192.168.0.101.



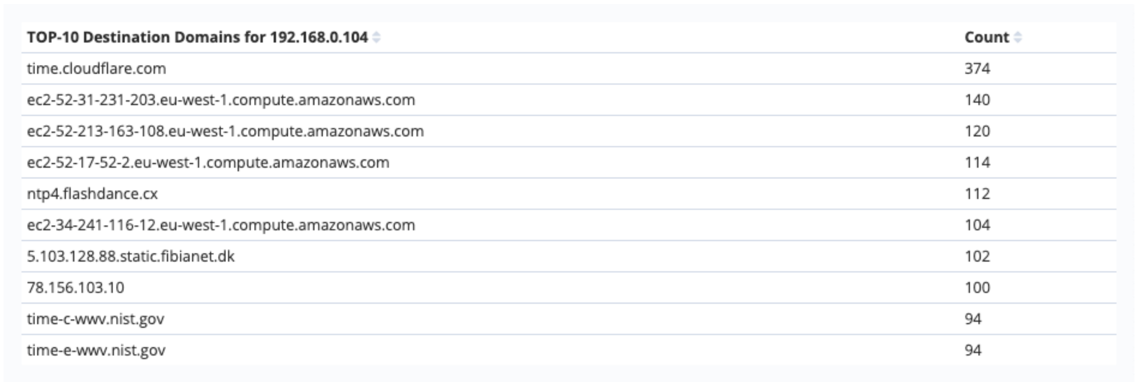
TOP-10 Destination Domains for 192.168.0.102	Count
ec2-18-197-38-168.eu-central-1.compute.amazonaws.com	48,746
ec2-18-197-219-180.eu-central-1.compute.amazonaws.com	22,018
ec2-35-156-101-141.eu-central-1.compute.amazonaws.com	20
18.197.219.180	2

Figure 2: The top ten most contacted domains for 192.168.0.102.



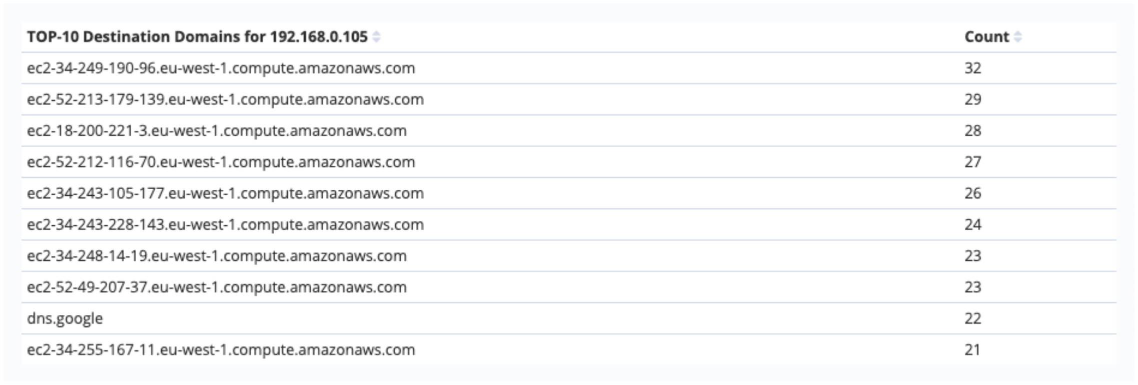
TOP-10 Destination Domains for 192.168.0.103	Count
31-208-33-212.cust.bredband2.com	1,857,488
47.91.88.40	139,392
47.88.33.190	137,876
119.23.131.217	137,740
ecs-117-78-2-34.compute.hwclouds-dns.com	36,146
101.37.89.64	21,996
ecs-119-3-171-187.compute.hwclouds-dns.com	21,860
ecs-49-4-52-109.compute.hwclouds-dns.com	20,978
116.62.56.164	19,292
47.97.213.205	17,750

Figure 3: The top ten most contacted domains for 192.168.0.103.



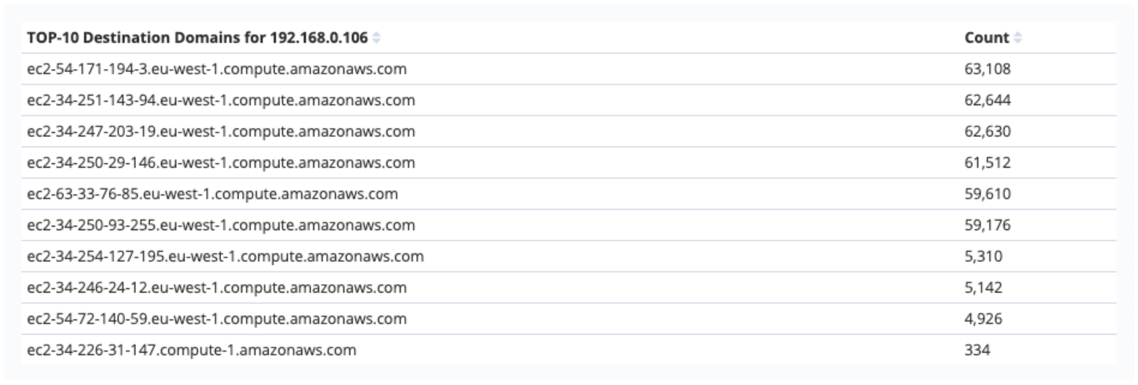
TOP-10 Destination Domains for 192.168.0.104	Count
time.cloudflare.com	374
ec2-52-31-231-203.eu-west-1.compute.amazonaws.com	140
ec2-52-213-163-108.eu-west-1.compute.amazonaws.com	120
ec2-52-17-52-2.eu-west-1.compute.amazonaws.com	114
ntp4.flashdance.cx	112
ec2-34-241-116-12.eu-west-1.compute.amazonaws.com	104
5.103.128.88.static.fibianet.dk	102
78.156.103.10	100
time-c-wwv.nist.gov	94
time-e-wwv.nist.gov	94

Figure 4: The top ten most contacted domains for 192.168.0.104.



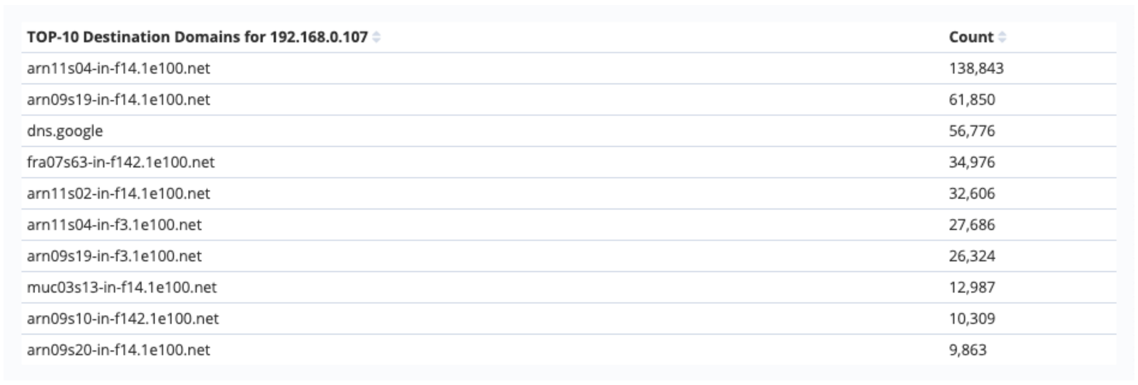
TOP-10 Destination Domains for 192.168.0.105	Count
ec2-34-249-190-96.eu-west-1.compute.amazonaws.com	32
ec2-52-213-179-139.eu-west-1.compute.amazonaws.com	29
ec2-18-200-221-3.eu-west-1.compute.amazonaws.com	28
ec2-52-212-116-70.eu-west-1.compute.amazonaws.com	27
ec2-34-243-105-177.eu-west-1.compute.amazonaws.com	26
ec2-34-243-228-143.eu-west-1.compute.amazonaws.com	24
ec2-34-248-14-19.eu-west-1.compute.amazonaws.com	23
ec2-52-49-207-37.eu-west-1.compute.amazonaws.com	23
dns.google	22
ec2-34-255-167-11.eu-west-1.compute.amazonaws.com	21

Figure 5: The top ten most contacted domains for 192.168.0.105.



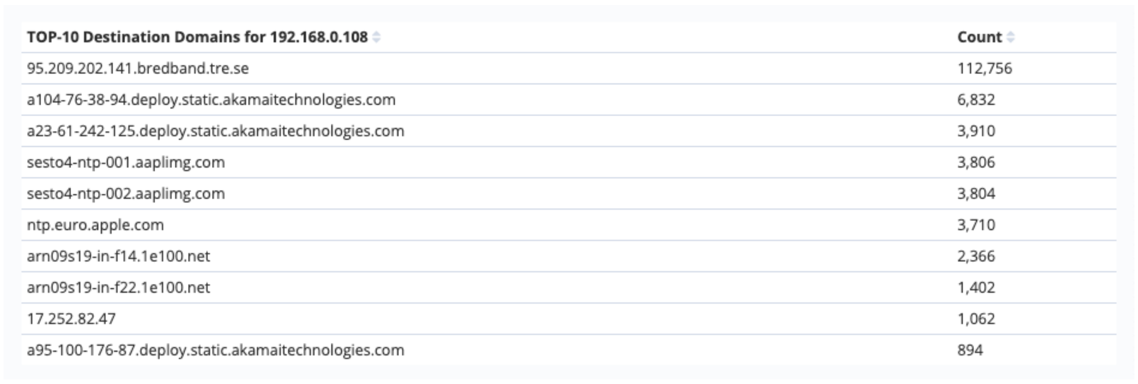
TOP-10 Destination Domains for 192.168.0.106	Count
ec2-54-171-194-3.eu-west-1.compute.amazonaws.com	63,108
ec2-34-251-143-94.eu-west-1.compute.amazonaws.com	62,644
ec2-34-247-203-19.eu-west-1.compute.amazonaws.com	62,630
ec2-34-250-29-146.eu-west-1.compute.amazonaws.com	61,512
ec2-63-33-76-85.eu-west-1.compute.amazonaws.com	59,610
ec2-34-250-93-255.eu-west-1.compute.amazonaws.com	59,176
ec2-34-254-127-195.eu-west-1.compute.amazonaws.com	5,310
ec2-34-246-24-12.eu-west-1.compute.amazonaws.com	5,142
ec2-54-72-140-59.eu-west-1.compute.amazonaws.com	4,926
ec2-34-226-31-147.compute-1.amazonaws.com	334

Figure 6: The top ten most contacted domains for 192.168.0.106.



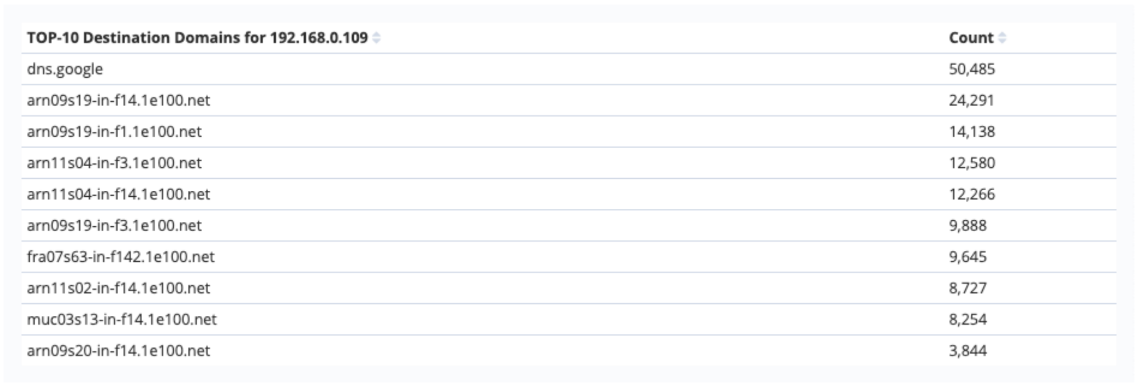
TOP-10 Destination Domains for 192.168.0.107	Count
arn11s04-in-f14.1e100.net	138,843
arn09s19-in-f14.1e100.net	61,850
dns.google	56,776
fra07s63-in-f142.1e100.net	34,976
arn11s02-in-f14.1e100.net	32,606
arn11s04-in-f3.1e100.net	27,686
arn09s19-in-f3.1e100.net	26,324
muc03s13-in-f14.1e100.net	12,987
arn09s10-in-f142.1e100.net	10,309
arn09s20-in-f14.1e100.net	9,863

Figure 7: The top ten most contacted domains for 192.168.0.107.



TOP-10 Destination Domains for 192.168.0.108	Count
95.209.202.141.bredband.tre.se	112,756
a104-76-38-94.deploy.static.akamaitechnologies.com	6,832
a23-61-242-125.deploy.static.akamaitechnologies.com	3,910
sesto4-ntp-001.aaplimg.com	3,806
sesto4-ntp-002.aaplimg.com	3,804
ntp.euro.apple.com	3,710
arn09s19-in-f14.1e100.net	2,366
arn09s19-in-f22.1e100.net	1,402
17.252.82.47	1,062
a95-100-176-87.deploy.static.akamaitechnologies.com	894

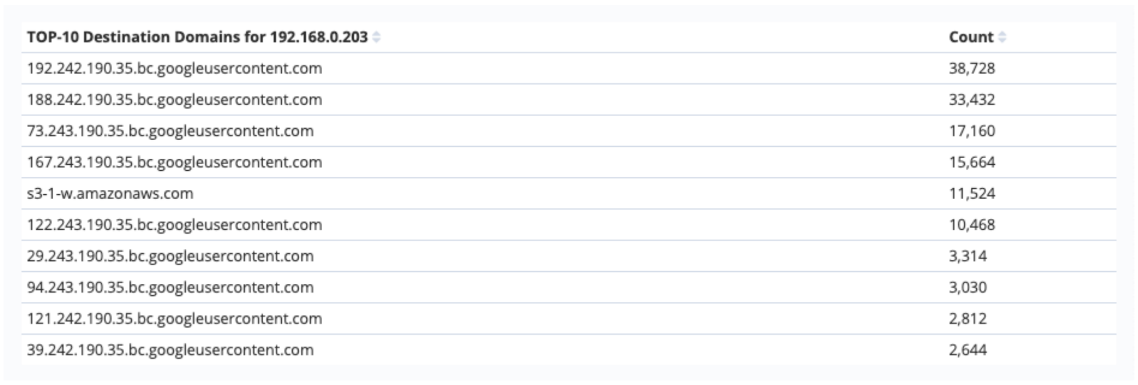
Figure 8: The top ten most contacted domains for 192.168.0.108.



TOP-10 Destination Domains for 192.168.0.109	Count
dns.google	50,485
arn09s19-in-f14.1e100.net	24,291
arn09s19-in-f1.1e100.net	14,138
arn11s04-in-f3.1e100.net	12,580
arn11s04-in-f14.1e100.net	12,266
arn09s19-in-f3.1e100.net	9,888
fra07s63-in-f142.1e100.net	9,645
arn11s02-in-f14.1e100.net	8,727
muc03s13-in-f14.1e100.net	8,254
arn09s20-in-f14.1e100.net	3,844

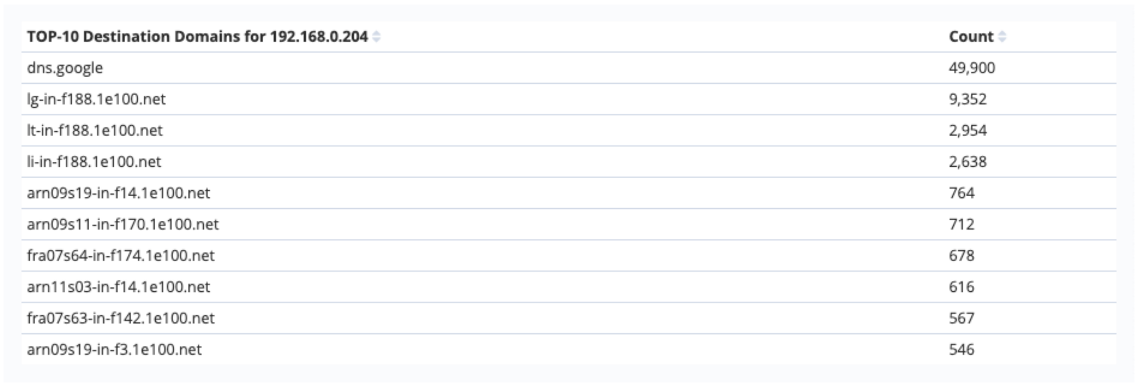
Figure 9: The top ten most contacted domains for 192.168.0.109.

F.2 North American Segment: Top Ten Destination Domains per Device



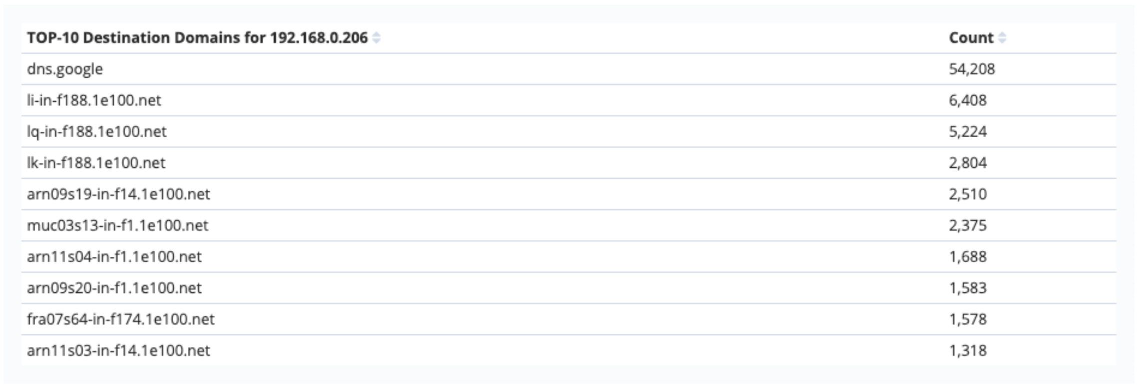
TOP-10 Destination Domains for 192.168.0.203	Count
192.242.190.35.bc.googleusercontent.com	38,728
188.242.190.35.bc.googleusercontent.com	33,432
73.243.190.35.bc.googleusercontent.com	17,160
167.243.190.35.bc.googleusercontent.com	15,664
s3-1-w.amazonaws.com	11,524
122.243.190.35.bc.googleusercontent.com	10,468
29.243.190.35.bc.googleusercontent.com	3,314
94.243.190.35.bc.googleusercontent.com	3,030
121.242.190.35.bc.googleusercontent.com	2,812
39.242.190.35.bc.googleusercontent.com	2,644

Figure 10: The top ten most contacted domains for 192.168.0.203.



TOP-10 Destination Domains for 192.168.0.204	Count
dns.google	49,900
lg-in-f188.1e100.net	9,352
lt-in-f188.1e100.net	2,954
li-in-f188.1e100.net	2,638
arn09s19-in-f14.1e100.net	764
arn09s11-in-f170.1e100.net	712
fra07s64-in-f174.1e100.net	678
arn11s03-in-f14.1e100.net	616
fra07s63-in-f142.1e100.net	567
arn09s19-in-f3.1e100.net	546

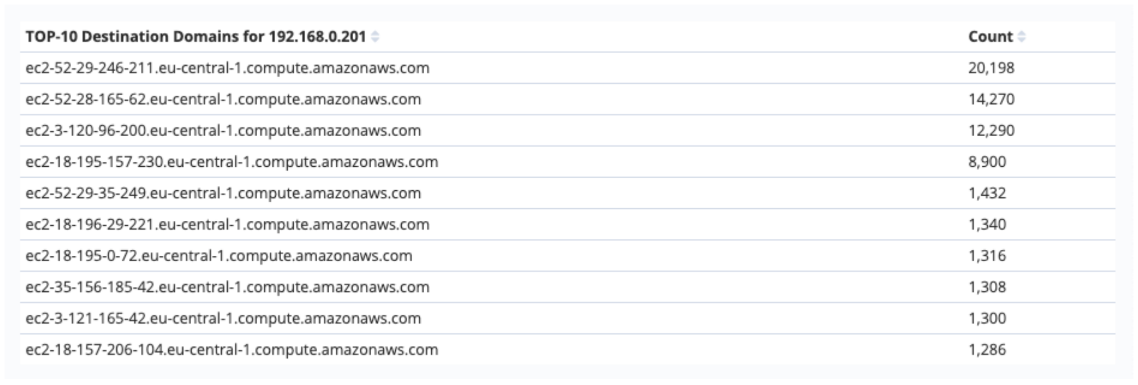
Figure 11: The top ten most contacted domains for 192.168.0.204.



TOP-10 Destination Domains for 192.168.0.206	Count
dns.google	54,208
li-in-f188.1e100.net	6,408
lq-in-f188.1e100.net	5,224
lk-in-f188.1e100.net	2,804
arn09s19-in-f14.1e100.net	2,510
muc03s13-in-f1.1e100.net	2,375
arn11s04-in-f1.1e100.net	1,688
arn09s20-in-f1.1e100.net	1,583
fra07s64-in-f174.1e100.net	1,578
arn11s03-in-f14.1e100.net	1,318

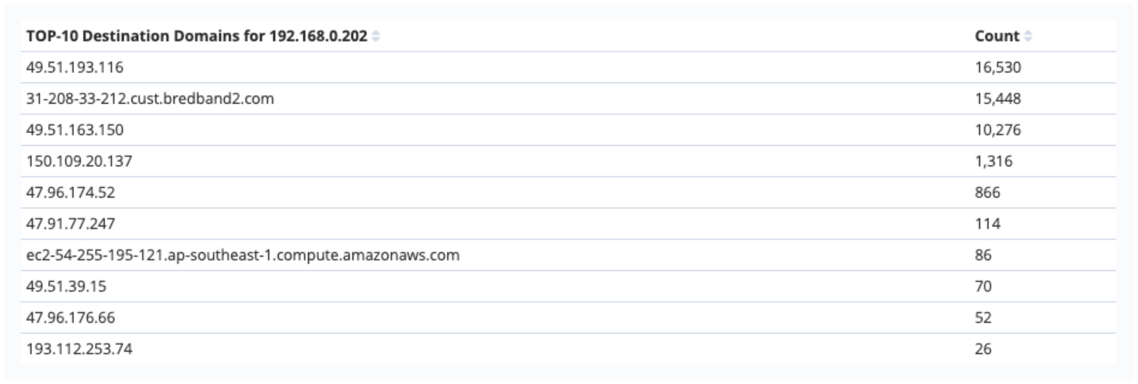
Figure 12: The top ten most contacted domains for 192.168.0.206.

F.3 Asian Segment: Top Ten Destination Domains per Device



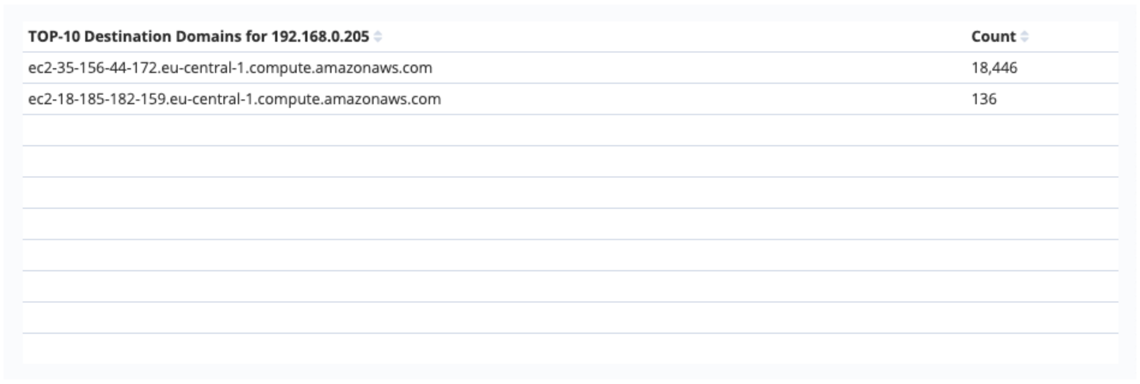
TOP-10 Destination Domains for 192.168.0.201	Count
ec2-52-29-246-211.eu-central-1.compute.amazonaws.com	20,198
ec2-52-28-165-62.eu-central-1.compute.amazonaws.com	14,270
ec2-3-120-96-200.eu-central-1.compute.amazonaws.com	12,290
ec2-18-195-157-230.eu-central-1.compute.amazonaws.com	8,900
ec2-52-29-35-249.eu-central-1.compute.amazonaws.com	1,432
ec2-18-196-29-221.eu-central-1.compute.amazonaws.com	1,340
ec2-18-195-0-72.eu-central-1.compute.amazonaws.com	1,316
ec2-35-156-185-42.eu-central-1.compute.amazonaws.com	1,308
ec2-3-121-165-42.eu-central-1.compute.amazonaws.com	1,300
ec2-18-157-206-104.eu-central-1.compute.amazonaws.com	1,286

Figure 13: The top ten most contacted domains for 192.168.0.201.



TOP-10 Destination Domains for 192.168.0.202	Count
49.51.193.116	16,530
31-208-33-212.cust.bredband2.com	15,448
49.51.163.150	10,276
150.109.20.137	1,316
47.96.174.52	866
47.91.77.247	114
ec2-54-255-195-121.ap-southeast-1.compute.amazonaws.com	86
49.51.39.15	70
47.96.176.66	52
193.112.253.74	26

Figure 14: The top ten most contacted domains for 192.168.0.202.



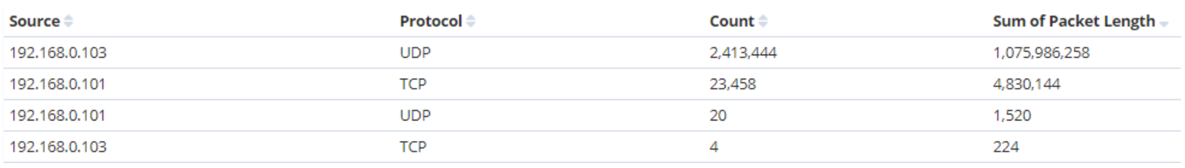
TOP-10 Destination Domains for 192.168.0.205	Count
ec2-35-156-44-172.eu-central-1.compute.amazonaws.com	18,446
ec2-18-185-182-159.eu-central-1.compute.amazonaws.com	136

Figure 15: The top ten most contacted domains for 192.168.0.205.

APPENDIX G

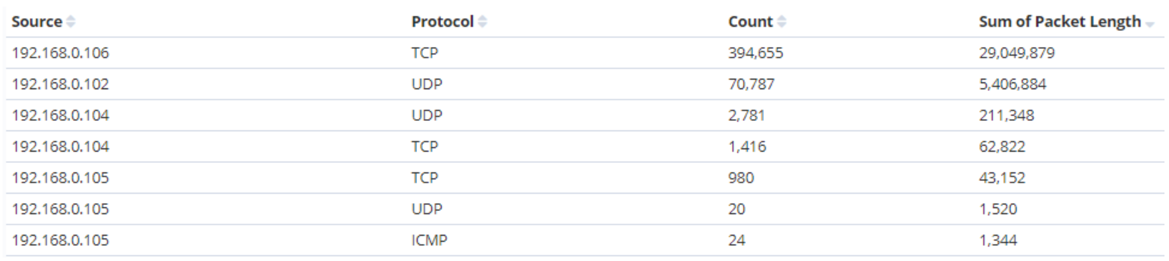
Traffic Metrics

G.1 Brand Segments



Source	Protocol	Count	Sum of Packet Length
192.168.0.103	UDP	2,413,444	1,075,986,258
192.168.0.101	TCP	23,458	4,830,144
192.168.0.101	UDP	20	1,520
192.168.0.103	TCP	4	224

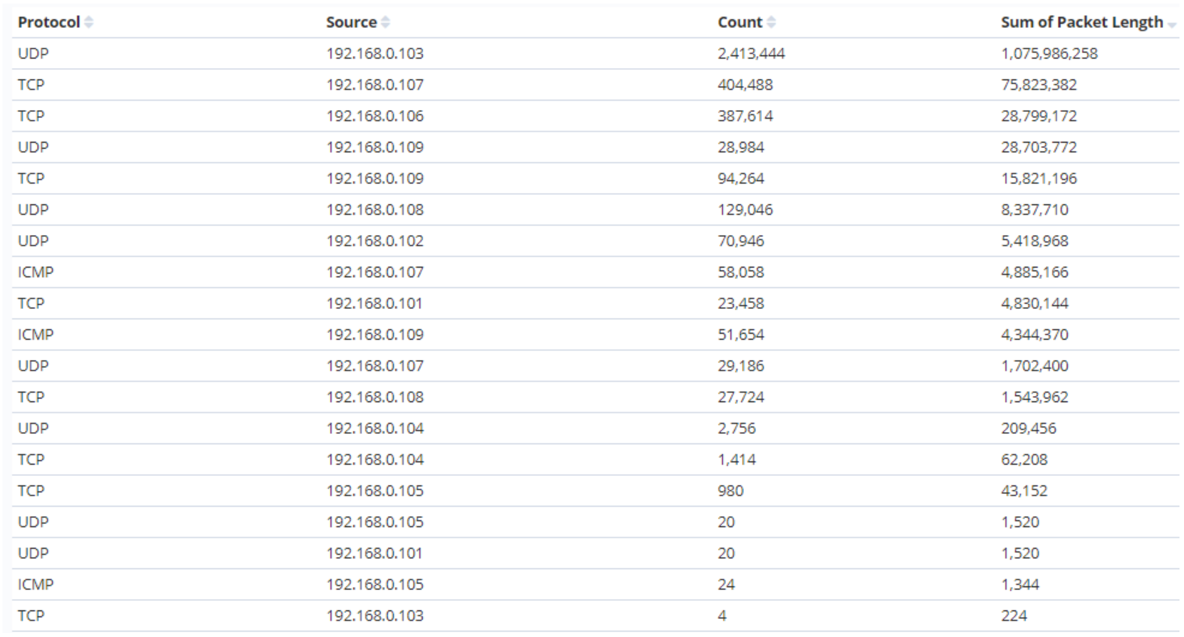
Figure 1: The size of the data sent by the IP cameras from the well-known (192.168.0.101) and less-known (192.168.0.103) brands segments using different protocols.



Source	Protocol	Count	Sum of Packet Length
192.168.0.106	TCP	394,655	29,049,879
192.168.0.102	UDP	70,787	5,406,884
192.168.0.104	UDP	2,781	211,348
192.168.0.104	TCP	1,416	62,822
192.168.0.105	TCP	980	43,152
192.168.0.105	UDP	20	1,520
192.168.0.105	ICMP	24	1,344

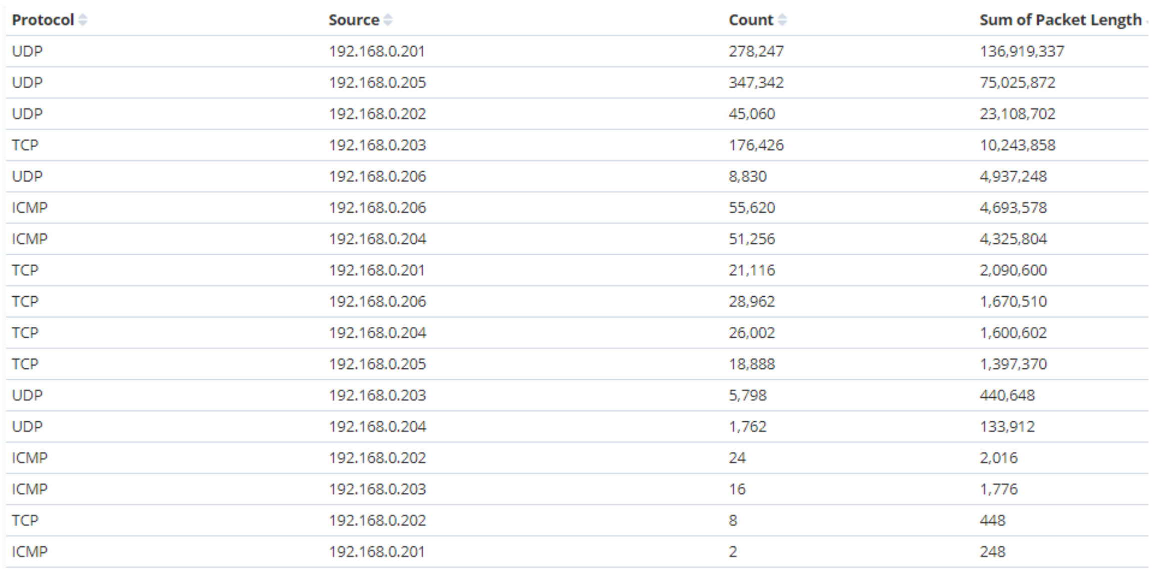
Figure 2: The size of the data sent by the different home automation devices from the well-known (192.168.0.104, 192.168.0.105 and 192.168.0.106) and less-known (192.168.0.102) brands segments using different protocols.

G.2 Region Segments



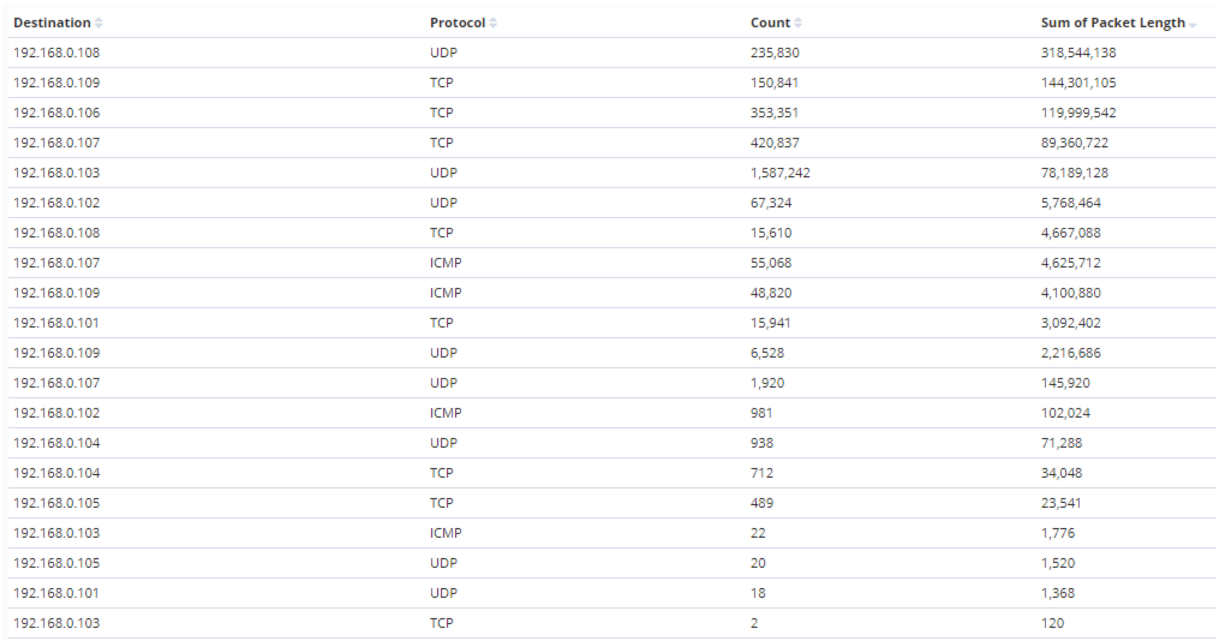
Protocol	Source	Count	Sum of Packet Length
UDP	192.168.0.103	2,413,444	1,075,986,258
TCP	192.168.0.107	404,488	75,823,382
TCP	192.168.0.106	387,614	28,799,172
UDP	192.168.0.109	28,984	28,703,772
TCP	192.168.0.109	94,264	15,821,196
UDP	192.168.0.108	129,046	8,337,710
UDP	192.168.0.102	70,946	5,418,968
ICMP	192.168.0.107	58,058	4,885,166
TCP	192.168.0.101	23,458	4,830,144
ICMP	192.168.0.109	51,654	4,344,370
UDP	192.168.0.107	29,186	1,702,400
TCP	192.168.0.108	27,724	1,543,962
UDP	192.168.0.104	2,756	209,456
TCP	192.168.0.104	1,414	62,208
TCP	192.168.0.105	980	43,152
UDP	192.168.0.105	20	1,520
UDP	192.168.0.101	20	1,520
ICMP	192.168.0.105	24	1,344
TCP	192.168.0.103	4	224

Figure 3: The size of the data sent by all the devices in the EU segment using different protocols.



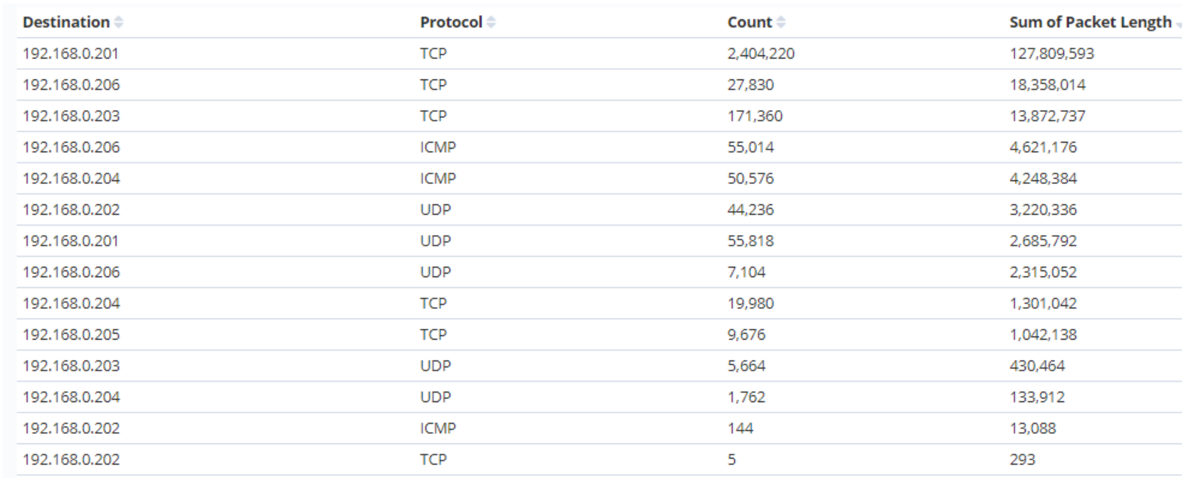
Protocol	Source	Count	Sum of Packet Length
UDP	192.168.0.201	278,247	136,919,337
UDP	192.168.0.205	347,342	75,025,872
UDP	192.168.0.202	45,060	23,108,702
TCP	192.168.0.203	176,426	10,243,858
UDP	192.168.0.206	8,830	4,937,248
ICMP	192.168.0.206	55,620	4,693,578
ICMP	192.168.0.204	51,256	4,325,804
TCP	192.168.0.201	21,116	2,090,600
TCP	192.168.0.206	28,962	1,670,510
TCP	192.168.0.204	26,002	1,600,602
TCP	192.168.0.205	18,888	1,397,370
UDP	192.168.0.203	5,798	440,648
UDP	192.168.0.204	1,762	133,912
ICMP	192.168.0.202	24	2,016
ICMP	192.168.0.203	16	1,776
TCP	192.168.0.202	8	448
ICMP	192.168.0.201	2	248

Figure 4: The size of the data sent by all the devices in the North American and the Asian segments using different protocols.



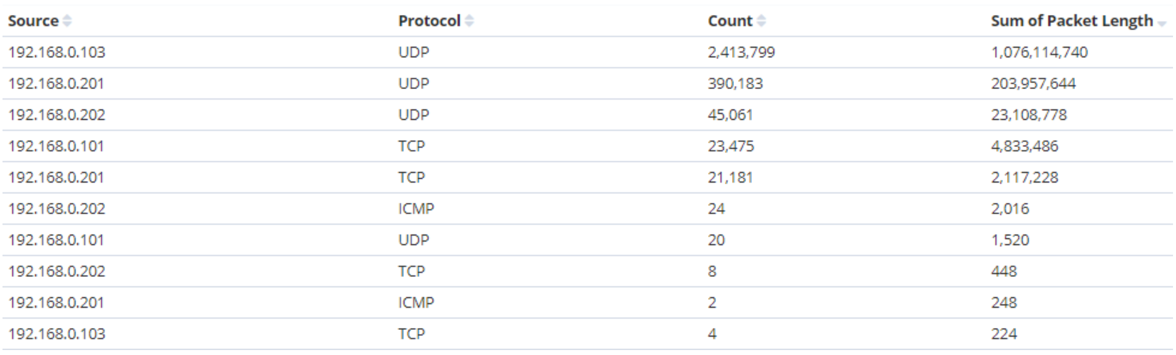
Destination	Protocol	Count	Sum of Packet Length
192.168.0.108	UDP	235,830	318,544,138
192.168.0.109	TCP	150,841	144,301,105
192.168.0.106	TCP	353,351	119,999,542
192.168.0.107	TCP	420,837	89,360,722
192.168.0.103	UDP	1,587,242	78,189,128
192.168.0.102	UDP	67,324	5,768,464
192.168.0.108	TCP	15,610	4,667,088
192.168.0.107	ICMP	55,068	4,625,712
192.168.0.109	ICMP	48,820	4,100,880
192.168.0.101	TCP	15,941	3,092,402
192.168.0.109	UDP	6,528	2,216,686
192.168.0.107	UDP	1,920	145,920
192.168.0.102	ICMP	981	102,024
192.168.0.104	UDP	938	71,288
192.168.0.104	TCP	712	34,048
192.168.0.105	TCP	489	23,541
192.168.0.103	ICMP	22	1,776
192.168.0.105	UDP	20	1,520
192.168.0.101	UDP	18	1,368
192.168.0.103	TCP	2	120

Figure 5: The size of the data received by devices in the EU segment using different protocols.



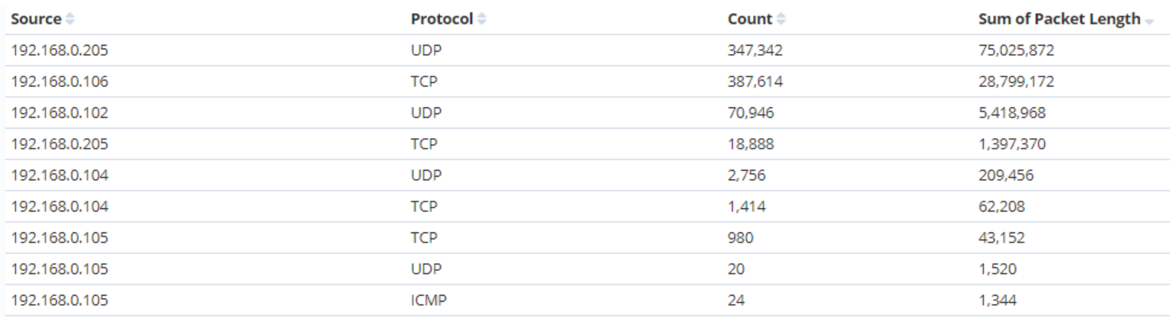
Destination	Protocol	Count	Sum of Packet Length
192.168.0.201	TCP	2,404,220	127,809,593
192.168.0.206	TCP	27,830	18,358,014
192.168.0.203	TCP	171,360	13,872,737
192.168.0.206	ICMP	55,014	4,621,176
192.168.0.204	ICMP	50,576	4,248,384
192.168.0.202	UDP	44,236	3,220,336
192.168.0.201	UDP	55,818	2,685,792
192.168.0.206	UDP	7,104	2,315,052
192.168.0.204	TCP	19,980	1,301,042
192.168.0.205	TCP	9,676	1,042,138
192.168.0.203	UDP	5,664	430,464
192.168.0.204	UDP	1,762	133,912
192.168.0.202	ICMP	144	13,088
192.168.0.202	TCP	5	293

Figure 6: The size of the data received by devices in the North American and the Asian segments using different protocols.



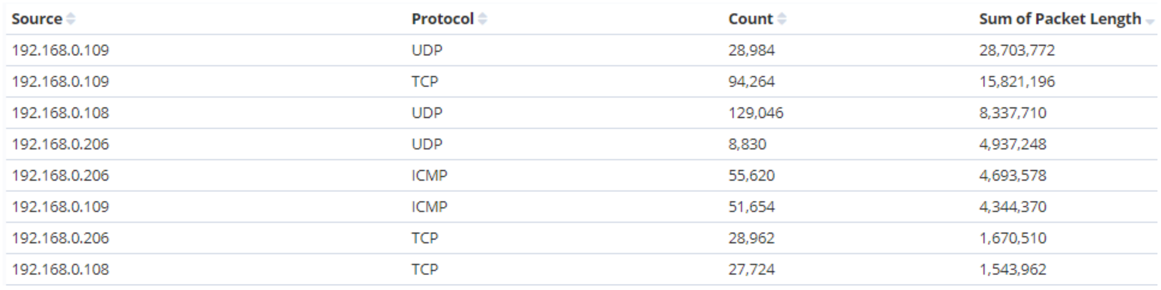
Source	Protocol	Count	Sum of Packet Length
192.168.0.103	UDP	2,413,799	1,076,114,740
192.168.0.201	UDP	390,183	203,957,644
192.168.0.202	UDP	45,061	23,108,778
192.168.0.101	TCP	23,475	4,833,486
192.168.0.201	TCP	21,181	2,117,228
192.168.0.202	ICMP	24	2,016
192.168.0.101	UDP	20	1,520
192.168.0.202	TCP	8	448
192.168.0.201	ICMP	2	248
192.168.0.103	TCP	4	224

Figure 7: The size of the data sent by the IP cameras from the EU (192.168.0.101 and 192.168.0.103) and the Asian (192.168.0.201 and 192.168.0.202) segments using different protocols.



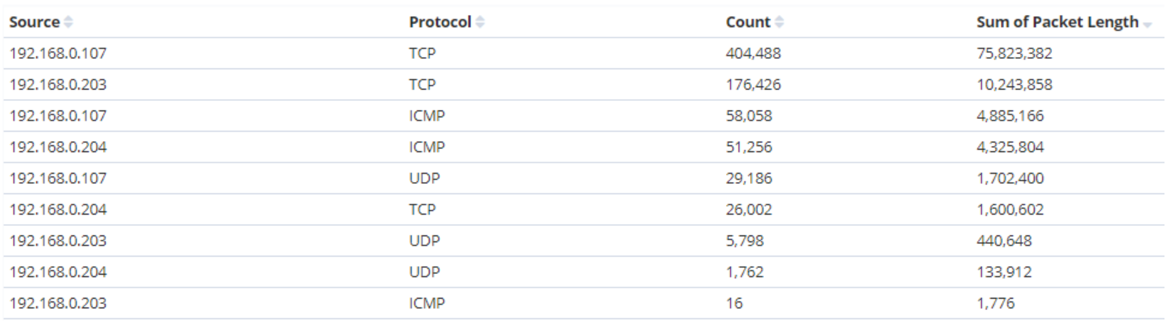
Source	Protocol	Count	Sum of Packet Length
192.168.0.205	UDP	347,342	75,025,872
192.168.0.106	TCP	387,614	28,799,172
192.168.0.102	UDP	70,946	5,418,968
192.168.0.205	TCP	18,888	1,397,370
192.168.0.104	UDP	2,756	209,456
192.168.0.104	TCP	1,414	62,208
192.168.0.105	TCP	980	43,152
192.168.0.105	UDP	20	1,520
192.168.0.105	ICMP	24	1,344

Figure 8: The size of the data sent by the different home automation devices from the EU (192.168.0.102, 192.168.0.104, 192.168.0.105 and 192.168.0.106) and the Asian (192.168.0.205) segments using different protocols.



Source	Protocol	Count	Sum of Packet Length
192.168.0.109	UDP	28,984	28,703,772
192.168.0.109	TCP	94,264	15,821,196
192.168.0.108	UDP	129,046	8,337,710
192.168.0.206	UDP	8,830	4,937,248
192.168.0.206	ICMP	55,620	4,693,578
192.168.0.109	ICMP	51,654	4,344,370
192.168.0.206	TCP	28,962	1,670,510
192.168.0.108	TCP	27,724	1,543,962

Figure 9: The size of the data sent by the different streaming devices from the EU (192.168.108 and 192.168.0.109) and the North American (192.168.0.206) segments using different protocols.



Source	Protocol	Count	Sum of Packet Length
192.168.0.107	TCP	404,488	75,823,382
192.168.0.203	TCP	176,426	10,243,858
192.168.0.107	ICMP	58,058	4,885,166
192.168.0.204	ICMP	51,256	4,325,804
192.168.0.107	UDP	29,186	1,702,400
192.168.0.204	TCP	26,002	1,600,602
192.168.0.203	UDP	5,798	440,648
192.168.0.204	UDP	1,762	133,912
192.168.0.203	ICMP	16	1,776

Figure 10: The size of the data sent by the different smart speakers from the EU (192.168.0.107) and the North American (192.168.203 and 192.168.0.204) segments using different protocols.