SPYKE: Security ProxY with Knowledge-based intrusion prEvention

Sheng Wang

Thesis to obtain the Master Degree in

Information Systems and Software Engineering

Supervisor: Prof. Miguel Filipe Leitão Pardal

Examination Committee
Chairperson: Prof. Miguel Nuno Dias Alves Pupo Correia
Supervisor: Prof. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Fernando Manuel Valente Ramos

June 2019
Acknowledgments

First of all, I would like to express my very great appreciation to Professor Miguel Pardal for accepting me as his student and guiding me throughout this journey with his ideas. I thank Rui Claro for his help, by being available for discussing and providing ideas. I am profoundly grateful to my family for encouraging me when things were not going as well as planned. A special thanks to my friends for all the support. Last but not least, I thank my girlfriend for motivating me and being always by my side. My research would have been impossible without the aid and support from all of you - Thank you.
Resumo

Num futuro próximo, a Internet of Things (IoT) será uma realidade e existirão muitos recursos de dados nos nossos espaços inteligentes assim como os sensores nas nossas casas inteligentes. Estes sensores irão eventualmente enviar os dados para a nuvem. Neste trabalho apresentamos SPYKE (Security ProxY with Knowledge-based intrusion prEvention), um intermediário de rede que fica entre os dispositivos IoT e a Internet, que fornece a visibilidade das comunicações que estão a ocorrer entre os dispositivos e os seus servidores remotos; e também permite bloquear e limitar conexões. Foi implementado um protótipo do sistema, que foi avaliado, tendo-se concluído que o seu sobrecusto é reduzido e que é eficaz contra um conjunto de ataques relevantes. O código fonte do SPYKE é aberto e o sistema pode ser facilmente instalado num Raspberry Pi ou noutra dispositivo equivalente.

Palavras-chave: Sistema de Detecção de Intrusões, Internet das Coisas, Spyware
Abstract

In the near future, the Internet of Things (IoT) will be a reality and there will be many sensors in our smart homes, for example, these data sources will eventually upload data to the cloud. In this work we present SPYKE (Security ProxY with Knowledge-based intrusion prEvention), a network intermediary that stands between IoT devices and the Internet, that provides visibility to which communications are taking place between devices and remote servers; and that is able to block and limit connections. We evaluated SPYKE with respect to the performance and security. It has low performance overhead and is effective against a set of common attacks. SPYKE is available as an open-source project and is deployable in inexpensive, off-the-shelf hardware like the Raspberry Pi.

Keywords: Intrusion Detection System, Internet of Things, Spyware
Contents

Acknowledgments .......................................................... iii
Resumo ........................................................................ v
Abstract ................................................................. vii
List of Tables .............................................................. xiii
List of Figures ............................................................. xv

1 Introduction ............................................................ 1
  1.1 Motivation .......................................................... 2
  1.2 Proposal Overview .............................................. 2
  1.3 Objectives ........................................................ 3
  1.4 Thesis Outline .................................................... 4

2 Background ........................................................... 5
  2.1 Middlebox Overview ............................................ 5
    2.1.1 Protocols ................................................... 6
    2.1.2 Proxies ...................................................... 10
    2.1.3 Survey of existing Proxies ............................ 11
  2.2 Smart Home ...................................................... 15
    2.2.1 Smart Devices ............................................ 15
    2.2.2 Smart Hubs ................................................. 15
    2.2.3 Security Monitors ....................................... 17
    2.2.4 Network Attack ......................................... 22
  2.3 Summary .......................................................... 24

3 SPYKE .................................................................. 25
  3.1 Goals .............................................................. 25
  3.2 Attacker model .................................................. 26
  3.3 Design ............................................................. 26
3.3.1 Funcionalidades .................................................. 26
3.3.2 Architecture ...................................................... 27
3.3.3 Summary .......................................................... 28
3.4 Prototype Implementation ........................................ 29
  3.4.1 Device Life Cycle .............................................. 29
  3.4.2 Data Model ..................................................... 30
  3.4.3 Data Flow ...................................................... 31
  3.4.4 Hardware ....................................................... 31
  3.4.5 Third-Party Tools ............................................ 32
  3.4.6 Summary ........................................................ 35

4 Evaluation ............................................................. 37
  4.1 Performance Evaluation ......................................... 37
    4.1.1 Baseline Performance ....................................... 38
    4.1.2 Overhead ..................................................... 39
    4.1.3 Commercial Devices ........................................ 41
    4.1.4 Discussion .................................................. 43
  4.2 Rule Enforcement ................................................ 44
    4.2.1 Quota ........................................................ 44
    4.2.2 Bandwidth ................................................... 45
    4.2.3 Quota and Bandwidth ....................................... 45
    4.2.4 Discussion .................................................. 46
  4.3 Effectiveness against attacks .................................. 46
    4.3.1 Deauthentication and Disassociation ........................ 47
    4.3.2 Authentication Request Flooding ............................ 48
    4.3.3 Dictionary ................................................... 48
    4.3.4 Spoofing ..................................................... 49
    4.3.5 Discussion .................................................. 49
    4.3.6 Summary .......................................................... 50

5 Conclusion ............................................................. 51
  5.1 Achievements .................................................... 51
  5.2 Future Work ..................................................... 51

Bibliography ............................................................ 53
A Privacy Protection

A.1 Techniques .............................................................................. 57
  A.1.1 Anonymization ................................................................. 57
  A.1.2 Obfuscation .................................................................. 58
  A.1.3 Summary ........................................................................ 59
A.2 Schemes .................................................................................. 59
List of Tables

2.1 HTTP proxy vs SOCKS proxy ........................................ 10
2.2 Results of proxy implementation survey .......................... 14
2.3 Comparison of Smart Home Monitors ............................... 21
2.4 Availability attacks .................................................. 23
2.5 Man-In-The-Middle attacks .......................................... 24

3.1 Example hostapd configuration .................................... 33
3.2 Example dnsmasq configuration .................................... 34

4.1 Average total transfer results for the performance experiments .......... 40
4.2 Average bandwidth results for the performance experiments ............ 41
4.3 SPYKE availability attacks coverage ................................ 50

A.1 Taxonomy of Vergara-Laurens et al. [22] .......................... 60
# List of Figures

1.1 The proposed system within the home environment . . . . . . . . . . . . . . . . . . 3
2.1 Overview of an HTTP request mediated by a proxy . . . . . . . . . . . . . . . . . 8
2.2 Overview of a SOCKS proxy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 Overview of forward and reverse proxies . . . . . . . . . . . . . . . . . . . . . . . 10
2.4 HS110 Smart Plug with Energy Monitoring by tp-link . . . . . . . . . . . . . . . . 16
2.5 Amazon Echo Dot 3 with Alexa embedded and listening . . . . . . . . . . . . . . 17
2.6 Comparison of Smart Home Ecosystems . . . . . . . . . . . . . . . . . . . . . . . 18
2.7 Architecture of Davies et al. [18] . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.1 SPYKE proposed architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.2 Block device from uploading data to an undesired third-party . . . . . . . . . . . . 29
3.3 State machine for a device managed by SPYKE . . . . . . . . . . . . . . . . . . . 30
3.4 Data Model of SPYKE database . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.5 Data Flow during SPYKE operation . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3.6 Raspberry Pi 3b+ within a case . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
3.7 Chain of rules of “ALLOWED” device with and without bandwidth defined . . . 35
4.1 Performance evaluation testbed . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
4.2 Outgoing communication measurement without SPYKE . . . . . . . . . . . . . . 39
4.3 Comparison of the first evaluation with and without SPYKE running . . . . . . . 39
4.4 Comparison of the first evaluation with different device number . . . . . . . . . . 40
4.5 TP-Link Smart Plug HS110 outgoing communication measurement . . . . . . . . 42
4.6 Amazon Echo outgoing communication measurement . . . . . . . . . . . . . . . . 43
4.7 Amazon Echo radio listening outgoing measurement . . . . . . . . . . . . . . . . 44
4.8 Results for Amazon Echo with limited upload quota . . . . . . . . . . . . . . . . 45
4.9 Results for Amazon Echo with limited upload bandwidth . . . . . . . . . . . . . 46
4.10 Results for Amazon Echo with limited upload quota and bandwidth . . . . . . . 47
4.11 Alfa network AWUS036NHA ................................................................. 48

A.1 Architecture of Vergara-Laurens et al. [22] ................................. 59
A.4 Three authentication architectures mentioned by D. Rotiroti [26] ........ 62
Chapter 1

Introduction

The Internet of Things is becoming increasingly widespread in home environments, with consumers transforming their homes into smart homes. As a consequence, data from sensors situated into devices are increasing in numbers. These devices are connected to the Internet, so these data are usually sent to remote cloud servers for data mining purposes.

Companies have been proposing different smart home systems, namely: Amazon Echo\(^1\), Google Home\(^2\), SmartThings\(^3\), Wink Hub\(^4\), among others. These systems use a hub as an intermediary to control the smart home. The ability to control the entire home gives the users a good and useful experience. However, there are significant weaknesses that can be used to expose private data.

One successful example of a smart home is the Amazon Echo. The Echo is a device with an embedded digital voice assistant, Alexa, that allows a user to control the different devices in a smart home by issuing voice commands. The user retrieves very useful information by saying the wake word “Alexa” followed with what the user intends to know or wants to do, for example: “Alexa, how is the weather today?” - Alexa tells the user how the weather is in the user location; “Alexa, add rice to my shopping list.” - Alexa adds the product to the shopping list which can be accessed by using Amazon’s Alexa app. Whenever Alexa receives the wake word, it starts to upload the voice recordings to the remote server until the user is finished. Then, it processes the voice, extracts the command, interprets it, and responds with the requested information.

Alexa can be extended with new skills\(^5\). For example, the user can ask for the calculator by saying “Alexa, open my calculator.”. However, this usage can lead to vulnerabilities, as stated

\(^1\)https://www.amazon.com/all-new-amazon-echo-speaker-with-wifi-alexa-dark-charcoal/dp/B06XCM9LJ4 accessed on May 16, 2018
\(^2\)https://store.google.com/?srp=/product/google_home accessed on April 13, 2018
\(^3\)https://www.smartthings.com accessed on April 13, 2018
\(^4\)https://www.wink.com/products/wink-hub/ accessed on April 13, 2018
\(^5\)A skill is an application that can extend Amazon Alexa.
in a recent report\(^6\) that has shown the Amazon Echo can be infected after the user activates a malicious skill, and, as a result it continues sending empty requests that maintain the connection established without the user’s knowledge. This way, it is effectively turned into a spy, that sends voice recording to an attacker.

### 1.1 Motivation

To avoid information leakage from smart home devices, there is a need for privacy protection of personal data. The majority of systems already use data encryption to hide the data content. Even so, they cannot hide their identities, because the destination IP address and the source IP address are needed to make the communication. Data leakage is very difficult to avoid, but it is possible to have a trusted intermediary, who can inform the user of the communications that are happening as it is intercepting them. Some examples are: data being sent beyond a limit, or data being sent to undesired locations on the Internet; data containing voice records or images sent to the Internet; unknown devices or users connected to the home network, and so on.

### 1.2 Proposal Overview

To provide privacy protection we propose SPYKE – Security ProxY with Knowledge-based intrusion prEvention – an intermediate network device between the user (and his devices) and the cloud (service providers). Our system is situated on the user’s side in order to intercept the data before it is sent to the Internet.

Figure 1.1 presents an overview of the proposed system. The red connections show a spy device trying to connect to the Internet, which is detected by SPYKE that informs the user giving the option whether to block further connections or not. It also informs the user when an unusual third-party tries to access the proposed system, i.e., when a device is uploading data to a unknown third-party, and then, the user has the ability to block the further connections.

SPYKE is used to forward device requests and receive its responses in such a way that the user remains oblivious of its functionality. It incorporated some typical functions of firewalls. The rules are defined to allow or deny those connections. For example, block connections to a certain IP address or domain name. The user is aware of the communications taking place, considering that SPYKE logs each communication, such as original sender, data size and destination. In the design of SPYKE, we used some concepts from Intrusion Detection Systems (IDS). On the one hand, it informs the user whenever a new device connects to the home network, asking his

---

\(^6\)https://www.wired.com/story/amazon-echo-alexa-skill-spying/ accessed on May 16, 2018
approval which provides an extra authentication besides passwords. On the other hand, it blocks devices that deviate from the usual behavior or that upload data to an unknown third-party. In our implementation, the user has the final decision on the permissions that a device has. These user policies can be, for example, to limit the maximum quota of a certain device per period of time and the bandwidth it can use.

We evaluate SPYKE and show that it can handle a large number of device’s rules, that enforce limitations on outgoing packets, with no significant degradation in performance. In addition, we assessed the network security by testing the system with a set of well-known attacks described by authors of [1].

1.3 Objectives

The goal was to build a prototype SPYKE, to provide privacy protection to the user. The objectives stated were to be able:

- to monitor communications between every single device in the network and the Internet;
- to limit the total transfer and the bandwidth;
- to block devices that are sending data to suspicious domains or IP addresses;

Figure 1.1: The proposed system within the home environment
• and to block traffic from new or unknown devices by default, i.e., enforce a whitelist access policy.

1.4 Thesis Outline

The remainder of the document is structured as follows. Chapter 2 exposes the concept of middleboxes, proxies, and smart homes. Chapter 3 provides the implementation of the proposed system, SPYKE. Chapter 4 presents the results obtained in regards to the efficiency and effectiveness of the proposed system. Finally, Chapter 5 presents the achievements and conclusions of this thesis, proposing some pointers for future work.
Chapter 2

Background

This Chapter presents the background and existing works in the following Sections: 2.1 introduces the middlebox concept, an intermediary box situated between the user and the remote server; 2.2 presents the smart home environment by referring devices, hubs, security monitors, and attacks; and 2.3 summarizes this Chapter.

2.1 Middlebox Overview

A middlebox is an intermediary device that is located in the communication flow of a network environment and the Internet. The main functionality of a middlebox is to intercept, inspect, filter or modify traffic instead of only performing packet forwarding. A middlebox can be a physical device or any software that has the mentioned functionality. One example is a Network Address Translation (NAT) box that allows several private network devices to share a single public IP address to access the Internet, because NAT has the ability to masquerade a private IP address to the public IP address, and vice versa.

Another middlebox example is a firewall. A firewall can monitor the network traffic by filtering incoming and outgoing packets. It needs a set of rules that defines whether to allow or block specific traffic. The basic functionalities of a firewall are packet filtering and stateful inspection: the first one filters packets that matched a defined pattern or set of rules such as the traffic state, port, and protocol; the second one inspects the state and filters the packets based on previous traffic, e.g., it caches and counts previous connections and when it achieves a defined number of connections it starts to block further ones.

If a middlebox is correctly deployed, it can improve the performance by dropping unwanted packets and performing an efficient distribution of packets through devices. It can also improve security by blocking packets that may compromise the private network.
2.1.1 Protocols

Now we presents the protocols that will be needed in the further on, in order to provide a better understanding of concepts presented. There are two main communication protocols, which are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). We present below these two protocols which are at the transport layer in the TCP/IP model. We also present the cryptographic protocols to provide secure communication over the network, namely Secure Shell (SSH) and Transport Layer Security (TLS). Afterward, we present the system services, such as Domain Name Service (DNS) and Dynamic Host Configuration Protocol (DHCP). Finally, we present two protocols that handle high level network packets between two end-points, namely Hypertext Transfer Protocol (HTTP) and Socket Secure (SOCKS).

UDP

UDP [2] is a standard that defines a communication through the Internet. It uses connectionless communication and provides checksums and port numbers, for data integrity and for addressing different functions consequently. It is not reliable since it does not track packets that are sent, i.e. the sender simply sends messages and he does not confirm if these messages are actually delivered. However, it is very useful for video streaming or multicasting.

TCP

TCP [3] is another standard that defines a communication through the Internet. TCP provides reliable, ordered, and error-checked delivery of each packet. In this case, the sender sends the message and waits for its delivery confirmation.

SSH

SSH [4] is a cryptographic network protocol that provides a secure channel over an unsecured channel. The communication is made over TCP and can be established with only two round-trips, which in practice represents four communication flows. The SSH server by definition uses port 22, and the server must have the public key of the user. The communications are the following: First, a user requests to establish a connection to a server which it responds with a challenge. Then, the user sends the challenge using his private key, and the server confirms the challenge to establish the communication through a secure channel, i.e., the communication data is encrypted.
TLS

TLS [5] is also a cryptographic protocol. It uses asymmetric cipher and Diffie–Hellman [6] key exchange to securely generate a random and unique session key. The main assurance is to provide authentication by using the private key, confidentiality by encrypting data, and integrity by message one-way encryption.

DNS

DNS [7] has the main goal to convert the IP address to human memorable name, i.e., domain name, and vice versa. When a user wants to connect to a remote server, he needs to know its IP address. For this reason, DNS is used due to the ability to convert the human memorable name to the IP address of the remote server. A DNS server by definition relies on the port 53 and may be used with UDP or TCP as the communication protocol.

DHCP

DHCP [8] distinguishes devices that are connected to the same network by assigning different IP addresses to each one. A device initializes a DHCP request that contains its MAC address to the DHCP server. Then, the DHCP server assigns dynamically an IP address to the device. The DHCP server maintains a lease file about the information of each connected device, such as the assigned IP address, MAC address, name, and lease time.

HTTP

HTTP [9] is a stateless protocol, in which the sender and the receiver do not need to know the state of each other. HTTP is considered reliable due to the usage of TCP as the communication protocol. It is classified at the application layer of the OSI model[10].

Figure 2.1 represents the typical usage of an HTTP proxy. There are several differences between the HTTP request made by the user through an HTTP proxy to the remote server and the direct HTTP request. The most important differences are highlighted in blue. The request Uniform Resource Identifier (URI) from the user to the HTTP proxy (in the third line) has to include the remote server’s host following with the path. Then, the HTTP proxy redirects the request to the indicated remote server and adds an extra field via (in the fifth line) indicating that the present request is handled by a proxy. In contrast, direct HTTP request has the only path in the request URI and the destination IP address along with the port of the remote server. Other differences include fields like forwarded (it discloses information of the user), max-forwards (it limits the number of times the message can be forwarded through the
HTTP request:
Src IP: 192.168.1.1 (port:38888)
Dst IP: 192.168.2.1 (port:8888)
Request URI: http://my.server.com/path
Host: my.server.com

HTTP Request:
Src IP: 192.168.2.1 (port:38882)
Dst IP: 192.168.3.1 (port:80)
Request URI: /path
Host: my.server.com
Via: HTTP Proxy

HTTP Response:
Src IP: 192.168.3.1 (port:80)
Dst IP: 192.168.2.1 (port 38882)
Response: Status Code 200

HTTP Response:
Src IP: 192.168.2.1 (port:8888)
Dst IP: 192.168.1.1 (port:38888)
Response: Status Code 200

Figure 2.1: Overview of an HTTP request mediated by a proxy

proxy) and proxy-authorization (authorization credentials for connecting to a proxy). Besides, there are also differences in respect to the HTTP response namely fields like age (it is used by the cache proxy), proxy-authenticate (it requests authentication to access the proxy), and vary (it tells the cache proxy how to match the future request headers to decide whether the cached response can be used rather than requesting a fresh one from the server).

SOCKS

SOCKS [11] is a protocol that relays TCP sessions to allow application users access through a firewall. It operates at the session layer of the OSI model. SOCKS can be used to establish communication through a proxy.

Figure 2.2 shows several steps to achieve the established connection between the user and the remote server through a SOCKS proxy.

First of all, the user tries to connect the SOCKS proxy in order to get an accept. Secondly, it
makes a request to the remote server passing through the SOCKS proxy. This request includes the destination IP address, the listening port number of the remote server, the username and the SOCKS version, i.e. SOCKS4 or SOCKS5. SOCKS5 is an improvement of SOCKS4 by adding UDP to improve domain name resolution, and still using TCP to have a strong authentication scheme. For this case, the request may include the destination domain name instead of the destination IP address. After that, the remote server responds with acceptance or rejection to the user again passing through the proxy. Finally, in case that the response came with acceptance, the connection is established.

Since it supports TCP, i.e., it guarantees that packets arrive in the same order as they are sent, and UDP, i.e., it translates flexibly URI into IP addresses by using DNS, it is considered reliable and flexible. Furthermore, it does not rewrite the packet header, hence the packets have no chance to be misrouted.

One of the best examples of the use of SOCKS is The Onion Router (TOR)\textsuperscript{1} [12], in which data is encrypted and routed through several proxies before it is delivered to the destination. In this case, the sender chose beforehand which proxies the data are going to pass through, and has the public key of the chosen proxies. Regarding the encryption process, the data is encrypted with the destination’s public key. Then, like a chain, the encrypted data is again encrypted with the proxy’s public key and once again encrypted with the other proxy’s public key, and so on. In this way, the proxy knows the sender and the next receiver but it does not know the content of the message. Furthermore, it does not know if the next receiver is the final destination, hence it is hard to track the original sender as well as the final destination.

\textbf{Comparison of HTTP and SOCKS proxies}

Table 2.1 shows the list of some differences between HTTP proxy and SOCKS proxy. One of the differences is the level: SOCKS proxy (session level) is considered low-level compared to HTTP proxy (application level). For this reason, one supports more protocols than the other. Higher level means that the data passed through can be read and interpreted, and in contrast, SOCKS proxy simply opens a secure tunnel with data encrypted. In respect to connection, HTTP proxy sends a message, after the message is delivered, it closes the connection. Since it is stateless protocol it simply redirects messages. This is the reason it rewrites message header to redirect message. The last difference is the protocol used for transporting message, both HTTP and SOCKS proxies use TCP but the last one may additionally use UDP.

\textsuperscript{1}https://www.torproject.org/about/overview.html.en accessed on April 13, 2018
### 2.1.2 Proxies

The protocols presented in the previous section can be used to implement middleboxes. One example of a middlebox is a proxy that forwards the two-way communication by requesting services, such as a file, a web page, among other resources available on a remote server, and by responding to the request.

A proxy is a mechanism that can be used for achieving privacy protection by filtering or blocking the message from the unknown source, masking the IP address of the original sender, among other usages.

Proxies can have features to minimize the remote server latency, namely *cache proxy* and *compression proxy*. The first one saves contents such as images, videos, among other files given as response from the remote server. The second minimizes the message body size. There are also privacy protection techniques that can be used as presented in Appendix A.

Proxies are categorized into different types according to their placement in the network. Two examples of such placement, *forward proxy* and *reverse proxy*, are illustrated in Figure 2.3.

![Figure 2.3: Overview of forward and reverse proxies](image)

The first one acts as the user, i.e., it stays on the private network’s side, and forwards the requests to the remote server and receives the response from it. In addition, it can also be used to hide the users’ identity and to restrict their requests by filtering. The second one acts as the remote server, i.e., it stays on the server cloud’s side, and forwards the request to the remote
server from the users as well as the response from the server to the users. Again, it is used to restrict the user’s requests avoiding overload, e.g., denial of service, and provides load balance, e.g., requests are well distributed to the different servers.

Besides those two mentioned above, there are three other types of proxies, which are open proxy, closed proxy, and transparent proxy: the first one, as its the name suggests, is accessible by any Internet user; the second one, as the name suggests, is only accessible by users within a group in the network; and the third one only requires that pre-configuration in order to remain fully functional, after that, no further configurations are needed. On contrary, a standard proxy involves the user to be aware of its presence by often requiring configurations, e.g., configuring on their browsers each time they want to use it.

To conclude the concept of proxies, a tunnel proxy is an intermediate who creates a secure channel in an unsecured network. Furthermore, it cannot read data that is passing through, it simply creates a tunnel to exchange data between the user and the remote server. One way to provide this secure channel is by using SSH.

2.1.3 Survey of existing Proxies

We performed a survey of existing proxy implementations. The majority of proxy implementations provide simple anonymity, support the HTTP protocol, and filtering. Proxies which support SOCKS5 protocol demand more performance and in addition give more security as an advantage. Most of them are reverse proxies and not forward proxies due to the fact that they are designed in order to protect the server by staying on the server cloud’s side. In spite of the fact that the majority of proxies support data compression, only ZIProxy gives an extra compression in order to optimize performance and reduce latency. Beyond that, Privoxy was designed with privacy mechanism in mind and it gives an extra privacy mechanism by modifying data, e.g., cookie, to prevent their disclosure while other proxies give almost the same privacy mechanism.

Tinyproxy

Tinyproxy is an HTTP proxy which may be used as a forward proxy or reverse proxy. Likewise, it forwards HTTP requests by adding a via field with its name in order to announce its presence and forwards HTTPS requests by simply redirecting them. There is no data compression due to the intention of keeping it simple and fast. Every time a request is made, the IP address of the original sender is presented in the IP header. Tinyproxy provides little anonymity by using proxy chain, i.e., there is a list of proxies to send before reaching the destination. Tinyproxy can be configured to deny certain subnetworks or IP
addresses and can refuse the connection after reaching a certain number of connections. Indeed, it can blacklist or whitelist based on domain name or web address, e.g., it can be configured in a way that only certain web address can be accessed (whitelist) or cannot be accessed (blacklist). It is also a transparent proxy since it only needs to be configured at the beginning and after that, it does not need more configurations. To give its management more control, Tinyproxy also provides authentication with credentials.

**Shadowsocks**

Shadowsocks provides two types of proxy as well, forward proxy and reverse proxy. When a user wants to send a request, he first needs to authenticate himself by knowing the proxy’s domain name or IP address with corresponding port and the correct password. Once authenticated, it becomes transparent since it does not need more configuration after that. Although the data is not compressed, it is encrypted. It uses the SSH protocol to communicate securely in an unsecured network. For this reason, it may be considered as a tunneling proxy, due to the fact that it creates a secure channel for the communication between the user and the server.

This proxy provides the following privacy protections: authenticity, due to the need for authentication at the configuration phase; anonymity once it uses a plugin very similar to the Pluggable Transport\(^2\) plugin from TOR; integrity since it uses SSH protocol.

**Mitmproxy[13]**

Mitmproxy, (Man-in-the-middle proxy) is a forward and reverse proxy as well. The principal objective is for TLS Interception and SSL Inspection\(^3\), for this reason, it is also considered a TLSproxy[14]. It works as follows: first, it intercepts and terminates the user SSL/TLS connection; then, it re-establishes the connection to the server; finally, it can inspect plaintext and use the installed certificate previously in the interception in order to establish a TLS connection back to the user.

On one hand, this proxy can be used to violate user privacy by accessing the data, on the other hand, it can be used for defense by verifying the message content. It has the ability to intercept, inspect, and also modify HTTP and HTTPS requests and responses. It can save HTTP conversation using cache proxy in order to perform replay HTTP responses of a previously recorded server or to analyze it later.

---

\(^1\)https://github.com/zythom/cryptopshipper
\(^2\)https://gitweb.torproject.org/torspec.git/tree/pl-spec.txt, accessed on June 15, 2018
\(^3\)https://tlseminar.github.io/tls-interception/, accessed on May 24, 2018
NGINX

Nginx is a reverse proxy along with cache proxy embedded in the server cloud’s side. Therefore it provides load balance to servers. Moreover, it uses an asynchronous event-driven which can handle many requests. It uses the TLS protocol to ensure the privacy and integrity of data.

Apache traffic server

ATS is a reverse proxy and forward proxy. It receives traffic as TLP (traffic light protocol). It is a transparent proxy and also includes caching. It uses ModSecurity\(^4\) which is an Open Source Web Application Firewall. For this reason, ModSecurity provides privacy protection with network traffic filtering.

Apache httpd

Apache httpd is similar to NGINX, i.e., it is a reverse proxy along with cache proxy, it provides load balancing and uses TLS protocol. In addition, it provides fault tolerance with automatic recovery and dynamic configuration. It may rewrite header or content of traffic.

This proxy provides the following privacy protections such as anonymity by rewriting the traffic and integrity by using the TLS protocol.

Lighttpd

Lighttpd is similar to Apache httpd but optimized with speed in mind. It uses FastCGI, which consists in removing the file when it is sent, to reduce the overhead. Furthermore, the HTTP response is kept open while the server can send multiple responses even if the user did not make any HTTP request. It is also embedded on the server’s side.

ArashPartow

ArashPartow proxy is a TCP proxy server. It provides load balance as well as it is considered a forward and reverse proxy. It can inspect, filter or modify data flowing between the user and the remote server. Finally, it has access control mechanisms.

This proxy provides privacy protections such as filtering by giving access control and anonymity by modifying data.

\(^4\)\url{http://www.gupiaoya.com/tools/Miscellaneous/ModSecurity_The_Open_Source_Web_Application_Firewall.pdf}, accessed on May 30, 2018
Privoxy

Privoxy is focused primarily on privacy with filtering capabilities. Privoxy has ability to manipulate cookies, HTTP headers and even web page data, e.g., filter advertisements. It can be used as proxy chain or, in contrast, as stand-alone.

This proxy provides privacy protections such as filtering, remove extra content from web pages and anonymity with proxy chain.

ZIProxy

ZIProxy is a forward proxy, the main purpose is to perform an extra compress to already compressed and optimized data. It is considered as a lightweight proxy, for the reason that it provides low latency, low memory, and low battery consumption.

Although this proxy provides no privacy protections, it compresses data to optimize performance.

The majority of proxies on Table 2.2 has common mechanisms included, like using TLS to provide a secure channel, using proxy chain to forward the request, and using filtering to block packet from a certain sender. We consider Privoxy with privacy mechanism because it has the ability to manipulate the data content.

<table>
<thead>
<tr>
<th>Proxy</th>
<th>Protocol</th>
<th>Forward</th>
<th>Reverse</th>
<th>Data compression</th>
<th>Privacy mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tinyproxy</td>
<td>HTTP</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Shadowsocks</td>
<td>SOCKS5</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Mitmproxy</td>
<td>HTTP</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>NGINX</td>
<td>HTTP</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Apache traffic server</td>
<td>HTTP</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Apache httpd</td>
<td>HTTP</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Lighttpd</td>
<td>HTTP</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>ArashPartow</td>
<td>TCP</td>
<td>N</td>
<td>Y</td>
<td>n/a</td>
<td>N</td>
</tr>
<tr>
<td>Privoxy</td>
<td>HTTP</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>ZIProxy</td>
<td>HTTP</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

After some analysis, ShadowSocks, Tinyproxy, and Privoxy are highlighted. They are able to forward and receive traffic and provide authenticity by password authentication. ShadowSocks claims to be secure, fast and supports more protocols. Nevertheless, it requires two proxies in order to make a secure channel between them, one on the private network’s site and another on the server cloud’s site. Tinyproxy is lightweight but fast, moreover, it can rewrite header to provide anonymity. And Privoxy is also lightweight and has an enhanced privacy mechanism due to its ability to remove the malicious content of HTTP communications.
2.2 Smart Home

By definition, Smart Home is a home, inhabited by people, equipped with smart devices that can be controlled remotely by the user. This section is subdivided into the following subsections: 2.2.1 presents the well-known smart devices, and their importance to a smart home; 2.2.2 describes smart home hub systems and a recently found vulnerability as well as its solution; 2.2.3 lists the existing home network monitors that have functionalities like listing all connected devices, checking and controlling the traffic, and informing all these situations to the user; finally, 2.2.4 shows a set of well-known attacks that can be performed in the home network environment.

2.2.1 Smart Devices

A smart environment results from the composition of a set of smart devices. These smart devices are usually connected to each other to allow a better experience to the user. Some of them are even connected to the Internet. However, devices report data about the user. Examples of smart devices that report home status are smart sensors, e.g., motion sensor reports whenever someone passes by, smoke detector reports the user whenever it detects smoke. Some devices can also perform actions like a coffee machine, a smart plug, or a smart light.

Figure 2.4 shows a running smart plug, connected to an access point, that is able to monitor energy usage. The user is able to remotely control this device through Kasa, performing commands like turn plug on or off. In addition, the user is also able to check the energy consumption.

2.2.2 Smart Hubs

For the purpose of collecting data from smart devices and controlling them, there are smart hubs like SmartThings, Wink, Amazon Echo, and others. Figure 2.5 shows an Amazon Echo Dot 3 with Alexa embedded. Those hubs are intermediates that control the communications between smart devices and the server cloud.

The authors of [15] present some of those smart hubs’ ecosystem, namely SmartThings, Wink and Amazon Alexa, represented in Figure 2.6. In the SmartThings’ ecosystem, each device communicates with a hub which remains in constant communication with the server cloud. The communication is encrypted to provide privacy and data integrity. Moreover, the user uses a smart application to manage devices through the server cloud. In the Wink Hub’s ecosystem, each device also communicates with a hub which once more remains in constant communication.

---

5Kasa is an application developed by TP-LINK: https://www.tp-link.com/us/kasa-smart/kasa.html, accessed on May 19, 2019
with the server cloud. In addition, the user can manage devices through the server cloud or
directly through the private network. In the Amazon Echo’s ecosystem, the Virtual Personal
Assistant (VPA) Alexa relies on the voice channel to communicate with users. It is similar to the
previous two, so each device communicates with a hub that remains in constant communication
with the server cloud.

Kumar et al. [16] and Zhang et al. [17] highlight the security risks of Amazon Echo and
Google Home for using voice-controlled third-party skills. They explained *voice squatting* attack
and in addition, Zhang et al. also explained *voice masquerading* attack. In the first one, the
attacker makes a skill which invocation is phonetically similar to the original skill. Then, the
user may activate it and give the attacker sensitive information that is supposed to use on the
original skill. In the last one, the attacker again makes a simple skill to be executed by the user.
And when the user is executing the attacker’s skill, he may want to switch or activate another
skill without interrupting the current one. In this case, the attacker’s skill may fool the user by
saying that the wanted skill is activated and waiting for the user to give sensitive information.

They first exploited Amazon Echo and Google Home, then to mitigate the first threat they
sought a technique by building a skill-name scanner that is able to convert the invocation name
of a skill into a phonetic expression to measure the phonetic distance between two different skills.

For the second threat, a context-sensitive detector was built upon the VPA infrastructure.
2.2.3 Security Monitors

Aside from smart hubs, there are security monitors which control the network traffic. Davies et al. [18] presented an example of security monitor, called Privacy Mediator situated in a Cloudlet which is a small data center located between the Internet and the devices and it is within the trust domain of the end user. It can be installed on a high-end WiFi access point or can be physically installed in homes, schools or small businesses. Figure 2.7 shows that raw information, obtained by devices, is converted and then aggregated and obfuscated by Privacy Mediator before sending it to the Internet. Moreover, the authors mentioned that the information flow is very important for the user and also the remote server, so having a good set of data redaction and privacy policy enforcement is needed. Privacy Mediator also has user policies to configure which devices have access to the Internet.
More examples of security monitors are: *Pi-Hole*\(^6\) and *IoT Inspector*\(^7\), which are open-source, and *Fingbox*\(^8\), *Google WiFi*\(^9\) and *Bitdefender BOX* \(^10\), which are commercial offerings. Their characteristics are described below.

**Pi-Hole**

Pi-hole is designed to block ads in smart home’s devices using as standard hardware a Raspberry Pi. However, it may be run in any hardware that supports Raspbian, Ubuntu, Debian, Fedora and CentOS as the Operating System. It works with *dnsmasq* which is a DNS server that

---

\(^6\)https://pi-hole.net/ accessed on April 8, 2019
\(^7\)https://iot-inspector.princeton.edu/ accessed on May 4, 2019
\(^8\)https://www.fing.com/products/fingbox#info accessed on April 13, 2019
\(^9\)https://support.google.com/wifi/answer/7168315?hl=en accessed on April 8, 2019
\(^10\)https://download.bitdefender.com/resources/media/materials/box/v2/user_guide/BOX_UserGuide_v2_en_.pdf accessed on April 8, 2019
translates IP addresses to their associated domain names and vice versa along with DHCP server that provides and assigns IP address automatically to connected devices. It is not designed to act as a router, but it can run with `hostapd`. As mentioned before, the main goal is to block advertisement.

To achieve this, it requires a smartphone, laptop, or any device to set the DNS to the Pi-hole IP address. After that, all web requests will be asked through Pi-hole, hence it will only respond with those that are allowed by the administrator.

**IoT Inspector**

IoT Inspector is designed to monitor the home network and inspect IoT devices. It collects the home network information like where devices are connected with, and how many and often data is exchanged. IoT Inspector sends the collected information to their remote server for the data mining purpose.

Concerning security, IoT Inspector provides information like whether a device is sharing information with third-parties, or a device is hacked and controlled by an attacker, or whether a device is slowing down the home network. In addition, it is able to identify the device’s manufacturers based on the MAC address. As drawbacks, it reduces the home network performance, and may not work with all IoT devices.

IoT Inspector is an application that requires a laptop or desktop in order to be functional.
Fingbox

Fingbox’s name came from “fast ping box”. The main functionality is to check every device that is connected to the network. Similarly to Pi-Hole, Fingbox is not designed to act as a router. The idea is to plug in the Ethernet cable to the router, setup with a mobile application and then it starts working.

Regarding the security, it has the functionality of detecting and blocking connected devices and their communications. In addition, it can detect and block unconnected devices, that are near to the home network, even before they try to connect it.

Fingbox can schedule and pause the internet access, analyze the bandwidth of each device and it can even discover near devices which are not connected. This way, it may protect against physical networking hacking and WiFi eavesdropping.

Google WiFi

Google WiFi is intended to provide fast and seamless WiFi connectivity to the home. To use it, it needs a modem, broadband connection to an Internet Service Provider (ISP), a Google Account and Google Wifi application for remote control through a smartphone. It can use several Google WiFi routers at the same time to improve the WiFi range, the only requirement is to connect one to the modem via Ethernet cable. It is transparent since it uses the protocol Universal Plug and Play (UPnP). Google WiFi has several great features as a router. For our purpose, we are going to focus only on performance and security.

Regarding performance, on the setup stage, it searches already existing WiFi channel (1-14) in the air, in order to avoid mesh connection. It uses the same SSID name for both the 2.4 and 5 GHz band network, i.e. when the devices are near they will be connecting to the 5 GHz band, hence the speed can be more than 100 Mbps. Google WiFi provides Band Steering (dual-band operation), i.e. a device is connected to the 2.4 band networking, and when it becomes nearer, it will switch to the 5 GHz band automatically.

Concerning security, Google WiFi is able to provide graphs about data usage on their application. It is able to blacklist IP or domain name as well as give priority to certain devices. All connections to their servers are done through TLS which provides some level of privacy.

Bitdefender Box 2

Bitdefender Box 2 has the intention to secure the smart home network. Bitdefender is an antivirus company, and for this reason, instead of being on a computer, Bitdefender Box 2 provides the antivirus to all the network. As their documentation says, it only requires a small
setup to be functional. In addition, on the setup process, it deauthenticates all devices from introduced WiFi and connects to it with the same SSID name and password. It requires an ISP modem.

Regarding performance, it uses as concurrent dual-band network 2.4 GHz and 5 GHz. It provides Gigabit Ethernet to ISP modem. In terms of security, Box offers vulnerability assessment to detect network security flaws, exploitation prevention to block attempts to exploit vulnerabilities in connected devices, local device security to protect connected computers, smartphones, and tablets in place of a locally installed antivirus, as well as anomaly detection, brute force detection, and data protection. The application can be used outside of the house through the Internet. Which means that the data is uploaded to their server.

Comparison

Table 2.3 compares the mentioned systems with SPYKE that we are proposing (detailed in Chapter 3). Regarding the authentication, Google WiFi, Bitdefender Box and SPYKE provide authentication by WiFi using the password protocol WiFi Protected Access (WPA)². Other systems’ authentications are granted with an existing router, i.e., any device that is connected to the router, can connect to them. In case of Pi-Hole, devices need to know its IP address and set it as DNS server so it be used as the intermediary. All systems can be defined by user policy to block the Internet access as well as limiting the bandwidth of each connected device, except for Pi-hole and IoT Inspector. Regarding the Network intrusion detection, Fingbox and Bitdefender Box notify the user who is trying to connect the system. Finally, Pi-Hole and Bitdefender Box perform filtering due to the ability to block traffics that contain advertisement content.

<table>
<thead>
<tr>
<th></th>
<th>Pi-Hole</th>
<th>IoT Inspector</th>
<th>Fingbox</th>
<th>Google WiFi</th>
<th>Bitdefender Box 2</th>
<th>SPYKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>User policy</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Network intrusion detection</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N*</td>
</tr>
<tr>
<td>Filtering</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N*</td>
</tr>
<tr>
<td>Open-Source</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

SPYKE was designed to implement network intrusion detection and filtering, but the prototype did not implement them and they are planned as future work.
2.2.4 Network Attack

Nowadays, the majority of network communications have encryption applied in order to provide privacy and integrity in the traffic content. Nevertheless, encryption can no longer guarantee privacy protection. Third-parties, e.g., ISP, have access to the network traffic and so, they may be able to analyze the traffic.

Apthorpe et al. [19] showed how encrypted data is vulnerable because of the metadata of the traffic to the attacker. They referred several attacks using these metadata against the encrypted communications. Mostly, side-channel privacy attack and fingerprinting attack which can be performed by ISP: the first attack comes with the idea of analyzing the traffic pattern, i.e., during a period of time, how traffic behaves, a concrete example is a peak of traffic at a certain time may show a user’s activity; the second attack takes the advantages of metadata by analyzing the traffic information, e.g. sender IP address, destination IP address, timestamp, among other sensitive information.

In order to mitigate these attacks, they showed three solutions: the first one is to retain all traffic in the private network by blocking the traffic of going outside, however, the majority of smart devices like Amazon’s Alexa need Internet access to be functional; the second one is to tunnel traffic to another place, in order to gain anonymity, again, this solution cannot handle the vulnerability of side-channel attack; and the last one is the traffic shaping, this technique remains communication in a constant traffic rate to camouflages the traffic spikes to protect against the side-channel attack.

Besides attacks from the third-party, it is also possible to perform attacks to the physical network. The majority of the existing routers provide Wireless Local Area Network (WLAN) access with password protection protocol. At the initial stage, it used Wired Equivalent Privacy (WEP) as the security protocol. However, WEP had several flaws due to its Stream Cipher - Rivest Cipher 4 (RC4) for confidentiality and Error Detection - Cyclic Redundancy Check 32 (CRC-32) checksum for integrity. The authors of [1] provided several attacks to discover the Initialization Value (IV) used on the WEP. WEP was replaced with the WPA and WPA2, new password protection protocols.

Even though WPA2 provides a much better protection, it may not be considered completely secure. The authors of [20] presented a brute force attack to get the network password when the WPA2 protocol is used. This attack uses several tools, namely aireplay-ng, airodump-ng and aircrack-ng. The idea is to keep watching traffic from the air, while it sends some deauthentication frame to an access point with the authenticated device’s MAC address. Then, the device will try to reauthenticate to the access point which makes possible to the attacker to obtain the
WPA password handshake traffic. Finally, having the password handshake traffic, it is possible to perform a brute force attack to determine the network password.

The easier way to mitigate this attack is to use a difficult password and change the password often. However, the authors presented two different techniques to mitigate it: the first one is to map all devices’ MAC address into the access point, to prevent access from unknown devices. Nevertheless, this can lead to a MAC address spoofing attack. This is, an attacker may use Macchanger to change the MAC address of the device and authenticate himself as a legitimate device; and the second one is to increase the periodicity of transmitting the beacon frame, this frame serves to announce the presence of the WLAN. Then, it reduces its frame broadcasting frequency and makes very difficult to perform deauthentication flooding attack. More details about the attack are presented by Čisar et al. in [21].

Kolias et al [1] present a list of availability (Table 2.4) and man-in-the-middle attacks (Table 2.5). We will use these lists to evaluate SPYKE in Chapter 4.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deauthentication</td>
<td>Sending a deauthentication frame to the access point in order to deauthenticate the device.</td>
</tr>
<tr>
<td>Disassociation</td>
<td>Sending a disassociation frame to the access point in order to disassociate the device.</td>
</tr>
<tr>
<td>Deauthentication Broadcast</td>
<td>Sending a lot of deauthentication frames.</td>
</tr>
<tr>
<td>Disassociation Broadcast</td>
<td>Sending a lot of disassociation frames.</td>
</tr>
<tr>
<td>Block ACK Flood</td>
<td>Falsify an Add Block Acknowledgment frame, that contains a sequence number. All further frames within this sequence number are dropped.</td>
</tr>
<tr>
<td>Authentication Request Flooding</td>
<td>Sending a lot of authentication request to the access point.</td>
</tr>
<tr>
<td>Fake Power Saving</td>
<td>Sending a null data frame with the Power Save bit field set to 1 to the access point. Then frames are dropped since the device is on power saving mode.</td>
</tr>
<tr>
<td>Clear To Send (CTS) Flooding</td>
<td>Sending a huge CTS to himself or to other making other postpone their frame.</td>
</tr>
<tr>
<td>Request To Send (RTS) Flooding</td>
<td>Sending a huge RTS to himself making other to back-off from transmitting.</td>
</tr>
<tr>
<td>Beacon Flooding</td>
<td>Beacon advertise non-existing Extended Service Set IDentifier (ESSID)</td>
</tr>
<tr>
<td>Probe Request Flooding</td>
<td>Sending a constant stream of fake Probe Request packets to the access point.</td>
</tr>
<tr>
<td>Probe Response Flooding</td>
<td>Sending flood of fake and inaccurate Probe Responses to the device.</td>
</tr>
</tbody>
</table>
### Table 2.5: Man-In-The-Middle attacks

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honeypot</td>
<td>Free access point in order to attract the connections.</td>
</tr>
<tr>
<td>Evil Twin</td>
<td>Fake access point advertises the same ESSID as the access point.</td>
</tr>
<tr>
<td>Rogue Access Point</td>
<td>Install a backdoor in access point open for outsiders.</td>
</tr>
</tbody>
</table>

#### 2.3 Summary

In this Chapter, we started by presenting the concept of a middlebox, an intermediary located in the middle of communications between a user and a remote server with the ability to inspect and modify the network traffic. And we presented one example of a middlebox, i.e., proxy, and survey of existing proxies. Afterwards, we presented a private network environment where a middlebox can be located, i.e., a smart home. We provided information about utilities of smart devices following with smart hubs in a smart home. Then, we discussed the existing security monitors and their importance in a private network environment. We finished by presenting network attacks such as side-channel and fingerprinting attacks [19], and DoS and Spoofing attacks [1].
Chapter 3

SPYKE

In this Chapter, we introduce SPYKE, a prototype system situated in the middle of the communication between devices and the Internet. Section 3.1 explains the goals that were established for our system. Section 3.3 presents the design of SPYKE, highlighting its functionalities and architecture. Finally, Section 3.4 describes the implementation of the prototype system.

3.1 Goals

At the early stages of development, our intention was to develop SPYKE as a proxy. However, while surveying the existing proxies and solutions, we concluded that implementing a network proxy would lead to leakage of some important information. Because a proxy intercepts and handles particular types of communications based on the protocol, such as HTTP and SOCKS. Therefore, we decided to implement a gateway that belongs to the network layer of OSI model with the capability to handle and inspect the Internet Protocol of the communications.

The proposed system has the main goal to be similar to a classical gateway, an intermediary situated in the home environment, managing the connection between devices and the Internet. However, instead of making all devices change their default gateway or DNS to the SPYKE gateway, devices are directly connected to the SPYKE gateway via wireless.

The proposed gateway has the ability to forward packets from devices to the Internet as well as the ability to receive packets from the Internet to devices. In addition, it needs to be a transparent system in order to be easy for the user to use. For the security purpose, the proposed gateway cannot be accessible by everyone. In addition, SPYKE should control uploading packets and able to know where packets are going. Moreover, the proposed system should provide privacy to the home environment against several known network attacks. One typical privacy violation mentioned above is, for example, after devices are successfully authenticated, they may somehow
become under control of an attacker and upload data to an unknown and undesired third-party.

### 3.2 Attacker model

For a smart home device to upload data to the Internet through SPYKE, it has to possess a valid password and the explicit user permission. An attacker may exploit this feature by having a valid password and having the device identity. In this model, we consider the following capabilities to model different types of attackers:

- **A1** Record any Ethernet frames on the air.
- **A2** Inject Ethernet frames with a given source and destination MAC addresses.

Capability A1 can be acquired by an attacker by snooping anywhere near SPYKE using a network interface with the monitor mode ability. A2 can be acquired by using a network interface with the packet injection ability. By having the capability to record and inject Ethernet frames the attacker may perform several attacks:

- **B1** Disconnect the legitimate devices.
- **B2** Discover the network password.
- **B3** Spoof the legitimate device.

Capability B1 can be performed by having the MAC addresses of legitimate devices and of SPYKE. B2 is very hard to acquire due to the fact that it needs to perform password guessing. Having the legitimate device’s MAC address and the valid password, the attacker gains the B3.

### 3.3 Design

To achieve the goals mentioned above, the functionalities of the prototype system are presented in Section 3.3.1 and its architecture in Section 3.3.2.

#### 3.3.1 Funcionalities

The prototype system is a Plug-and-Play (PnP) system, i.e., it only requires a pre-configuration to be functional in order to provide transparency to the user.

The prototype system is not connectable by everyone. It requires password authentication to accept the connection of new devices to the network. However, even after devices are authenticated with the correct password the connection further requires the user’s intervention to grant
permission. This means that a new device is defined by default as blocked, effectively dropping all connections until the user defines the boundaries of that device.

In addition, SPYKE provides a level of control on the uploading of packets, i.e., it controls the quantities of packets with respect to the data size by giving the user the ability to configure the maximum bandwidth and maximum transfer quota per a defined period of time.

After permission is granted, SPYKE also presents to the user with all the destinations where devices have uploaded data.

3.3.2 Architecture

Regarding the system architecture, represented in Figure 3.1, it is subdivided into two modules: the first one is authentication, where devices authenticate with the gateway in order to make requests; the second one is user policy enforcement that SPYKE uses for providing privacy protection. Figure 3.1 presents an overview of the entirety of the proposed system.

![Figure 3.1: SPYKE proposed architecture](image)

**Authentication**

A device needs to authenticate itself to prove that it belongs to the home network. The SPYKE gateway performs the wireless authentication using WPA2 as the password authentication protocol. This means that a device can authenticate itself by showing it knows the WiFi password. However, knowing the password of a home network can be an easy task for an attacker, either because the gateway is using a default password, or because the user simply shared the WiFi password. So the device permission to access the Internet is not only controlled by knowledge...
of the password, but also the explicit user’s approval on the user interface, i.e., whitelist access policy.

To distinguish all connected devices, the gateway uses a DHCP server to assign a different IP address as the identity to each authenticated device. The IP address assignment is based on the unique MAC address provided by devices.

Data Processing

After the authentication is established, and the permission granted by the user, data that is uploaded and downloaded by a device passes through the gateway. This data processing is represented into two processes: Analysis and User policy.

Analysis

Data is analyzed at the first step by understanding the packet metadata, e.g., the sender and receiver IP addresses and content size. Then, the user has access to see connected devices, and destination IP addresses of connections made by each device. The proposed system allows the user to check how much data has passed and how much had been dropped.

User Policy

Regarding the user policy, the user may define which device has the access to the Internet, how many data the device can upload, and the available bandwidth. After the data analysis, the proposed system compares data with the user defined policies. It drops packets that came from unknown devices and ones that are previously blocked by the user. In addition, it drops also traffic that exceeded the maximum transfer quota or bandwidth defined by the user. Finally, all allowed traffic goes to the Internet.

3.3.3 Summary

To summarize, SPYKE has functionalities to monitor a home environment network by inspecting outgoing traffics from devices to the Internet. SPYKE is an easy to use PnP system. It provides WiFi access with WPA2 as the password authentication protocol. In order to give the user control of his data, it provides the user the ability to define values of bandwidth and maximum transfer value. SPYKE informs the user through the user interface when a new device is connected, the outgoing data of each device, and destinations that have been detected so far. SPYKE does not have a filtering ability to filter packet content, rather it can block packets that violate the rules enforced on the firewall.
3.4 Prototype Implementation

The SPYKE prototype requires intervention from the user. Whenever a device tries to connect to SPYKE, it should authenticate itself providing the correct password. Once a device is authenticated to the gateway, its information like unique MAC address, assigned IP address, and hostname is recorded and presented in the user interface. If the user approves the device, the gateway starts to give the device access to the Internet. After that, it starts recording the packet’s destination IP address and total packets have been sent within a period of time.

![Diagram](image)

Figure 3.2: Block device from uploading data to an undesired third-party

Figure 3.2 presents the communication between a device and a remote server through the gateway. The user is connected to the gateway through eth0 in subnetwork 192.168.1.0/24 while devices are connected via wireless in subnetwork 192.168.8.0/24. After a while, the user may access the user interface and see the destination IP address accessed by a device and the total uploaded bytes, and then he may decide to set a maximum transfer quota per a period of time and/or set a maximum bandwidth value, or even block the device access.

3.4.1 Device Life Cycle

Figure 3.3 shows the lifecycle of a device in SPYKE. When a new device is detected by the system, its status becomes “NEW” and the information is stored in the database. Then the user, using the user interface, may allow or block the device. In the first case, the device’s status is changed to “BLOCKED” and the information is stored in the database. For the second case, the device’s status is changed to “ALLOWED”, the information is stored in the database and in addition a period is created and stored to the database and `iptables`’ rules are created and added to `iptables`.

Whether the device is “ALLOWED” or “BLOCKED”, the user may still allow or block the device but never change back to “NEW”. Furthermore, when the device’s status is set to
“ALLOWED”, it occurs two following cases: the first case, the user may change the device’s maximum bandwidth, maximum transfer quota, or period, hence the device is updated, information is stored in the database and the `iptables` rules are updated; the second case is whenever the end of periods is achieved, `iptables` registers are extracted and stored in the database within the period and a new period is created.

3.4.2 Data Model

Figure 3.4 shows the Data Model, composed by two tables, Device and Period, with an one-to-many relationship. In the Device’s table, the id is the MAC address that is considered as the primary key. In the Period table, the id is composed by start and end time, and the associated MAC address from the device, creating a composite key.

When a new device is detected, SPYKE stores information containing variables like the MAC address and name provided by the device, the IP address provided by DHCP server, the status as new, quota and bandwidth values as zero (0) with corresponding unit as KiloByte (KB), and the period as zero with corresponding unit as minute (m). After the user changes the status of the device to “ALLOWED”, a Period is created and stored with the associated device’s MAC address, the start time and end time. For example, if a device is registered at 6:00 with a period of 5 minutes, the value of `start_time` will be 6:00 and the value of `end_time` will be 6:05. If the period is not defined, then it will be set to 1 hour by default.
3.4.3 Data Flow

Figure 3.5 represents the data flow of the entire system. First, devices authenticate themselves and obtain an IP address from DHCP server provided by dnsmasq\(^1\). The engine stores the device information in the database and waits for the user’s approval. After the user’s approval, the engine adds rules on iptables allowing the access of the device to the Internet, and adds the defined period to the In-Memory data storage that relies on main memory of the computer data storage.

The periods of each device are stored in the database whenever it achieves the end of the period, and then new periods are created, and iptables’ rules are renewed. Meanwhile, the user can access the information about bytes allowed and dropped by each period. The user can also decide whether to allow or block any devices as well as modify the value of period or limit.

3.4.4 Hardware

The proposed system was implemented on a Raspberry Pi 3b+\(^2\) with Raspbian Stretch Lite as the Operating System.

The Raspberry Pi 3B+ uses Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC, providing a processor with a frequency of 1.4 GHz, and uses LPDDR2 SDRAM that provides 1 GB of RAM. The Raspberry Pi 3B+ provides a network interface card with protocols IEEE 802.11.b/g/n/ac WLAN (wlan0) that provides 2.4 GHz and 5 GHz bands. It also provides

\(^1\)http://www.thekelleys.org.uk/dnsmasq/doc.html accessed on April 8, 2019
\(^2\)https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ accessed on April 8, 2019
another network interface via cable with Gigabit Ethernet (eth0) up to 300 Mbps.

Figure 3.6 shows a running Raspberry Pi 3b+ within a case. The off-the-shelf hardware is connected with power cable and Ethernet cable. In addition, it provides wireless connections acting as an access point.

Despite being implemented into a Raspberry Pi, it can be implemented in any off-the-shelf hardware that has two network interfaces, one for providing the network to the devices and another to provide Internet access.

### 3.4.5 Third-Party Tools

To make the proposed system work, we used several third-party tools, namely: hostapd, dnsmasq, iptables, Java, and SQLite.

**hostapd**

With the help of host access point daemon or simply hostapd, it is possible to provide a wireless connection between the proposed system and multiple devices. Hostapd provides the capability to turn a normal network interface card into an access point with authentication method. Table 3.1 presents characteristics of hostapd configuration file. Robust Security Network (RSN) is a security network that only allows to create strong network associations and Counter Mode Cipher Block Chaining Message Authentication Code Protocol (CCMP) is an encryption protocol.
Figure 3.6: Raspberry Pi 3b+ within a case

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Set</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>wlan0</td>
<td>The wireless interface</td>
</tr>
<tr>
<td>Driver</td>
<td>nl80211</td>
<td>Configuration management for wireless devices</td>
</tr>
<tr>
<td>SSID</td>
<td>spyke</td>
<td>Service Set IDentifier name</td>
</tr>
<tr>
<td>hw_mode</td>
<td>g</td>
<td>Protocol IEEE 802.11g with band of 2.4 GHz</td>
</tr>
<tr>
<td>channel</td>
<td>7</td>
<td>Wireless channel number used</td>
</tr>
<tr>
<td>beacon_int</td>
<td>500</td>
<td>Beacon frame broadcast interval in millisecond</td>
</tr>
<tr>
<td>wpa</td>
<td>2</td>
<td>Version of WPA</td>
</tr>
<tr>
<td>wpa_passphrase</td>
<td>spyke2018</td>
<td>The password used to authenticate</td>
</tr>
<tr>
<td>wpa_key_mgmt</td>
<td>WPA-PSK</td>
<td>WPA-Personal mode is design for home network</td>
</tr>
<tr>
<td>rsn_pairwise</td>
<td>CCMP</td>
<td>RSN pairwise cipher using CCMP</td>
</tr>
</tbody>
</table>

Dnsmasq

The proposed system uses dnsmasq to deploy DNS server that converts IP addresses to domain names, and vice versa, and DHCP server that provides identity to each connected device by assigning a different IP address and saves device’s information such as MAC address and hostname.

Dnsmasq is a lightweight network infrastructure service. Table 3.2 shows the configuration characteristics of dnsmasq. The gateway uses interface wlan0 that listens at the 192.168.8.1, For the purpose of a home network, we configured DHCP to receive 254 devices by assigning IP addresses between 192.168.8.2 and 192.168.8.255. We also configured the lease time to infinite in order to assign the same IP address to the same device. The gateway uses the Google DNS
service because it has a good performance and it has georeplicated servers all over the world. It is also secure because it assures that requests for domain name resolution are verified and signed.

We configured 3.2 to run a script whenever a device connects. This script stores each connected device’s information such as the assigned IP address, the unique MAC address, and the device’s name in the database.

Table 3.2: Example dnsmasq configuration

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
<td>wlan0</td>
<td>Listen interface</td>
</tr>
<tr>
<td>listen-address</td>
<td>192.168.8.1</td>
<td>Listen address</td>
</tr>
<tr>
<td>dhcp-range</td>
<td>192.168.8.2,192.168.8.255, 255.255.255.0, infinite</td>
<td>IP address range for devices with subnet /24, and infinite lease time</td>
</tr>
<tr>
<td>dhcp-option</td>
<td>3,192.168.8.1</td>
<td>Set 192.168.8.1 as router</td>
</tr>
<tr>
<td>dhcp-option</td>
<td>6,8.8.8.8</td>
<td>Set 8.8.8.8 as DNS server</td>
</tr>
<tr>
<td>dhcp-script</td>
<td>path to script</td>
<td>execute script.sh whenever a device connects</td>
</tr>
</tbody>
</table>

iptables

Iptables is available by default with the Linux kernel firewall. It has the ability to block the IP address by rules, or chain of rules. In addition, it is also able to control the outgoing and incoming traffic by limiting the number of bytes per second, and/or the maximum number of bytes per an IP address or a subnetwork. There are also rules that allow logging packets metadata information for further inspections.

By default, it is set to block all connections. We added two rules for incoming and two rules for outgoing to allow traffic with UDP and port 67 and 68, which is used for DHCP server and DHCP client successively. This allows devices to connect to the system through DHCP exchange. We also added one rule for incoming and another for outgoing to allow traffic with UDP and port 53 to allow devices to retrieve domain name by DNS server. The six mentioned rules are set with a limit average of 10 connections per second, i.e., when the incoming or outgoing packets exceeded the defined average, it starts to drop the packets in order to prevent DoS attack.

Figure 3.7 provides chain of rules. Whenever device’s status is set to “ALLOWED” in the user interface, rules are added to the iptables. If the bandwidth value is zero or not defined, it adds four rules the iptables. Otherwise, it adds five rules. For the first case, the first rule is to accept all forward download packets, i.e., from the Internet to the device. The second rule is to accept forward uploading packets limited by a defined value of size, if the value is zero, then it is unlimited. The third rule is to drop all forward uploading packets, i.e., when uploading packets size exceeded or become near to the defined value, all further packets start to
be dropped. Finally, the last rule is to drop all forward download packets.

<table>
<thead>
<tr>
<th>RULE</th>
<th>TRAFFIC</th>
<th>DIRECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACCEPT</td>
<td>DOWNLOAD</td>
</tr>
<tr>
<td>2</td>
<td>ACCEPT</td>
<td>UPLOAD</td>
</tr>
<tr>
<td>3</td>
<td>DROP</td>
<td>UPLOAD</td>
</tr>
<tr>
<td>4</td>
<td>DROP</td>
<td>DOWNLOAD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RULE</th>
<th>TRAFFIC</th>
<th>DIRECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACCEPT</td>
<td>DOWNLOAD</td>
</tr>
<tr>
<td>2</td>
<td>DROP</td>
<td>EXCEEDED</td>
</tr>
<tr>
<td>3</td>
<td>ACCEPT</td>
<td>DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>DROP</td>
<td>UPLOAD</td>
</tr>
<tr>
<td>5</td>
<td>DROP</td>
<td>ALL</td>
</tr>
</tbody>
</table>

Figure 3.7: Chain of rules of “ALLOWED” device with and without bandwidth defined

In case bandwidth is set, an extra rule is appended in the second position of the `iptables` chain. This rule drops all forward uploading packets that exceeded defined bandwidth value.

Java

We used **Java** as the programming language. The Java Development Kit (JDK) used is Oracle JDK instead of OpenJDK, because it is better regarding with performance and it puts more focus on the stability. The version is Java Standard Edition (Java SE) 8 Long-Term-Support (LTS) version. The build process and dependency management are handled by Maven, and Spring is used to provide a user interface via browser.

SQLite

**SQLite** is a relational database management system. We used it as the database because it is a lightweight Structured Query Language (SQL) database, and it is embedded into the system program. Beyond that, the program can execute SQLite database without it being installed in the system, by importing the SQLite dependency using maven.

3.4.6 Summary

To summarize, SPYKE has the following functionalities: it forwards and receives traffics between devices and the Internet; the user only needs to plug the Ethernet cable and the power cable

---

5Maven is a build automation tool used primarily for Java projects: [https://maven.apache.org/](https://maven.apache.org/) accessed on April 8, 2019.
6The Spring is an application framework for the Java platform: [https://spring.io/](https://spring.io/) accessed on April 8, 2019.
7SQLite is small, fast, self-contained, high-reliability, full-featured, SQL database engine: [https://www.sqlite.org/index.html](https://www.sqlite.org/index.html) accessed on April 8, 2019.
to the gateway and the system is ready to be used (PnP); to access the Internet, devices need to authenticate by having the correct password and be granted access by the user; whenever an unknown device connects to the gateway, it is blocked by default, and the user can then confirm the block by defining its state to “BLOCKED”; SPYKE can also limit the number of packets sent by defining a maximum transfer quota and a bandwidth value.

The SPYKE prototype was deployed in a Raspberry Pi with third-party tools, hostapd, dnsmasq, iptables, Java, and SQLite.
Chapter 4

Evaluation

In this chapter, we present the results of the experiments that were done to evaluate the SPYKE prototype. We conducted several experiment sets, each one with a specific goal. The initial experiments were related to performance. Smart home devices were represented by a Raspberry Pi that sends data to another Raspberry Pi via gateway with and without using SPYKE. This was done to measure the overhead introduced by the system. The second set of experiments used real IoT devices, such as smart plugs or smart speakers, also to measure the performance of the system. The third set of experiments were related to security. We tested if SPYKE is indeed effective in blocking connected but undesired devices. In addition, we verified if a device can somehow violate the rule and send more data than is permitted to the Internet. Then, the final tests reproduced flooding attacks. The goal of these experiments was to evaluate the efficiency of the prototype system (4.1); evaluate the effectiveness of data limit rules (4.2); and finally, to evaluate the effectiveness against a selection of known attacks (4.3).

4.1 Performance Evaluation

The iPerf1 measurement tool was used for performance evaluation, to check maximum achievable bandwidth and data transferred per second.

We needed a Server and a Client in order to use iPerf. Therefore two more Raspberry Pi 3b+ were needed to execute as an iPerf Server and an iPerf Client, while the proposed gateway was located between them.

Figure 4.1 shows the experimental structure using iPerf. SPYKE has two interfaces, one for wireless (wlan0) interface with the IP address as 192.168.8.1 and another for ethernet (eth0) interface with the IP address as 192.168.1.67.

1https://iperf.fr/ accessed on April 13, 2019
The communication between SPYKE and the Raspberry Pi 3B+ that is running iPerf Server is within a private network provided through a router, named Fiber Gateway. This router provided by an ISP is used due to the fact that commercial devices in the next experiments need to have access to the Internet.

The Fiber Gateway was connected to SPYKE through an Ethernet cable, with a speed up to 1 Gbps, as well as to the Raspberry Pi 3B+ that is running iPerf Server. Since Raspberry Pi 3B+ has a Gigabit Ethernet up to 300 Mbps, we conducted a first evaluation using iPerf to check the real speed, and we obtained an average speed of 250 Mbps.

### 4.1.1 Baseline Performance

For a better understanding of the base performance, we experimented first without SPYKE running on the gateway.

Each experiment using iPerf from the Client to the Server had a duration of 200 seconds. The results are transformed into a graphical representation using Matplotlib.

To quantify the random errors in measurements, the program runs have to be repeated several times. In fact, we used at least 30 runs, so that calculation can assume a normal distribution of the samples, according to the Central Limit Theorem.

Figure 4.2 represents the first result of the set. It shows us that it sent 1002 MB during 200
seconds, which means it has an average of 5 MBps.

Figure 4.2: Outgoing communication measurement without SPYKE

4.1.2 Overhead

The same environment and structure as mentioned above were used to measure and evaluate the SPYKE overhead. In this step though, we have SPYKE running.

Figure 4.3 compares the first experimental evaluation result with and without SPYKE running in a graphical representation. At the beginning, the bandwidth remains similar, but as time progresses the SPYKE performance increases.

Figure 4.3: Comparison of the first evaluation with and without SPYKE running
Device Number Increase

The system should be scalable, so there were also experiments with an increasing number of
devices. The devices were represented through added rules on the `iptables`. Each device adds
five rules on the `iptables` as mentioned at the Section 3.4.

Once again all the experiments had the same experimental environment and structure. The
system was tested with rules corresponding to 100, 1000 and 10000 devices.

Figure 4.4 shows the experimental evaluation of thirty measurements with different number
of device rules. Table 4.1 and Table 4.2 show the average of the total transfer and bandwidth
of thirty experiences consecutively. This table compares all the average of all experimental
evaluations that have been done for evaluating the system’s performance. We see that with
1 device, SPYKE transferred more data and has larger bandwidth than the baseline without
firewall. We believe that this difference exists because SPYKE blocks connections by default,
so it increases performance slightly by forwarding only the allowed connections.

![Figure 4.4: Comparison of the first evaluation with different device number](image)

<table>
<thead>
<tr>
<th></th>
<th>Total Transfer (MB)</th>
<th>Minimum Total Transfer (MB)</th>
<th>Maximum Total Transfer (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>1010</td>
<td>1001 (-09)</td>
<td>1016 (+06)</td>
</tr>
<tr>
<td>1 device</td>
<td>1045</td>
<td>1020 (-25)</td>
<td>1064 (+19)</td>
</tr>
<tr>
<td>100 devices</td>
<td>1020</td>
<td>1002 (-18)</td>
<td>1024 (+04)</td>
</tr>
<tr>
<td>1 000 devices</td>
<td>1023</td>
<td>1022 (-01)</td>
<td>1024 (+01)</td>
</tr>
<tr>
<td>10 000 devices</td>
<td>980</td>
<td>968 (-12)</td>
<td>993 (+13)</td>
</tr>
</tbody>
</table>
Table 4.2: Average bandwidth results for the performance experiments. Thirty runs were performed for each experiment.

<table>
<thead>
<tr>
<th></th>
<th>Bandwidth (MBps)</th>
<th>Maximum Bandwidth (MBps)</th>
<th>Minimum Bandwidth (MBps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>5.05</td>
<td>5.08 (+0.03)</td>
<td>5 (-0.05)</td>
</tr>
<tr>
<td>1 device</td>
<td>5.23</td>
<td>5.32 (+0.09)</td>
<td>5.1 (-0.13)</td>
</tr>
<tr>
<td>100 devices</td>
<td>5.1</td>
<td>5.12 (+0.02)</td>
<td>5.01 (-0.09)</td>
</tr>
<tr>
<td>1 000 devices</td>
<td>5.115</td>
<td>5.12 (+0.005)</td>
<td>5.11 (-0.005)</td>
</tr>
<tr>
<td>10 000 devices</td>
<td>4.9</td>
<td>4.96 (+0.06)</td>
<td>4.84 (-0.06)</td>
</tr>
</tbody>
</table>

100 Devices: For the 100 devices, we added 500 rules to the `iptables`. In comparison with the experiment with 1 device, the average of total transfer dropped 25 MB and bandwidth dropped 0.18 MBps. However, the result shows that the performance remains above 1000 MB and 5 MBps.

1 000 Devices: For the 1 000 devices, we added 5000 rules to the `iptables`. In comparison with the previous one, the average of total transfer slightly increased 3 MB and bandwidth increased 0.015 MBps. Once again, the result indicates that the performance remains above of 1000 MB and 5 MBps.

10 000 Devices: For the 10 000 devices, we added 50000 rules to the `iptables`. In comparison with all the previous experiments, the average of total transfer dropped to 980 MB and bandwidth dropped to 4.9 MBps. The results show that the performance falls below 1000 MB and 5 MBps.

Discussion

To summarize, the results obtained show that the used Raspberry Pi maintains a good performance whether SPYKE is running or not. However, with SPYKE running, the performance increased in terms of bandwidth. We believe the reason of this increase is that SPYKE blocks connections by default, so it increases the performance by dropping all other connections and forwarding only the allowed connections.

By increasing the number of `iptables` rules, especially in the case of 10 000 devices, the system started dropping the performance. Nevertheless, for a smart home environment, SPYKE is capable of handling a very large number of devices.

4.1.3 Commercial Devices

SPYKE is also able to handle commercial devices, we used a Smart Plug from TP-Link and a Smart Speakers from Amazon to perform the experiences. After the WiFi configuration phase
of both devices, we set status “ALLOWED” and put a large maximum limit for experiments.

Each device has rules that stated 100 GB of maximum transfer quota, 100 MBps of maximum bandwidth value and period of 1 Hour. The reason for these high values was to make sure that devices are not limited by rule enforcement that is discussed further on.

Smart Plug

Figure 4.5 shows the values of the device TP-Link smart plug model HS110\(^6\) obtained from `iptables` log and converted by Matplotlib. The device has the IP 192.168.8.70 provided by DCHP.

\[\text{Figure 4.5: TP-Link Smart Plug HS110 outgoing communication measurement}\]

The result presented the experiments during 3565 seconds, which is around one hour. As we can see, it remains constantly on 160 bytes of upload. By analyzing the result log, it was sending the awake message.

After the 3400 seconds pointed in the figure the communication pattern changes because the device was being accessed through Kasa and checking the smart plug’s data. The result achieved more than 2 KB transmission peak, due to their authentication method. Then, as the application is retrieving the information in real time, it transfers data at a peak of 500 KBps.

Smart Speaker

Amazon Echo Dot 3\(^7\) is a voice-controlled smart speaker provided by Amazon with the Alexa voice interface. Figure 4.6 represents the result obtained from `iptables`' log and converted by Matplotlib. The device has the IP 192.168.8.64 provided by DCHP as shown in the sender IP. Moreover, the result shows a lot of peaks. After analyzing the logs, it seems that the Amazon Elastic Compute Cloud (EC2) web service is constantly changing the remote server’s IP address, which makes the Amazon Echo to send authentication messages to the remote server whenever its IP address changes. In addition, it also sends constantly awake messages.

![Figure 4.6: Amazon Echo outgoing communication measurement](image)

Figure 4.7 shows results during voice interactions with Alexa listening to the Amazon Echo device. At the 2400 seconds mark, three questions were asked: the first one was “Alexa, what time is it?”; followed by “Alexa, how is the weather today?”; the last one was “Alexa, play BBC Radio 1”. For the first two questions, Alexa answered correctly and stopped. For the last one, it started to play Radio as asked.

While Alexa is playing the radio, it uploads additional awake packets to the remote server to inform that the communication is still established.

4.1.4 Discussion

The performance experiments show that the prototype overhead is very small, that a large number of device rules are supported and we verified the operation with two commercial devices.

\(^7\)https://www.amazon.com/All-new-Echo-Dot-3rd-Gen/dp/B0792K2BK6 accessed on April 8, 2019
Regarding the performance of real devices, it worked well as expected, i.e., all the device functionalities were preserved, even though they were being intermediated by SPYKE.

4.2 Rule Enforcement

Two important features of SPYKE are blocking upload traffic from unknown devices by default, and allowing the user to set limits on the usage for known devices. The following experiments validated the correct enforcement of the communication rules.

After the performance evaluations, we defined rules to limit the bandwidth as well as quota for a defined period of time for all upload data. The default period time is 1 Hour.

4.2.1 Quota

The upload quota defines the maximum amount of data that can be transferred during one period. Regarding the limit of maximum transfer per period, several experiments were performed on the Amazon Echo Dot 3 with 200 KB as the quota during 5 minutes and no limitation on bandwidth. During this period of time, the test user asked about the current weather and time.

Figure 4.8 shows the experience for 969 seconds, which is more than 16 minutes, and covers four periods. The 200 KB limit over 4 periods was enforced and a total 797 KB of data was transferred as expected.
4.2.2 Bandwidth

In terms of the bandwidth, we performed a similar experiment as before with the Amazon Echo, but this time, the bandwidth was set to 10 KBps, a period of 5 minutes was defined and there was no quota limitation.

Figure 4.9 represents the result for 600 seconds, which are equivalent to 10 minutes. Even with the bandwidth set to 10 KBps, the results come close to 20 KBps due to the packet burst. Basically, it may contain extra bytes from the previous second, for example, if the current second sent 0 KBps, then in the next second, it may achieve almost the double of the set value.

4.2.3 Quota and Bandwidth

Finally, we limited the bandwidth and quota at the same time. Again, we set a 200 KB limit within a period of 5 minutes and a bandwidth of 10 KBps. To understand better the combination of these two rules, we made some calculation\(^8\) to see if they interfere with each other. As a matter of fact that 10 KBps of bandwidth allows transferring 3000 KB within 5 minutes, it is possible to do tests by limiting 100 KB per 5 minutes.

As expected, according the previous two figures, figure 4.10 shows that bandwidth never achieved more than 20 KBps and Amazon Echo sent 401 KB within two periods. After analyzing the result, the extra 1 KB was caused by the last packet allowed in the first period. This packet contained 1500 B.

---

\(^8\)10 KBps = 10 \times 60 \times 5 = 3000 KB per 5 minutes
4.2.4 Discussion

After setting a quota and limiting bandwidth with some deviation in terms of bandwidth, data control still works as expected. Furthermore, the user may be aware of a possible malfunctioning device, which behaves strangely and uploads data to undesired destination and able to block it. The rule enforcement works as expected.

4.3 Effectiveness against attacks

In this section, we present the experiments done to evaluate the effectiveness of the SPYKE prototype against well-known attacks. The assessment started with a list of availability attacks compiled by [1], e.g., DoS attacks, such as Deauthentication Broadcast and Disassociation Broadcast attacks, and Authentication Request Flooding.

SPYKE uses WPA2 as the password security protocol. However, any password security protocol prone to brute force attack. Indeed, WPA2 is vulnerable to dictionary attacks. Another relevant attack is Spoofing attack.

For this purpose, a Raspberry Pi was used again with the aircrack-ng\textsuperscript{9} tool installed to assess network security. In addition, an external network interface, Alfa network AWUS036NHA\textsuperscript{10} was used in order to perform the aircrack-ng, because it has monitor mode and is able to do packet injection. Figure 4.11 shows Alfa network AWUS036NHA.

\textsuperscript{9}https://www.aircrack-ng.org/ accessed on April 12, 2019
\textsuperscript{10}https://www.alfa.net.my/webshaper/store/viewProd.asp?pkProductItem=15 accessed on April 12, 2019
For checking the effectiveness of SPYKE, the following attacks were performed: Deauthentication and Disassociation, Authentication Request Flooding, Dictionary, and Spoofing attacks.

### 4.3.1 Deauthentication and Disassociation

Deauthentication and Disassociation attacks have the goal of ending a connection established between a device and an access point.

We activated the external interface with the monitor mode and used `airodump-ng` to capture all the reachable traffic in the air. Then, we performed the attack by injecting deauthentication frame using `aireplay-ng` with the SPYKE and device’s MAC addresses, then the device lost the connection. As expected, the device lost the connection.

Furthermore, we were able to perform the Deauthentication Broadcast attack, which consists in sending an unlimited number of deauthentication frames to make SPYKE busy and ignore other devices. It was not able to identify the attack. However, other devices with status “ALLOWED” did not lose the connections and remained connected to the Internet.

The same experiment was tested again by setting 500 ms as the beacon interval (100 ms by default), which is a solution mentioned by [20]. With some difficulty, `airodump-ng` was again able to find the MAC address of the connected devices and the access point.

Since the MAC address of SPYKE and the connected devices are disclosed, it was not able to defend against the Deauthentication attack, i.e., a device with status “ALLOWED” will not be able to connect to the Internet while the attack is performing. Though, other devices are still connected and have the Internet accessed, in case they are allowed. For this reason, the attack
is not completely effective against SPYKE.

4.3.2 Authentication Request Flooding

The Authentication Request Flooding attack is executed with the goal to exhaust, slowdown, or even freeze an access point.

Fake authentication cannot be performed by aireplay-ng\textsuperscript{11} to an access point that uses WPA2 as the password security protocol. So we used the mdk3\textsuperscript{12} tool to perform fake authentication request.

The tool can send fake authentication requests to all access points in the range, but for our purpose we specified the access point by its MAC address.

To prevent devices disconnecting from the system, we added rules on iptables to limit 10 packets per second on the UDP connection on the ports 67 and 68. The result showed that SPYKE did not disconnect devices. So, we considered that SPYKE is able to protect against this attack.

4.3.3 Dictionary

A Dictionary attack is performed to find out the password of an access point. In most cases, it is used to crack WPA and WPA2 encryption schemes.

First, we ran airodump-ng to capture the packets. Then we deauthenticated a connected device. Once the device is disconnected, it will try to reconnect to the access point by exchanging

\textsuperscript{11}https://www.aircrack-ng.org/doku.php?id=aireplay-ng accessed on April 13, 2019

\textsuperscript{12}https://tools.kali.org/wireless-attacks/mdk3 accessed on April 13, 2019
handshake packets. Meanwhile, **airodump-ng** keep running and capturing packets which are stored in a pcap file. Finally, we used **aircrack-ng** with lists of password as input to the handshake packets. If the list contains the correct password the attack is accomplished.

There are several ways of preventing this attack, e.g., use a strong password, which is difficult to be guessed. **SPYKE** is effective against this attack if the WiFi password is well defined.

### 4.3.4 Spoofing

A Spoofing attack is performed to fool an access point by disguising as a legitimate device.

We used **macchanger** to change the MAC address to the legitimate device. After that, we connected to **SPYKE** with the valid password. Then, we grant the same permission as the legitimate device. Nevertheless, the connection is not stable due to two devices trying to connect **SPYKE** with the same identity.

The proposed solution limits the bandwidth and quota of each device. For this reason, we were not able to send a lot of information. Furthermore, the user is notified, through the user interface, of the destination where devices are connected to. Then, the user can block the device.

Since we can find out the MAC address of any connected device by using **airodump-ng**, we can perform Spoofing attacks to all connected devices. One way to exclude completely the attacker from the network is by changing the WiFi password regularly.

### 4.3.5 Discussion

Regarding the effectiveness against the attacks, **SPYKE** is able to defend against some DoS attacks, namely deauthentication broadcast, dissociation broadcast, and authentication request flooding, but it cannot eliminate DoS attacks on the devices. **SPYKE** cannot protect against the brute force attack on the handshake protocol. However, even if an attacker finds out the correct password and accesses the home network, he has no access to the Internet.

An attacker may falsify his identity by using the MAC address of a legitimate device and upload data to the Internet. In this case, **SPYKE** cannot block it immediately, but it will eventually block it when it overcomes the maximum data allowed defined by the user. In addition, the user can be informed by the user interface that the device is uploading data to an unknown IP address, and block it.

Table 4.3 shows the availability attacks [1] that are within the **SPYKE** protection coverage. The majority of attacks are performed in the data link layer, with the goal to compromise the device connectivity. This means that attacks which are not covered by **SPYKE** are focused on disconnecting devices. The **SPYKE** prototype covered attacks that aim to freeze or slowdown
it, such as Deauthentication Broadcast, Disassociation Broadcast, and Authentication Request Flooding.

Table 4.3: SPYKE availability attacks coverage

<table>
<thead>
<tr>
<th>Attack</th>
<th>Covered by SPYKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deauthentication</td>
<td>×</td>
</tr>
<tr>
<td>Disassociation</td>
<td>×</td>
</tr>
<tr>
<td>Deauthentication Broadcast</td>
<td>✓</td>
</tr>
<tr>
<td>Disassociation Broadcast</td>
<td>✓</td>
</tr>
<tr>
<td>Block ACK Flood</td>
<td>×</td>
</tr>
<tr>
<td>Authentication Request Flooding</td>
<td>✓</td>
</tr>
<tr>
<td>Fake Power Saving</td>
<td>×</td>
</tr>
<tr>
<td>Clear To Send Flooding</td>
<td>×</td>
</tr>
<tr>
<td>Request To Send Flooding</td>
<td>×</td>
</tr>
<tr>
<td>Beacon Flooding</td>
<td>×</td>
</tr>
<tr>
<td>Probe Request Flooding</td>
<td>×</td>
</tr>
<tr>
<td>Probe Response Flooding</td>
<td>×</td>
</tr>
</tbody>
</table>

4.3.6 Summary

The experimental results presented in this Chapter show that SPYKE has a good performance with an overhead is very small, even with real IoT devices. Regarding the scalability, we increased the number of devices by raising the number of `iptables` rules. The system only started dropping the performance after having 10,000 devices. This leads us to conclude that SPYKE is capable of handling a very large number of devices, more than a typical smart home environment has.

Regarding the rule enforcement, the results showed that a device is not able to upload more data than what was allowed. Furthermore, SPYKE was able to show the IP addresses of remote servers that the device had uploaded data, and was able to block them.

About the effectiveness against the attacks, SPYKE successfully defended against DoS attacks, but it cannot prevent DoS attacks on the devices. Although SPYKE cannot detect a malfunctioning device, it is able to inform the user of the communication end-point and the number of upload data of each device.
Chapter 5

Conclusion

This dissertation described the design, implementation, and evaluation of SPYKE, a prototype gateway which stands between IoT devices in a smart home and the Internet. This Chapter presents the achievements of SPYKE and describes future work that can improve SPYKE functionalities and future research possibilities.

5.1 Achievements

SPYKE was designed to be located in a smart home network between smart devices and the Internet. It provides visibility of communications between each device and the Internet. Additionally, SPYKE has the ability to enforce rules to block and limit connections. Furthermore, it is available as an open-source project and deployable in an inexpensive off-the-shelf hardware such as Raspberry Pi.

We evaluated the SPYKE prototype. The results showed good performance and effective rule enforcement. The prototype was able to handle connections even when adding 10 000 devices rules to the system. Beyond that, it was tested with commercial devices like Amazon Echo and TP Link Smart Electrical Plug. The rule enforcement is applicable to all connected devices, upload packets are indeed reduced according to the limitation set. We also did a security assessment covering a set of availability attacks [1] and SPYKE was able to prevent a set of relevant DoS and Spoofing attacks.

5.2 Future Work

Regarding future work, we propose the following items sorted by priority from the most important to the less important.
Intrusion Detection System

The current solution only analyzes the packet header and informs the user about the devices end-point communications and how many bytes a device has uploaded so far. On one hand, it does not alert the user that the mentioned end-point is a threat or not. On the other hand, it does not alert the user if devices are uploading packets with content that is not supposed to. For example, a smart plug uploading video stream.

For a future version, it should have a further destination IP address analysis and packet content analysis. This module can be implemented by an existing open source IDS like Snort, Suricata or Zeek. In addition, it should go beyond knowledge-based rules and add Machine Learning-based anomaly detection in order to detect previously unknown attacks.

Traffic Shaping

One of attacks that the current solution does not protect is side-channel attack. This attack compromises the user’s privacy, because an attacker may figure out which routine the user has by analyzing traffic spikes.

The technique that can protect against this attack is Traffic Shaping [19] that performs a constant traffic rate and camouflage traffic spikes. This can be achieved by using the tc\(^1\) tool along with iptables.

Incoming Traffic

The current solution focused more on limiting and checking the outgoing traffic because we want to block the devices to upload sensitive data to undesired third-parties. However, the same architecture and implementation can be extended to handle incoming traffic.

User Interface

The current solution does not provide authentication for accessing the user interface, so credentials to authenticate should be added.

\(^1\)Linux Traffic Control: https://linux.die.net/man/8/tc accessed by 8 April, 2019
Bibliography


Appendix A

Privacy Protection

Although this work focuses on using middleboxes to protect the privacy of users, there are some other techniques and schemes that can be used to provide privacy protection.

A.1 Techniques

In the current solution the technique used was the firewall, that can filter incoming and outgoing traffic. Nevertheless, this technique has limitations like it only analyses, block and limiting the traffic. Therefore, other techniques such as anonymization and obfuscation that modifies the traffic data can lead to an enhanced privacy protection.

A.1.1 Anonymization

Anonymization belongs to sensitive attribute modification, where the data that contains the user’s identity is hidden. There are several approaches that authors in [22] pointed out. Most of them have protection against external attacks (attacks from outside of the home network) but are def-aware (attackers might know the algorithm or approach being used).

**Tessellation** - an approach based on a region called tile, the users that belong to this its tile sends tile’s location instead of their actual location.

**Microaggregation** - an approach based on the maximum distance to average vector (VMDA), i.e., a center of several users' locations, instead of a region as tile. For example, there are three users with the following locations: User A in (2, 6), User B in (3, 8) and User C in (10, 7). The location that each one of these users sends to the remote server corresponds to the average of the maximum distance of their actual locations which is (6, 7), obtained by (2+10)/2 and (6+8)/2.
**L-Diversity** - an approach based on two stages. At the first stage, it applies VMDA over the spatial dimension and then over the temporal dimension. It also adds random values to the report and organizes into location, time and group ID.

**Data Aggregation** - an approach based on data aggregation function. It uses a slicing technique, i.e., it slices data into various small data, sends to the network where these data is mixed and then aggregated. In addition, it can be used against internal attacks.

**Partial-Inclusivity and Range-Independence** - an approach based on cloaked region, i.e., cloaking of participants’ locations, where reported queries of each device are observed in order to find out if there is any overlap. After that, it aggregates all queries that contain the same information into a single query and then forwards it as the report.

**Social Networks for Preserving Privacy** - an approach based on direct personal contact or recommendation based on common interest or location. In the latter case, the remote server returns a list of subscribed users to the sender for further friendship request, hence data are sent through these users before reach the server. The attacker may intercept the data, but he cannot know who is the actual sender.

### A.1.2 Obfuscation

Obfuscation belongs also to sensitive attribute modification, where the data are modified to hide the user’s identity. There are several approaches that authors in [22] pointed out. Most of them once again have protection against external attacks but def-aware.

**Points of Interest** - an approach that modifies the location by the point of interest. The system divides the area of interest into a set of regions called cells. Then, users receive the distribution of the cell. Finally, the users make reports using the closest cell.

**Position Sharing Approach** - an approach where users split up their precise location into position sharing. Each share is a vector and the server can get precise location by concatenating them.

**Random Data Perturbation** - an approach that modifies the original data adding random noise drawn from a known distribution. To reconstruct the original data is used the iterative algorithm.

**Negative Survey** - an approach that reports fake data value, which is not collected. It is divided into two protocols: the node protocol and the base station protocol. The node
divides data into categories, and then it sends fake categories to the base station where are restored to the original frequency distribution. In addition, it is a def-unaware approach, i.e., the attacker cannot know the algorithm or scheme is being used.

A.1.3 Summary

To summarize, L-diversity, Partial-Inclusivity, and Range-Independence, Social Networks for Preserving Privacy and Random Data Perturbation seem to be more suitable. The first one provides more anonymity than Tessellation and Microaggregation, in addition, it adds random values to the report. The second one which is taking advantage of overlap’s data in order to reduce communication. The third one in order to provide extra anonymity by routing the request through other proxies before reaching the remote server. And the last one modifies the original data in such way that can be reconstructed by the proxy.

A.2 Schemes

The communication between two entities is made usually with information exposed, i.e., when an entity wants to send a message to another entity is difficult to ensure if the destination is the entity wanted, and vice versa, then they need to expose their identity. They risk being intercepted by a third-party.

Vergara-Laurens et al. [22] present their architecture, which is shown in Figure A.1. The authors pointed out four phases of data collecting. The structure starts with devices collecting data from sensors, follows with data aggregation in order to proceed to report. Finally, the data is sent to the server cloud in order to execute the task, i.e., receive data and data storage. There are two challenges that may infringe privacy by collecting data. The first challenge is an entity collects data from another without knowing his identity. The second challenge is the entity who reports data does not reveal his identity. They presented solutions for these challenges. In the first one, it can collect data from a group of entities instead of a single one. In the second one, it is presented two solutions which are attribute modification (anonymization or obfuscation), and non-attribute modification (encryption), as shown in Table A.1.

![Figure A.1: Architecture of Vergara-Laurens et al. [22]](image-url)
Table A.1: Taxonomy of Vergara-Laurens et al. [22]

<table>
<thead>
<tr>
<th>Reporting</th>
<th>Attribution Modification</th>
<th>Non-Attribution Modification</th>
<th>Tasking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymization</td>
<td>Obfuscation</td>
<td>Encryption</td>
<td>No Encryption</td>
</tr>
<tr>
<td>- Tessellation</td>
<td>- Point of interest</td>
<td>- Group signature</td>
<td>- Anonymous authentication</td>
</tr>
<tr>
<td>- Microaggregation</td>
<td>- Position sharing approach</td>
<td>- Double encryption</td>
<td>- Attribute-based authentication</td>
</tr>
<tr>
<td>- L-diversity</td>
<td>- Random data perturbation</td>
<td>- Selective sensing</td>
<td>- Direct tasking without revealing location</td>
</tr>
<tr>
<td>- Data aggregation</td>
<td>- Negative survey</td>
<td>- Mix network and zones</td>
<td>-</td>
</tr>
<tr>
<td>- Partial-inclusivity and Range-independence</td>
<td>- Social network for preserving privacy</td>
<td>- Path cloaking algorithm</td>
<td>-</td>
</tr>
<tr>
<td>- Point of interest</td>
<td>- Group signature</td>
<td>- Anonymous authentication</td>
<td>-</td>
</tr>
<tr>
<td>- Position sharing approach</td>
<td>- Double encryption</td>
<td>- Attribute-based authentication</td>
<td>-</td>
</tr>
<tr>
<td>- Random data perturbation</td>
<td>- Selective sensing</td>
<td>- Direct tasking without revealing location</td>
<td>-</td>
</tr>
<tr>
<td>- Negative survey</td>
<td>- Mix network and zones</td>
<td>- Path cloaking algorithm</td>
<td>-</td>
</tr>
</tbody>
</table>

Although authors present k-anonymity (tessellation and microaggregation) and l-diversity for anonymization, Siddula et al. [23] say that both k-anonymity and l-diversity suffer vulnerabilities, e.g., attribute disclosure, and present a new technique called t-closeness which is a refinement of l-diversity. Siddula et al. pointed out the achievement of node privacy and link privacy.

The first one is using Naive anonymization which is replacing the node (user) with a random number or alphabet in order to achieve the anonymity. The other one uses Edge Perturbation, i.e., an edge (link) between users is either removed or added to introduce noise.

Pandiaraja and Manikandan [24], to overcome existing systems that hide temporal and spatial locality, propose their system, presented in Figure A.2, to protect against attacks such as DDoS (Distributed Denial of Service) by implementing traffic capturing, data extraction, data mapping, and attacker discovery and control. First, the system captures traffic and extracts data based on temporal and spatial locality behaviour, then it uses an interference algorithm for obfuscating. In stream analyzer, the requests are compared the request and the arrival time, if they are different, the request proceeds to the remote server otherwise the request goes to Threshold Based Attack Discovery (TBAD) algorithm with the hash function. Finally, using the TBAD algorithm, it is calculated a threshold based on the difference in time between packets from the same IP address and dropped if the request exceeds the threshold.

Chen et al. [25] proposed ePriLBS, presented in Figure A.3, as a novel efficient privacy-preserving location-based service framework.

They used TTL-free approach that requires either no third-party server or only a semi-trusted one. In their system, the user makes an encrypted request to the proxy (semi-trusted party), showing solely id and location. Then, the proxy forwards the request anonymously to location-based service (LBS) provider, where no information can be learned. The server responds with
redundant information as the union of the results of all k user query to the proxy, hence proxy executes blind filter protocol to filter out redundant POI (point-of-interest) records and return the result to the user.

Rotiroti [26] mentioned the standard authentication’s methods “basic” and “digest” from the HTTP protocol as Request-Response flow in authenticated proxies and two additional authentication’s methods. First one is using smart-card to provide the certificate and another one is using HTTPS URL since it is a secure tunnel, the proxy only forwards the request and the response, as shown in Figure A.4.
Figure A.4: Three authentication architectures mentioned by D. Rotiroti [26]