

POLITECNICO DI TORINO

Dipartimento di Automatica ed Informatica

Corso di Laurea Magistrale in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

**Home IoT devices: analysis of the
communication protocols**



Supervisors:

Prof. Danilo Giordano

Dr. Francesca Soro

Candidate:

Marco Andrea Greco

262624

December 2020

Contents

1	Introduction and problem description	2
1.1	Overview	2
1.2	Motivation	3
1.3	Goals	4
1.4	Organization of the thesis	4
2	State of the art	6
2.1	Overview	6
2.2	Literature	6
2.2.1	Internet Of Things	6
	IoT Applications	8
2.2.2	IoT - Application protocol	8
	Hypertext Transfer Protocol	8
	Constraint Application Protocol	9
	Message Queue and Presence Protocol	9
	Extensible Messaging and Presence Protocol	10
	Data Distribution Service	10
	Advanced Message Queuing Protocol	10
2.2.3	Security	11
	Countermeasures	13
	Cyber-attack detection	13
2.2.4	Honeypot	14
	Low-interaction Honeypot	15
	Medium-interaction Honeypot	16
	High-interaction Honeypot	16

2.3	IoT-oriented honeypots	17
2.3.1	Honeyd	17
	HoneyIo4	17
	IoTPOt	18
	IoTCandyJar	19
3	Architecture and Deployment	21
3.1	System Architecture	21
3.1.1	First setup	22
	SONOFF basic R2 & 4CH PRO R2	22
	Shelly 1	23
	Environment setup	23
3.2	Passive analysis	27
3.2.1	Traffic monitoring	27
	Wireshark	27
	Traffic capture via monitor mode interface.	29
3.2.2	Automate sniffing with Scapy	30
	SSL/TLS encryption	31
3.3	Active analysis	31
	Port scanning	32
	Nmap analysis	33
	Nmap	33
	Automating port scanning	34
3.4	Laboratory environment	35
3.4.1	Mon(IoT)r	35
	Network configuration	35
3.4.2	Devices	36
3.4.3	Packet inspection	36
3.5	Data storage and dictionary structure	39
	Database: RDBMS, no-SQL	39
	MongoDB	40
4	Case study and results	43
4.1	Overview	43

4.2	Active Analysis	46
4.3	Passive Analysis	48
4.4	Data extraction	50
4.5	Summary	52
5	Conclusions and future work	53
5.1	Conclusion	53
5.2	Future work	54
6	Appendix A	55
6.1	System setup	55
6.1.1	Access Point automated setup	55
6.2	Active Analysis	58
6.2.1	Port scanning automation	58
6.2.2	Data extraction	66
6.3	Passive Analysis	76
6.3.1	Network sniffing automation	76

List of Figures

1.1	IoT devices.	3
2.1	Internet Of Things: devices representation.	7
2.2	Portion of the code related to the setup of the credentials used in the brute-force attack. [1]	11
2.3	Diagram showing the dependency between elements in the IT security.	14
2.4	<i>Honeyd</i> sample configuration.	18
2.5	IoTPOT implentation design.	19
2.6	IoTCandyJar architecture.	20
3.1	SONOF basic R2, SONOFF 4CH PRO R2, Shelly-1.	23
3.2	Environment setup.	24
3.3	Man-in-the-middle attack performed by C against A and B. The message M is captured by C and possibly altered as M*.	24
3.4	Decrypted network traffic, captured in monitor mode.	30
3.5	MITM proxy - HTTPS interception.	32
3.6	Mon(IoT)r laboratory - Imperial College London.	35
3.7	List of devices.	37
3.8	Data extraction stateflow.	38
3.9	MongoDB logo.	40
3.10	Structure of the database.	41
4.1	Testbed sample setup.	43
4.2	Spatial information about the servers contacted during the experiments.	44
4.3	Statistical information on the traffic direction of the network divided by protocols used by the devices during the experiments.	45

4.4	Statistical information on network traffic direction.	45
4.5	Statistical information on the protocols used by the devices within the test environment.	46
4.6	Information collected by the automated port scanner.	47
4.7	Part of the information gathered by the port scanner organized in CSV files.	48
4.8	HTTP stream captured during the experiments.	49
4.9	Another fragment of the HTTP stream captured during the experiments.	49
4.10	List of collections of received/sent messages for each device.	51
4.11	Example of a document part of the IoT dictionary.	51

List of Tables

2.1	Return on Investment, and costs related to different types of honeypots.	17
3.1	Devices present in the environment and related interactions tested. . .	37

Abstract

The Internet Of Things is becoming one of the most used technologies today, a trend that will become more pronounced in the near future. The number of interconnected devices is growing rapidly, changing the very shape of the Internet. The IoT is already shown as a pilot of consistent investments for the development of new enabling technologies. However, numerous recent cyber-attacks have highlighted several critical issues in the security control of these systems. The implications of these types of attacks are often unexpected given the heterogeneous and physical nature of these devices. For this reason, basic resources such as honeypots are important, which allow to collect information about new attacks based on vulnerabilities not yet known. This study exposes some general characteristics of IoT devices, analyzing the state of the art of IoT honeypots. The study also proposes an implementation solution for a testbed used to collect messages that are exchanged by home IoT devices triggered by different types of interaction.

Chapter 1

Introduction and problem description

1.1 Overview

The **Internet of Things (IoT)** is a paradigm that is spreading over time in the most varied application contexts, changing and innovating in various technological sectors. One of the biggest challenges faced in the development of the IoT is the cyber-security of these complex, distributed, and pervasive systems. On several occasions, IoT devices have been the subject of various cyber-attacks with dramatically negative implications. The development of new enabling technologies, which suggest a possible extensive development of this technology (5G is just one example), and the security problems that often afflict this technology have attracted the attention of the research community. Attacks conducted against these devices exploit rather simple vulnerabilities and misconfigurations, such as weak authentication. In several cases, infected nodes have become part of botnets, used, and coordinated to conduct attacks such as DDoS, PDoS, spamming, of great scope. For this reason, in this area, resources that allow cyber-security researchers to obtain information on new attacks or even just trends play a fundamental role in the fight against cyber-threats. In this sense, resources such as **honeypots**, which can be used to gather information on new types of attacks on these technologies, play a fundamental role. A honeypot is a software module or piece of hardware used to attract malicious traffic for research purposes or as a decoy to detect an attack attempt. Generally these are isolated nodes of the network, not connected to any valuable asset. However, for these systems to be attractive enough they need to be defined as close as possible to the actual devices. The characterization of these

available for each device. The possibility of having a dataset of possible interactions available allows the application of machine-learning algorithms that would allow a smart honeypot to determine which sequences of messages obtain the best result in terms of interaction time.

1.3 Goals

The main objective of the thesis is to build a framework that can be used to collect information relating to the network traffic generated by consumer IoT devices, used to obtain information related to the interactions supported by these devices. Although the work conducted during the research period appears to be preliminary for the implementation of a larger project, related to the field of cyber-security rather than to the data science field, however, during the analyzes conducted in the test environment, several interesting questions found an answer. The development and implementation of the proposed research environment made it possible to identify not only the network traffic generated by some common home IoT devices, but thanks to the data collected it was possible to determine additional information such as: how much of these messages were protected by encryption, where the traffic collected is sent and other statistics regarding the mode of operation of the different devices.

1.4 Organization of the thesis

This thesis work is organized as follows

- Chapter 2 - **State of art**: An overview on the existing tools and approach used for the characterization of the communication in the IoT field.
- Chapter 3 - **Architecture**: Design choices, description of the architecture of the test-bed used to collect data.
- Chapter 4 - **Results**: Overview and analysis of the results collected by the test-bed.

-
- Chapter 5 - **Conclusions**: Summary of the results obtained, possible improvements and future work.

Chapter 2

State of the art

2.1 Overview

In the following chapter the concept of the Internet Of Things will be introduced, then some of the application protocols most used by the various devices currently on the market will be analyzed. The growing diffusion of IoT devices has highlighted the need for security and privacy, in particular in the following chapter some studies carried out on the subject by the research community are reported. Finally, honeypot systems are illustrated and in particular some solutions developed specifically for the IoT.

2.2 Literature

2.2.1 Internet Of Things

The Internet Of Things (IoT) [2] is an evolving technology that refers to a network of interconnected objects ("things"). Since it is such a varied, ever-changing topic that encompasses various heterogeneous technologies and communications systems, it is actually impossible to formulate a precise definition, the road-map of this technology changes every year. A "thing" is a physical device, capable of sensing data from the environment, and it is also able to communicate with other device or with humans, it can be a smartphone, a sensor, a "smart speaker", RFID device, NFC tag, etc. The

IoT is a paradigm, today used in the most disparate application contexts that is fastly changing the shape of internet itself. The advent of the IoT marked the beginning of a new era in the digital world, introducing an unparalleled blend of the physical and digital world [3], proving to be one of the main drivers of modernization in the telecommunications sector. The extent of innovation is such that it changes the shape of the Internet itself. [4]

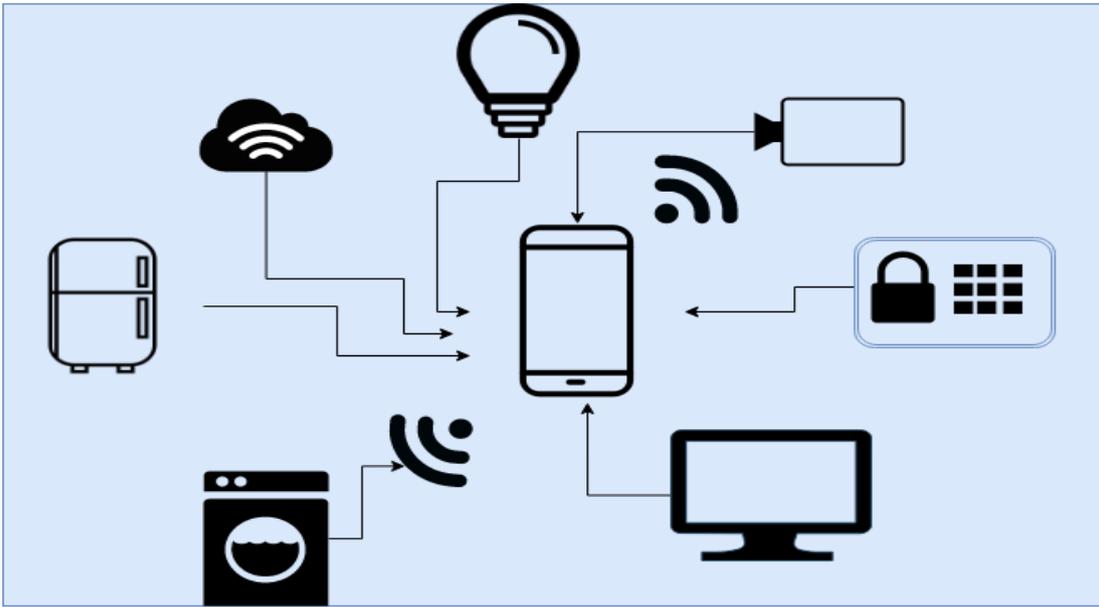


Figure 2.1: Internet Of Things: devices representation.

As analyzed by Perera et al.[5], the advent of the IoT has led to the addition of everyday devices to the internet, a trend that is becoming stronger over the years. If a few years ago most of the data generated on the internet were part of human-generated communications, today the situation is changing. The ubiquity of these devices results in the generation of a large amount of data, at high speed, with high variety. This kind of communication can occur between human and "thing" or between machines (Machine-to-machine or M2M communication).For this reason, the IoT is becoming a topic of interest in big data[6][7], since the characteristics of the data that are exchanged in such a pervasive, ubiquitous and distributed environment are shown to be interesting from several points of view. Although there are many efforts in this direction, nowadays there is no standard for the IoT, the tendency is for each construction

company to define and build their own, proprietary architecture. This lack of uniformity to a standard increases production costs, funds that could be used for more in-depth research on the cyber risks to which these devices are exposed. Moreover, the result is often a complex architecture, which is difficult to secure. Manufacturers develop devices that use different solutions without following a precise standard. [8]

IoT Applications

As already said this paradigm can be applied to several context using different enabling technologies. In general it is possible to categorize the various applications in the following areas.

- Home Automation: Utilities monitoring, remote control over actuators, actuators automation.
- Environment: Environmental data monitoring, natural disaster forecasting.
- Medical and healthcare: Continuous monitoring of patient health condition, remote diagnostics.
- Automotive: Energy efficient and intelligent vehicle.
- Industry: One of the largest sectors. Also known as IIoT, it includes many different solutions with the common purpose of gaining information and reducing development time.
- Smart City: Pollution monitoring, buildings corruption state monitoring, smart traffic management.
- Transportation: Smart fleet management, real time tracking, remote diagnostics.

2.2.2 IoT - Application protocol

Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is a generic, stateless, textual application protocol defined in RFC-1945[9] (HTTP/1.0), RFC-2616[10](HTTP/1.1) and then improved with HTTP/2 (RFC-7540[11], the syntax remains unchanged.). It is one of the

most widespread application protocols and on which the World-Wide Web is based. It is a client-server protocol based therefore on requests and responses. The client who intends to obtain information, formats a request, characterized by an URI[12] (Uniform Resource Identifier) which allows determining univocally a resource, as well as possibly the message body. The server will respond with a status code, which indicates the outcome of the request (eg request correctly formatted, resource not accessible, error, etc.) and possibly the body of the message. The request verb can be one of the following: GET, POST, HEAD, PUT, DELETE, OPTIONS, TRACE, PATCH, CONNECT. The Representation State Transfer (REST) is based on this protocol. An architecture where the exchange of messages takes place using the HTTP methods. The REST architecture remains today one of the most widely used solutions in the IoT field although it is a solution based on a generic protocol and which can therefore determine the exchange of additional overhead information.

Constraint Application Protocol

The Constraint Application Protocol (CoAP, RFC-7272[13]) was created to overcome some limitations related to the REST architecture and to streamline the amount of information that is entered into the network. Again this is a request and response based protocol that uses HTTP. However, unlike the REST architecture, at the transport layer it uses UDP. Requests are formulated using HTTP verbs and are defined using a URI. The types of messages used in synchronous/asynchronous requests are the following:

- Confirmable: requires an ACK
- Non-Confirmable: does not require an ACK
- Acknowledgment (ACK)
- Reset (RST)

Message Queue and Presence Protocol

The Message Queue and Presence Protocol (MQTT, Standard ISO/IEC 20922:2016[14]) is a reliable, TCP/IP-based message transport protocol that uses the client/server paradigm. Born with the aim of satisfying the needs related to the field of Machine-to-machine (M2M) and IoT communication, the protocol has the following characteristics:

-
- It is based on the publish/subscribe model, according to which a node in the network that communicates a data (publisher) forwards it to a node responsible for distributing the messages (broker) to the interested nodes (subscriber).
 - The message protocol doesn't need to be aware of the payload of the message.
 - Support different quality of service (QoS) policies.

As highlighted by Karagiannis et al. [15] this approach is more suitable for the Internet Of Things because in this case, not all messages require a response, reducing the amount of data exchanged in the network improving the power-management performances. Compared to a UDP-based protocol, like CoAP, is affected by the exchange of overhead data related to the TCP-based protocols.

Extensible Messaging and Presence Protocol

Extensible Messaging and Presence Protocol (XMPP, RFC-6120 [16]) is an open and extensible TCP-based standard for messaging based on the publish/subscribe and request/response paradigm. Firstly developed by J. Miller in 1998 with a focus on end-to-end security and later standardized by the Internet Engineering Task Force (IETF). The architecture of the XMPP networks is decentralized, and the messages exchanged are in XML format. It support channel encryption through TLS and it is widely used on the internet and for this reason, it is chosen for some IoT solutions.

Data Distribution Service

The Data Distribution Service[17] is a data-centric API standard and middleware protocol developed by the Object Management Group for Industrial IoT (IIoT). It is a protocol based on the publish/subscribe paradigm with quality of service properties that can be configured for a single entity. The characteristics of this protocol allow hiding the complexity of data sharing. It is generally used in contexts with real-time requirements. [18]

Advanced Message Queuing Protocol

The Advanced Message Queuing Protocol (AMQP)[19] is a reliable, extensible, open messaging protocol designed by a consortium of middleware vendors. It supports the

publisher/subscriber paradigm, through communication with a broker and a request/response mode of interaction with the messaging infrastructure.

2.2.3 Security

Since this type of technology is so pervasive and often used in critical contexts involving the recording, processing and communication of sensitive data, the security of these devices is a major challenge. The violation of the security controls of these devices also makes it possible in several cases to tamper with or misuse physical devices. On October 12, 2016 a distributed denial of service (DDoS) has deprived much of the east coast of the United States of internet connection. The attack, of nation-state scope was conducted by the Mirai botnet. It simply scanned big chunks of internet searching for IoT device, then it performed a TCP SYN scan on telnet ports (23, 2323), if open it attempted to log in with default credentials by performing a *brute-force* attack. The peak was of 600k infection with a volume of 600Gbps. The targets of this massive attack were Krebs on Security, OVH, and Dyn. [20] Analyzing in detail the functioning and the mechanism used by this botnet is not the purpose of this thesis, but it is certainly necessary to highlight some characteristics. The dictionary used to perform the brute force attack on the login phase consisted of only 62 pairs of username and password (fig. 2.2), although it presented some innovative features [20], especially in the reconnaissance phase, it still belonged to the *BASHLITE* [21] family of Botnets.

```
123 // Set up passwords
124 add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10); // root xc3511
125 add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9); // root vizxv
126 add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8); // root admin
127 add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7); // admin admin
128 add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6); // root 888888
129 add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5); // root xmhdipc
130 add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5); // root default
131 add_auth_entry("\x50\x4D\x4D\x56", "\x48\x57\x43\x4C\x56\x47\x41\x4A", 5); // root juantech
132 add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5); // root 123456
133 add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5); // root 54321
134 add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\x50\x56", 5); // support support
135 add_auth_entry("\x50\x4D\x4D\x56", "", 4); // root (none)
136 add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x51\x55\x4D\x50\x46", 4); // admin password
137 add_auth_entry("\x50\x4D\x4D\x56", "\x50\x4D\x4D\x56", 4); // root root
```

Figure 2.2: Portion of the code related to the setup of the credentials used in the brute-force attack. [1]

Another point worth dwelling on is the following: analyzing the timeline of secu-

rity events related to this malware, it is evident to underline that there was no effective countermeasure against this attack, although a reboot was enough to clean the devices. (No corruption of operating system services). This is the perfect example of lack of by-design security, use of default passwords and in some cases the inability to update vulnerable devices remotely. The source code of Mirai can be found on several public repositories, and at the time of writing there are devices vulnerable to this type of attack.

Mirai is not an isolated case. In October 2016, for the first time, a worm similar to Mirai was identified, which was then named Hajime. The main differences compared to Mirai are the use of a distributed communication paradigm. Once a new node is infected, Hajime is able to establish a connection with the attacking node, requesting malicious code specific for the infected hosting platform.

- ARMv7
- ARMv5
- MIPS, LE
- x86-64

Once infected it uses a P2P connection to retrieve additional malicious code. This kind of botnet was discovered through the use of an honeypot by the *SRG* researchers. [22] Also in this case it is important to underline the evolution of the botnet in the short time that has passed since Mirai's discovery, and the dedication of the attackers to optimize the functioning of the software, which as highlighted by Sam Edward et al. was written by hand. [22]

All these attacks were carried out exploiting "simple" IoT devices, built to perform a specific task. But this cannot be translated into a smaller attack surface. Of course countermeasures like blocking useless ports can be part of a good security control, but what makes the attack surface so extended is the complexity of the environment in which such devices operate. Another botnet, "BrickerBot", discovered in April 2017 after infecting an insecure IoT device, conducts a destructive attack against the host device (Permanent Denial Of Service, PDoS) by wiping the memory and making the node unable to connect. [23] Attacks on such devices are made more targeted through

the use of social engineering. To take an example, in 2017 a scammer broke into the security systems of a baby monitor to stage a fake kidnapping.[24]

The lack of security in the IoT is a fact, the assets at risk are different, sensitive data and privacy are some of them, it is important to highlight how possible vulnerabilities can have a high impact on the physical world. This is due to lack of experience from manufacturers, lack of security-by-design, out-of-date security measures, extended attack surface.

Malware is developed in an open source environment, evolution of these types of malicious software is not only possible, but probable.

Countermeasures

To protect the CIA triad an efficient security control have to be established with the aim of avoid, counteract and minimize the the security risks of physical properties, data, computer systems and other assets. Different cyber-security measures act at different stages of an event.

- Before the event: Preventive checks - E.g. access control, firewall and/or proxy configuration, etc.
- During the event: Identify and characterize an ongoing event - Usually performed by an NIDS (Network Intrusion Detection System)
- After the event: Corrective actions to limit the damage. E.g. patches, updates, NIDS's signatures update.

Cyber-attack detection

One of the major challenges in the field of IT security is the cyber-attack detection. Security is not something that can be achieved, rather something that must be continuously researched and improved. A system certainly cannot be declared as safe and remain so over time. The most clear example is carried out by the DoS attacks. This kind of attack is one of the oldest attack discovered in the literature of the IT security, yet today digital giants are still affected by this type of attacks.

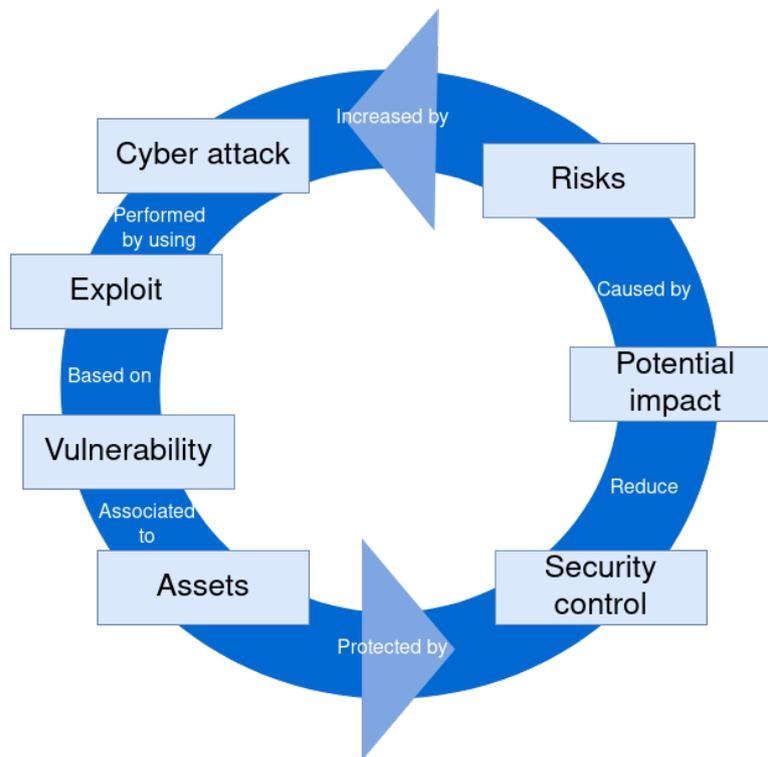


Figure 2.3: Diagram showing the dependency between elements in the IT security.

The challenge is more complex if the vulnerability on which an exploit is built is not known to the security community. This kind of attacks are also known as *zero-day attack*. Preventing zero-day attack is a real struggle for any security team. These attacks hard to predict and the relative exploit do not trigger the NIDS, bypassing it. If a rules for a specif attack it's not defined in the NIDS the attack is considered a legitimate request. In order to gain information on this kind of vulnerabilities resource like the honeypots are widely used. (The source code of Hajime, cited in 2.2.3, was discovered by using an honeypot).

2.2.4 Honeypot

A honeypot is a security resource used to collect information about cyber attacks. It is a computer system built and configured with the sole purpose of being attacked or compromised, eliminating the possibility of damaged valuable assets. A honeypot does

not host any production server and does not interact with any other host in the network unless it is solicited, so it does not receive legitimate traffic, all requests directed to it are suspicious by design, for this reason the number of false positive/negative is generally low. [25]

As highlighted by a. Mairh and al. [26] using honeypots as a resource in the reasearch of zero-day attacks has the following advantages:

- Honeypots targeted only by *blackhats*, normal users do not interact with this machines.
- Honeypots are built with to create a log file containing the interaction with other hosts, there is no need for implement a binary classifier used to differentiate legitimate and malicious traffic.
- The number of false positive/negative is low.
- Honeypots are presented as a real machine without the risks of being compromised and affect other machines on the network.

Honeypots can be classified as: low interaction honeypots or medium/high interaction honeypots. This classification is related to the availability of the system. Highly interacting honeypots are able to collect a greater amount of more valuable data, however they are often difficult to configure. Furthermore, if the system is fully accessible, it may be used as a vector for other illegal activities, physical devices and other assets may be damaged. A compromise is often necessary. This resource is also used to attract *blackhats* to an apparently more vulnerable machine with the aim of identifying and blocking possible attacks against more valuable assets. From this point of view, it is clear how building a honeypot that is able to attract attacks directed towards specific types of devices can be useful to ensure a greater level of security. To do this, it is necessary to build and model a system that operates as such devices without, however, the possibility that physical devices are damaged or that other nodes of the network compromised.

Low-interaction Honeypot

A low interaction honeypot provides low operating system availability, and for this reason, it is the most robust solution to compromise. They are relatively simple to

configure, once the personality is defined, in terms of operating system and active services they require no further maintenance. However, more experienced *blackhats* are able to fingerprint these machines, ceasing the interaction at the reconnaissance stage. For this reason usually this kind of honeypot are used to attract attackers that will be added to a blacklist.

Medium-interaction Honeypot

This type of honeypot is characterized by a higher level of interaction with the attacker. Like low-interaction honeypots, the attacker cannot interact with a full operating system, however in this case some services are simulated in a more sophisticated way. The responses generated by this type of machine are no longer generic, but more realistic. This requires more work during the development phase, however, as far as the deployment and maintenance phase is concerned, the performance is similar to low interaction honeypots. The same can be said regarding the level of risk to which these types of solutions are exposed. However, providing less generic answers makes this approach more profitable in terms of the interaction time with *blackhats*, one of the fundamental metrics in evaluating the overall performance of a honeypot.

High-interaction Honeypot

High-interaction honeypot act as a real production system. These honeypots gather the greatest amount of valuable information, allowing a deep interaction with attackers. This type of approach can be achieved by making a physical device available on the network, or by virtualization so that multiple honeypots can be virtualized on the same machine. As more operating system features are made available without restriction these types of machines can be subject to more complex attacks, which increase the level of risk these systems are exposed to. For this reason and since more complex infection mechanisms such as service corruption can be implemented, it is necessary to reinstall the device periodically and/or after each interaction. This determines a strong increase in costs during the development, deployment, and maintenance phases.

As pointed out by Nawrocki et al.[?], this taxonomy in practice is very difficult to apply, as in general, it is difficult to classify a honeypot. In fact, software solutions that simply listen to specific ports are classified as low-interaction honeypots while

	ROI	Development cost	Deployment Cost	Maintenance cost
Low-interaction	+	+	+	+
Medium-interaction	++	++	+	+
High-interaction	+++	+++	+++	+++

Table 2.1: Return on Investment, and costs related to different types of honeypots.

solutions that more or less faithfully emulate a specific service are classified as medium/high interaction honeypots.

2.3 IoT-oriented honeypots

2.3.1 Honeyd

Honeyd [27] is one of the most used open source solution for Honeypot. Honeyd provides a daemon that can be used to virtualize any topology, configuring the available TCP/UDP services for each host, emulating specific operating systems. When a host interacts with Honeyd, the software craft a response enriched with some details related to the personality of the machine. It is not a solution specifically developed for the IoT environment, but lends itself to extensions. This software can be used to configure a low-interaction honeypot, but it can be extended with several plugins, others approaches will use this solution as a base for further additions. Each packet it's processed by a central node that checks the correctness and then dispatches it to the specific handler. Honeyd supports GRE tunneling that can be used to create more scalable distributed setups.

HoneyIo4

A. Guerra Manzanares [28] propose a platform composed of a Linux based virtual machine OS. In particular, the author's goal is to build a software module that allows a virtualized machine to present itself as one of four different IoT devices for a TCP/IP fingerprinting analysis. Software such as Nmap support a scan which by sending a sequence of TCP, UDP and ICMP packets is able, based on the answers provided, to

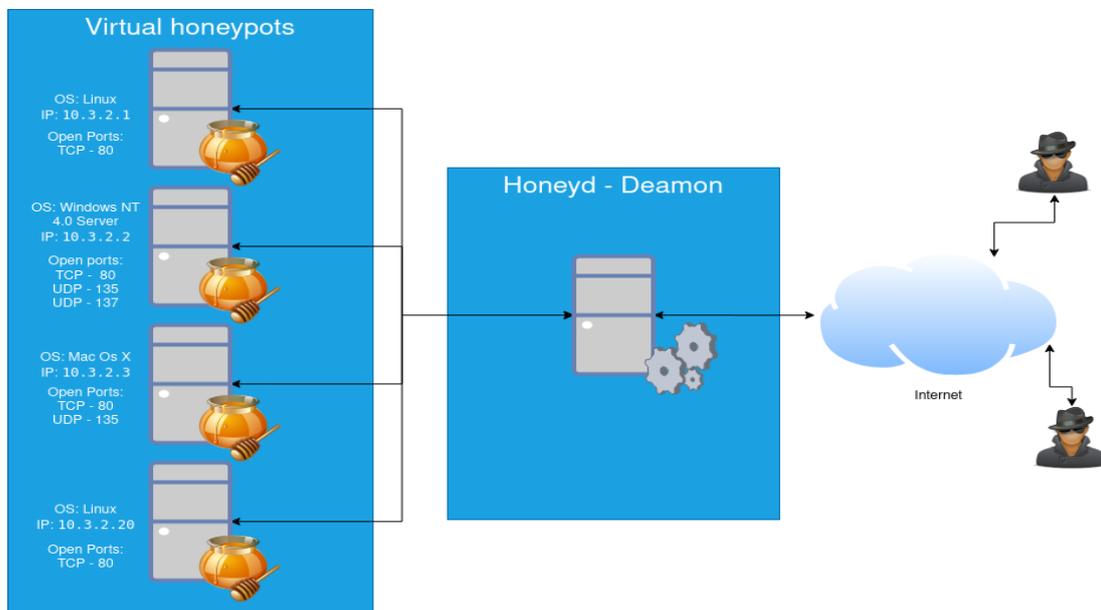


Figure 2.4: *Honeyd* sample configuration.

determine the target operating system with a percentage of confidence. In particular, the proposed approach is carried out in the following phases:

- A TCP / IP fingerprinting of a real device is performed, with the aim of capturing and analyzing the responses provided by the target.
- Analyze the target's responses to probes sent by the network scanner
- Define, in this case using Scapy, a Python library, the code necessary for the virtualized machine to respond exactly as the target in the first step.

IoT POT

The solution proposed by Yin Minn Pa Pa et al. [29] is a honeypot developed for the purpose of investigating Telnet-based attacks against various IoT devices. The implementation of IoT POT consists of several modules as shown in figure 2.5.

- A frontend Responder, whose purpose is to manage the interaction following different device profiles, which define: the responses sent by the honeypot when

the connection is opened, the authentication mode, and the responses sent to the various commands.

- IoTBOX is a Linux-based environment used to provide a specific response to a CPU architecture to the frontend responder, who will then forward it to the client.
- The Profiler component updates device profiles based on interactions managed by the responder.
- The Downloader component is used to inspect any injected binary malware.

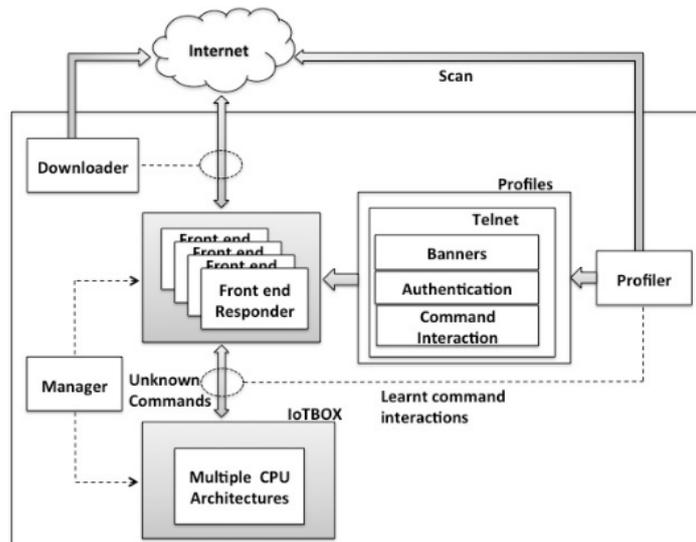


Figure 2.5: IoT POT implementation design.

IoT CandyJar

T. Luo et al.[30] propose an innovative approach by defining an Intelligent-interaction Honeypot. In this case, the proposed framework acts at first as a low interaction honeypot, which therefore provides generic answers to the client's requests. However, the requests received are then used to investigate the responses provided by real IoT devices, in order to update the model and insert a more realistic response in the IoT database, improving overall performance.

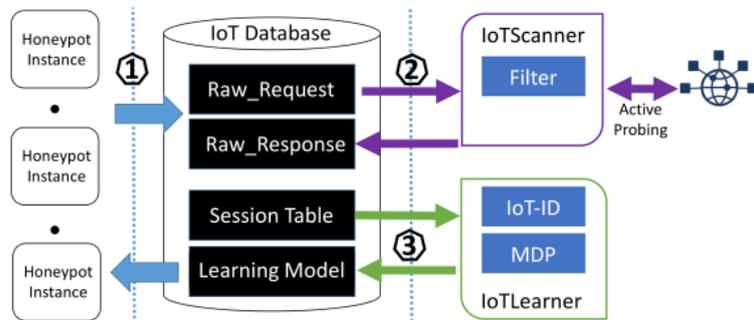


Figure 2.6: IoTCandyJar architecture.

The proposed approach is very efficient in identifying the messages used before an attack, during the reconnaissance phase, but clearly cannot forward messages containing exploits to real IoT devices.

Chapter 3

Architecture and Deployment

3.1 System Architecture

The following sections will analyze the architecture of the testbed system. In the first phase also due to the restrictions related to the Covid-19 epidemic, the system was tested in an environment consisting of a few devices. Later the same scheme was used in the IoT laboratory of the SmartData research group of the Politecnico di Torino. The latest solution allowed us to verify and test different devices, able to interact with each other, creating a more complex system. In particular, it was noted that some devices communicated directly, within the local network, others, instead, always relied on a cloud. Moreover, also considering the variety of devices present in the test environment, different types of interactions, specific for each product, were tested. (E.g. smart cam, voice assistant, smart plugs, etc.). The study related to the capture of the messages exchanged by the devices on the network can be addressed in different ways. The parameters used as metrics for choosing an implementation solution over another were the following:

- Coverage: in terms of messages captured by the test environment with respect to the totality of interactions that take place.
- Scalability: the system have to be designed to be able to operate even after the addition of new devices, different in terms of application protocols used and/or quantity and form of information flow generated.

-
- Intrusiveness: The system must operate transparently, minimizing the interference with the normal work cycle of the devices within the network.
 - Performance: the analysis features must be efficient.

3.1.1 First setup

The system architecture of the testbed will be analyzed in detail below. In a first phase, the test was conducted on a small number of devices.

- SONOFF basic R2
- SONOFF 4CH PRO R2
- Shelly-1

For these types of devices they were chosen as proof of concept for the testbed since the documentation provided by the manufacturers was complete and exhaustive regarding the interactions supported, this was fundamental in the validation phase of the results. Furthermore, all the devices under examination used the HTTP/HTTPS application protocol, which made it easier to inspect the captured data. The devices fall within the scope of consumer IoT devices, they are used for home automation. These devices are equipped with an actuator that can control a relay. In particular, the devices in question use RESTful web services to provide the proposed functionalities. Furthermore, both devices had similar operating modes that will be presented later. These devices, using a very limited number of commands (ON, OFF, status), have allowed an exhaustive study of the messages exchanged. The analysis conducted allowed the crafting of messages recognized as valid by the devices, thus verifying the correctness of the approach. The use of tools such as Postman [31] was useful for testing and verifying all message fields inspected and captured by the system.

SONOFF basic R2 & 4CH PRO R2

These devices can operate in two different modes:

- *EWeLink* mode: In this case the device is connected to the cloud and is controlled by an App. The device will exchange information within the LAN if the



Figure 3.1: SONOFF basic R2, SONOFF 4CH PRO R2, Shelly-1.

smartphone is connected to the same network, otherwise these will be conveyed by the cloud.

- *DIY mode*: In this case the device publishes its own capability services in the network, which can be reached through the HTTP-based RESTful API.

Shelly 1

This type of device has a function in fact similar to SONOFF products.

- *Access Point (AP) mode*: Used to perform the first configuration. The device advertises an HTTP service on port 80 using the mDNS protocol. All requests, properly formatted, return a JSON-encoded payload. If enabled, all resources will require HTTP authentication.
- *Client Mode (STA mode)*: The device is connected to the cloud, controlled by application, similar to the “eWeLink” mode of the previous devices.

Environment setup

This section describes the steps taken to define a first test environment. In this first phase, the architecture is shown in figure 3.2. A Raspberry Pi 3 Model B is used as an access point on which an instance of MITM Proxy runs[32].

MITM Proxy[33] is an open-source proxy that can be used to intercept HTTP/1, HTTP/2, WebSockets, or other protocols protected by SSL/TLS by additional plug-ins. It can operate as a transparent proxy, thus without altering the traffic that passes through it, and also supports a mode that allows you to make changes on the messages that are intercepted. The MITM proxy is used to perform a man-in-the-middle attack.

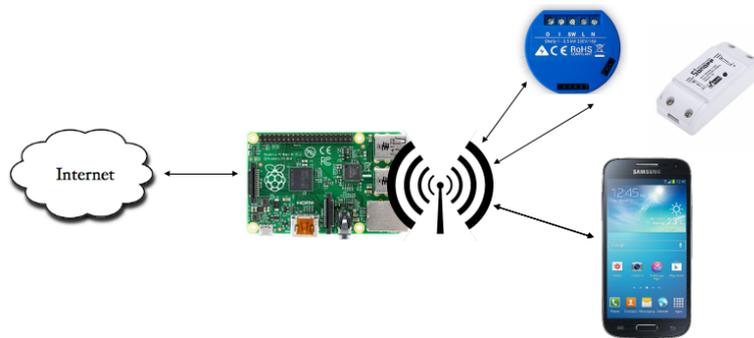


Figure 3.2: Environment setup.

An attack of this type aims to place the attacker in the middle of a communicative association previously established by two network nodes, with the goal of intercepting, possibly altering, and retransmitting the intercepted traffic. With MITM proxy all traffic can be viewed in real-time via a web interface.

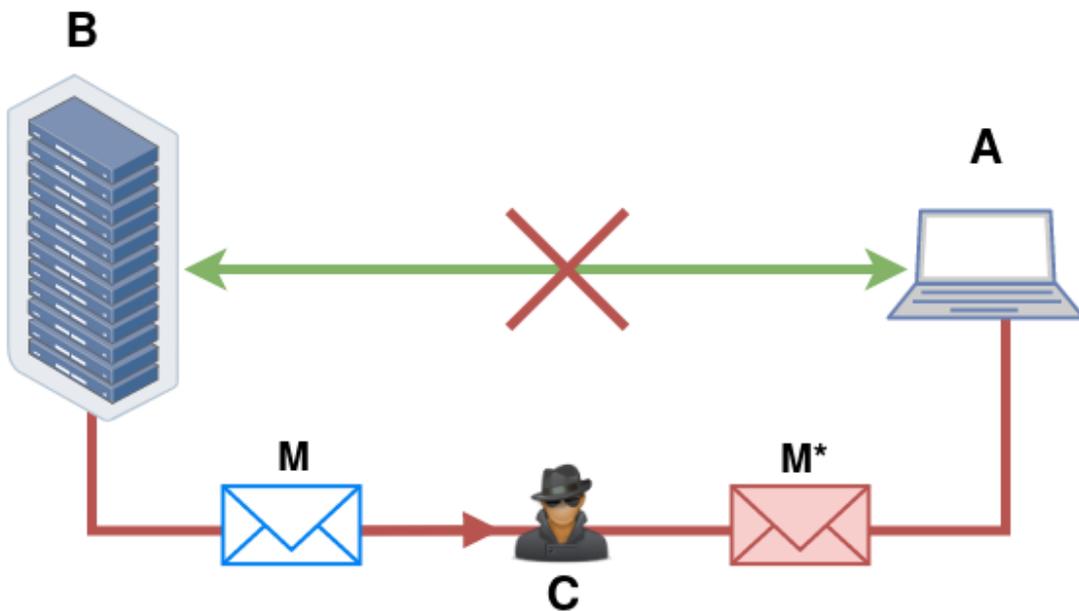


Figure 3.3: Man-in-the-middle attack performed by C against A and B. The message M is captured by C and possibly altered as M*.

The Raspberry must therefore be configured to host a WiFi hotspot, this is achieved

by using the hostapd service. To ensure internet access to the devices in the test environment, a DHCP server must be configured, for this purpose, the dhcpd and isc-dhcp-server service were used. The first step necessary to configure the DHCP client is the assignment of a static IP address for the chosen network interface.

```
1 interface wlan1
2 static ip_address = 192.168.42.1 / 24
3 nohook wpa_supplicant
```

Listing 3.1: DHCPd configuration file.

As shown in Listing 3.1, in this case, the wlan1 interface was assigned the address 192.168.42.1. Furthermore, the above configuration file prevents the WPA supplicant service from interfering with DHCP. Another software module is responsible for managing the addresses that will be assigned to the devices connected to the access point: isc-dhcp-server. Specifically, this is configured as shown in listing 3.2.

```
1 # If this DHCP server is the official DHCP server for the local
2 # network, the authoritative directive should be uncommented.
3 authoritative;
4 subnet 192.168.42.0 netmask 255.255.255.0 {
5 range 192.168.42.10 192.168.42.250;
6 option broadcast-address 192.168.42.255;
7 option routers 192.168.42.1;
8 option domain-name "local";
9 option domain-name-servers 8.8.8.8, 8.8.4.4;
10 }
```

Listing 3.2: ISC-DHCP-Server configuration file.

In this specific case, the range of IP addresses that will be used for the assignment belongs to the range 192.168.42.10 - 192.168.42.250. The router address is the static one configured for wlan1. At this point, the ISC-DHCPv4 server is configured.

At this point, the hostapd service has been configured as follows there are different parameters that can be varied among the possible options. The most relevant for this case study are the following (listing 3.3).

```
1 interface = wlan1
2 ssid = test-env0
3 macaddr_acl = 0
4 auth_algs = 1
```

```
5 wpa = 2
6 wpa_passphrase = testenv0pwd2020
7 wpa_key_mgmt = WPA-PSK
8 wpa_pairwise = TKIP
9 rsn_pairwise = CCMP
```

Listing 3.3: HostAPD configuration parameters.

- **interface = wlan1**: AP will be available through the wlan1 network interface.
- **ssid = test-env0**: The SSID of the WLAN.
- **macaddr_acl = 0**: Use blacklisting for Station MAC address-based authentication.
- **auth_algs = 1**: IEEE 802.11 specifies two authentication algorithms, Open System AuthN (to be used with IEEE 802.11X) or Shared Key Authentication, in this case, the latter is used.
- **wpa = 2**: Use WPA2 - Wi-Fi Protected Access 2 (IEEE 802.11i).
- **wpa_passphrase = testenv0pwd2020**: The passphrase used by WPA.
- **wpa_pairwise = TKIP**: Pairwise cipher for WPA.
- **rsn_pairwise = CCMP**: Pairwise cipher for RSN/WPA2.

Once the access point is configured it is necessary to set up the MITM proxy. An example of use is:

```
1 mitmweb --mode transparent --web-port 9090 --web-host 0.0.0.0
```

Listing 3.4: Example of use of MITM proxy.

In this case, the version with a web interface available on port 9090 is used, the proxy operates in transparent mode and can be reached starting from the LAN IP address of the device. To use the setup of the MITM proxy in transparent mode, you need to add new firewall rules, as shown below.

```
1 iptables -A FORWARD -i eth0 -o wlan1 -m state --state RELATED,  
   ESTABLISHED -j ACCEPT  
2 iptables -A FORWARD -i wlan1 -o eth0 -j ACCEPT  
3 iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE  
4 iptables -t nat -A PREROUTING -i wlan1 -p tcp -m tcp --dport 80 -j  
   REDIRECT --to-ports 8080  
5 iptables -t nat -A PREROUTING -i wlan1 -p tcp -m tcp --dport 443 -j  
6 REDIRECT --to-ports 8080
```

Listing 3.5: Firewall rules added for transparent proxy configuration.

Specifically, the set of rules shown in listing 3.5 allows you to redirect the packages that are received from the wireless interface to the wired interface. In this specific case, only the messages directed to port 80 and 443 are intercepted by the proxy. Finally, packet forwarding for IPv4 is required.

3.2 Passive analysis

This section analyzes the passive analysis techniques used to capture the traffic of devices on the network. These types of analysis do not require the direct interaction with the device. Instead, these techniques are based on access to the communication channel with the aim of obtaining information about the data exchanged by the devices that have established a communication association[34].

3.2.1 Traffic monitoring

Wireshark

Wireshark is the most common network protocol analyzer. It can be used to perform inspection of network protocols with different purposes, debugging or network protocols reverse engineering. It supports the inspection of a different set of protocols in live capture or offline. Live data can be retrieved from Ethernet, IEEE 802.11, Bluetooth, PPP, etc. It was used during the first phase of the testbed development to collect the messages used by different IoT devices to support different tasks. [35] The requests gathered by this tool were analyzed following a conceptual model as proposed by Forshaw [36].

- **Content layer:** Related to the conceptual meaning of such request. E.g. turn a smart relay to the ON state.
- **Encoding layer:** Related to the way used to perform such request. E.g. an HTTP POST method, as shown in listing 3.7
- **Transport layer:** Related to how the information is sent on the network. E.g. by a TCP/IP connection, as shown in listing 3.6

```

1 0000  00 00 24 00 2f 40 00 a0 20 08 00 00 00 00 00
2 0010  4e 07 b1 09 00 00 00 00 10 0b 85 09 a0 00 de 00
3 0020  00 00 de 00 88 41 df 00 20 b0 01 da 15 f1 04 4f
4 0030  4c 6a f7 78 dc 4f 22 93 b9 37 60 83 00 00 ef 49
5 0040  00 20 00 00 00 00 20 08 bf 8e a0 ba 17 c4 a2 2e
6 0050  15 c0 cd c4 d4 6e 91 aa e7 bf 0d b1 65 82 8b 7c
7 0060  dd 35 03 c5 28 9b f4 eb 69 b3 4c 56 91 51 31 69
8 ...

```

Listing 3.6: Transport layer representation of an HTTP POST request sniffed by Wireshark.

```

1 POST /zeroconf/switch HTTP/1.1
2 cache-control: no-store
3 accept: application/json
4 Content-Type: application/json; charset=UTF-8
5 Content-Length: 185
6 Host: 192.168.1.50:8081
7 Connection: Keep-Alive
8 Accept-Encoding: gzip
9 User-Agent: okhttp/3.12.1
10
11 {"sequence":"1601224819801","deviceid":"10005f3607","selfApikey
    ":".....","iv":"MTEwMTExMTA2OTUxMDY1Mw==","encrypt":
    true,"data":"c3aa3xzgkg9dwWIXkfk30w=="}

```

Listing 3.7: Encoding layer representation of an HTTP POST request sniffed by Wireshark.

This tool allowed us to verify the possibility of intercepting all the traffic in the network in question, to verify the loss of packets or the incorrect interpretation of the same without having to develop specific software in case some critical issues were detected. The captures made using Wireshark met the required requirements, but the

request to automate the capture, analysis, and creation of an archive led to the choice to use libraries used by the same tool in ad hoc developed scripts. Furthermore, the choice of using scripts based on libpcap libraries[37] allows you to customize and extend the information, which will then be collected and stored, based on the experimental needs. It was also decided to separate the message capture part from the message analysis and storage part. Thanks to this choice, the system can also be used with communication systems (E.g. bluetooth, NFC, ZigBee, etc.) other than the one used in this project. The re-usability of the code was in fact one of the requirements set at the time of design.

Traffic capture via monitor mode interface.

The approaches tested for passive traffic analysis also include traffic capture via the wireless network interface in monitor mode. Unlike the capture with the network interface in promiscuous mode in this case it is not necessary that the network card is connected to the same LAN to which the target devices belong. This approach therefore does not interact at all with the operation of the devices and does not in any way alter the flow of packets that are exchanged within the LAN or sent/received over the Internet. In particular, each device compliant with the 802.11a/b/g standard can operate in different modes:

- **Managed mode** (Client mode): In this case, the device behaves like a client and is, therefore, able to communicate with other nodes connected to the same master.
- **Master mode**: The network cards configured in this way operate as Access Point, thus providing connectivity to client devices.
- **Monitor mode**: This mode is only supported by some specific network cards, and is mainly used for diagnostics. The network interfaces configured in monitor mode are not able to transmit data but only to receive it.
- **Ad-hoc**: This mode is used for creating ad-hoc networks. In this case, the topology is that of a graph.

To configure a network interface in monitor mode, the Airmon-ng[38] script and an external WiFi adapter equipped with a chipset supporting monitor mode were used.

73	106.762907	HuaweiTe_6a:f7:78	Broadcast	ARP	42	Who has 192.168.1.254? Tell 192.168.1.245
74	106.907649	2001:b07:5d2e:6b9e:...	2a00:1450:4002:804:...	TLSv1.2	888	Application Data, Application Data
75	106.271821	2001:b07:5d2e:6b9e:...	2a00:1450:4002:804:...	TCP	94	[TCP Retransmission] 33380 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
76	106.272147	2001:b07:5d2e:6b9e:...	2a00:1450:4002:804:...	TCP	94	[TCP Retransmission] 33382 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
77	106.396384	2001:b07:5d2e:6b9e:...	2a00:1450:4002:804:...	TCP	94	[TCP Retransmission] 30852 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
78	106.539189	HuaweiTe_6a:f7:78	Broadcast	ARP	42	Who has 192.168.1.254? Tell 192.168.1.245
79	106.539246	HuaweiTe_6a:f7:78	Broadcast	ARP	42	Who has 192.168.1.254? Tell 192.168.1.245
80	106.539661	2001:b07:5d2e:6b9e:...	2a00:1450:4002:804:...	TCP	94	[TCP Retransmission] 33430 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
81	106.541192	2001:b07:5d2e:6b9e:...	2a00:1450:4002:805:...	TCP	94	[TCP Retransmission] 51936 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
82	106.541435	2001:b07:5d2e:6b9e:...	2a00:1450:4002:803:...	TCP	94	[TCP Retransmission] 38484 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
83	106.542017	2001:b07:5d2e:6b9e:...	2a00:1450:4002:803:...	TCP	94	[TCP Retransmission] 38486 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
84	106.542533	2001:b07:5d2e:6b9e:...	2a00:1450:4002:805:...	TCP	94	[TCP Retransmission] 51936 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
85	106.543094	2001:b07:5d2e:6b9e:...	2a00:1450:4002:805:...	TCP	94	[TCP Retransmission] 51936 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
86	106.566564	2001:b07:5d2e:6b9e:...	2607:f800:400a:803:...	TCP	94	47770 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
87	106.566942	2001:b07:5d2e:6b9e:...	2607:f800:400a:803:...	TCP	94	47772 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
88	106.567735	2001:b07:5d2e:6b9e:...	2607:f800:400a:803:...	TCP	94	47774 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
89	106.568564	2001:b07:5d2e:6b9e:...	2607:f800:400a:803:...	TCP	94	47776 - 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SF
90	106.737106	2001:b07:5d2e:6b9e:...	ff02::1:ff00:1	ICMPv6	86	Neighbor Solicitation for 2001:b07:5d2e:6b9e::1 fr

Figure 3.4: Decrypted network traffic, captured in monitor mode.

With this configuration, it is possible to capture all the wireless traffic in the range of the adapter, to do this once again the Wireshark software was used. Once the traffic coming exclusively from the LAN in question has been filtered, Wireshark tools have been used to decrypt the captured traffic, after that the capture file appear as shown in 3.4 In this case, however, the traffic is burdened by a series of messages necessary for the functioning of the IEEE 802.11 protocol but not interesting for the purposes of the project. Furthermore, since the device that performs the capture does not belong to the LAN, a further step is required in the data processing phase. However, this choice could be interesting to extend the analysis to devices that use different protocols.

3.2.2 Automate sniffing with Scapy

Several options were available to automate the traffic scanning process, such as writing a script for bash using the Wireshark command-line interface, tshark, or using libraries developed specifically for this purpose. In this case, a program developed in Python, Scapy, was used. Scapy allows you to intercept, create, and inspect network packets conveyed via TCP, UDP or ICMP. In particular, Scapy acts as a wrapper for the libpcap library. The choice of this wrapper is related to the fact that this library allows not only to capture and inspect network traffic but also allows the modification of received packets and the forging of new ones. Scapy was therefore chosen to leave the possibility of extending the functionality of the testbed. Scapy also allows for a layered analysis of captured packets. The considerable flexibility of the chosen library can allow an in-depth analysis of any attacks on monitored devices, and testing of any exploits.

SSL/TLS encryption

Although the first solution appears to be the most practical and apparently complete, several critical issues emerged during its use. In particular, this solution does not meet the requirements chosen during the design phase of the testbed. The most severe criticality found is related to HTTPS traffic. In this case, the proxy should recognize the secure connection established via TLS/SSL, automatically detecting the ClientHello message. The client then establishes a secure connection with the proxy believing it is communicating with the remote server. MITM proxy in turn establishes a secure connection with the server, receiving a certificate, it also generates a fake certificate that will be sent to the client to perform the authentication. From this moment on, the two TLS connections are active, between client and proxy, and between the proxy and remote server. In this procedure, it is therefore essential that the client accepts the certificate generated by the proxy recognizing it as valid and owned by the remote server. Unfortunately for almost all tested devices, the certificate validation process failed, thus preventing the devices from communicating. A possible solution, suggested by the developers of the tool themselves, could be to install an ad hoc generated certificate as trusted, thus changing the list of trusted certification authorities (Trusted CAs), but in the case of IoT devices, this is impractical. However, even if this procedure were possible this would go against the scalability and intrusiveness requirements defined in the design phase. In fact, this procedure should be practiced for each new version of the device firmware and for each new device added to the network. On the basis of the previous considerations, it was decided not to use the aforementioned proxy but to automate the traffic capture phase and postpone its analysis to a later, offline phase.

3.3 Active analysis

This section shows the techniques and tools used during the active analysis phase. In this case, these techniques require interaction with the target node, with the aim of acquiring information from this. These types of techniques can be carried out manually or can be automated using specific tools, the tools and the approaches used to automate the pipeline are described below. The active analysis allows you to collect additional information and link the results obtained by drawing on any further requests directed to the monitored devices. These scans also allow us to detect any device firmware updates

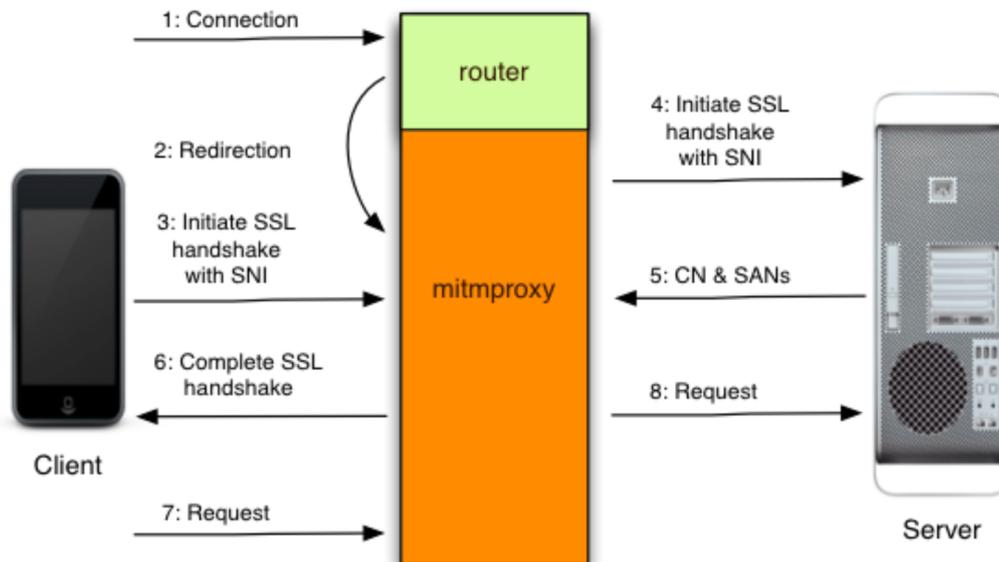


Figure 3.5: MITM proxy - HTTPS interception.

that change the ports used and their protocols. It should also be noted that active analysis is fundamental for the characterization of the personalities of a honeypot. This type of analysis is sufficient to create a low-interaction honeypot.

Port scanning

The analysis of active ports on a host is fundamental in defining the interactions supported by the device and during the host discovery. For this reason, it is one of the scans that are carried out by both pentesters and blackhats during the reconnaissance phase, since the information obtained is the basis for subsequent searches for vulnerabilities. An example is the aforementioned Mirai botnet, which to determine new possible targets conducted a scan on large blocks of IP addresses in order to determine which services were active on that host and if there were any vulnerable ones among them. In the present study, the analysis of the active ports allows to facilitate the subsequent traffic analysis, furthermore this analysis is useful for determining the hosts present on the network and their active ports even if they will not generate traffic during the experiments.

Nmap analysis

Nmap

Nmap is an open-source network mapper. It is used by the network administrator to perform various tasks, such as monitoring hosts and/or services, host discovery, port scanning.[39] Nmap supports the creation of various IP packets which can be used to probe for active services on a specific host or network. Since the purpose of this thesis is preliminary to the creation of a digital twin of an IoT device, the scanning of active services and related open ports is a necessary step.

Nmap supports different types of port scanning techniques for TCP and UDP[40], Below are some types of scans carried out during laboratory tests.

- **TCP SYN Scan:** The device send a SYN packet, as if it is going to open a real connection. It waits for the SYN/ACK (open port) or RST (closed port, it is the default option for Nmap). It is fast, unobtrusive and stealth.
- **TCP connect scan** The device tries to establish a connection with the target machine by using the connect system call. Although it is less efficient, less controllable (it uses high level API) it is more reliable rather than the TCP SYN Scan. The request can be logged. Used also by users without raw packet privileges.
- **UDP scans** UDP scanning is generally more slower and complex than TCP. The devices send a UDP packet to the target machine. For some well known port a protocol-specific payload is sent to increase response rate. Empty otherwise.
- **TCP NULL, FIN, Xmas scan** Useful to differentiate between open and closed ports. These scans exploit a loophole in the TCP RFC (793). Can be used for systems compliant with this RFC

In the case study aspects such as request logging or the furtiveness of scans are not considered fundamental, rather the goal is not to interfere with the correct functioning of the devices in the test environment.

```
1 # Nmap 7.80 scan initiated Sun Sep 27 17:20:35 2020 as: nmap -Pn -sS  
  -oN tcp_syn_scan.txt 192.168.1.126  
2 Nmap scan report for shelly1-12345.lan (192.168.1.126)  
3 Host is up (0.016s latency).
```

```
4 Not shown: 999 closed ports
5 PORT      STATE SERVICE
6 80/tcp    open  http
7 MAC Address: AA:BB:CC:11:22:33 (Espressif)
8 # Nmap done at Sun Sep 27 17:20:40 2020 -- 1 IP address (1 host up)
   scanned in 4.44 seconds
```

Listing 3.8: Example of Nmap scan report.

Automating port scanning

The automatic analysis of hosts on the network and their open ports was done using a nmap wrapper for python. This choice is consistent with the previous design choices, and also in this case allows you to easily extend the functionality of the testbed, introducing further scans and analyzes. In particular, the hosts on the network are identified by means of a pingsweep on the test network (as shown in listing 3.9), then for each online host, a scan of the open TCP and UDP ports is performed. To increase script performance, we have chosen to delegate scanning to a process pool of 1 to 8 processes. This choice was made on an experimental basis, optimizing the time required for scanning and the computational resources available and used. This type of scan must be repeated each time the monitored devices are changed. All the information collected is stored in a no-sql archive, the details of which will be better specified later.

```
1 nm.scan(hosts=subnet, arguments='--max-parallelism 100 -sP -PE -PA
   21,23,80,3389,8081')
2 print('Found %d devices' % len(nm.all_hosts()))
3
4     for host in nm.all_hosts():
5         print('[-] Host:\t %s (%s)' % (host, nm[host].hostname()))
6         print('[-] State:\t %s' % nm[host].state())
7         row_list.append([host, nm[host].hostname(), nm[host].state()
   ])

```

Listing 3.9: Code used for port scanning automation.

3.4 Laboratory environment

The testing of the realized architecture was carried out at the IoT laboratory created by the SmartData research group of the Politecnico di Torino. In this case, the testing environment had a greater number of devices than in the previous case, the system was configured following the approach proposed by Ren et al.[41]. The laboratory also allowed to test other more complex devices than the one analyzed at first. The analyzes carried out, extended to such a heterogeneous environment, were fundamental to validate the previously defined design choices.

3.4.1 Mon(IoT)r

The laboratory configuration follow the architecture presented by J. Ren et al. [41] in the Imperial College of London. This environment was used by the research team to conduct a multidimensional analysis of information exposure from different common IoT devices, highlighting differences related to different to different privacy regulation.

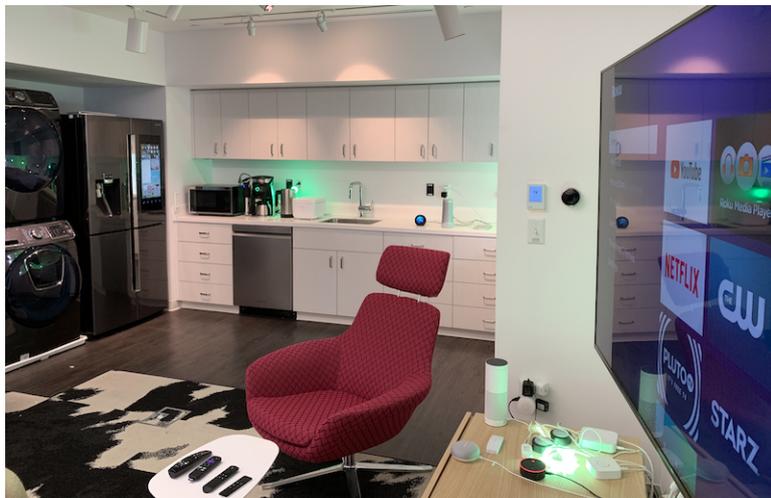


Figure 3.6: Mon(IoT)r laboratory - Imperial College London.

Network configuration

The configuration of the physical network interfaces is the following.

-
- eth0: This interface is used to intercept traffic that is was sent/received from the Internet. For this reason, the captures conducted on this interface have the purpose of obtaining information relating to the interactions concerning the device with the cloud.
 - switch-vlan10, switch-vlan11, switch-vlan12: These virtual LANs (VLAN10, VLAN11, VLAN12) represent the three networks in which the IoT devices will operate, and will be used to collect data exchanged within the LAN.
 - mirror-vlan10, mirror-vlan11, mirror-vlan12: Mirroring interfaces for the three VLANs.
 - copy-vlan10, copy-vlan11, copy-vlan12: Used to copy traffic which will then be analyzed by the system.
 - wlan0.1, wlan1.1, wlan2.1: 2.4 GHz WiFi network for traffic from VLAN10
 - wlan0.1, wlan1.2, wlan2.2: 5 GHz WiFi network for traffic from VLAN11
 - wlan0.3, wlan1.3, wlan2.3: 2.4GHz WiFi network for traffic from VLAN12

3.4.2 Devices

As already pointed out in this case, the devices under examination are 11 as shown in table 3.1, and they belong to different families of consumer IoT devices. For each device, the interaction tested inside the testbed is different, for example with regard to the smart cams during the capture phase motion recording mechanisms were triggered to start a series of more complex messages exchange (Transfer of images to the cloud and subsequent transmission to a companion app). For other devices, it was enough to give voice commands, while for others the only possible interaction was through a smartphone application.

3.4.3 Packet inspection

Once the traffic of a specific VLAN has been captured, it is analyzed by a python script whose main purpose is to classify the type of packet based on the transport protocol used and then extract all the related information. In addition to this function, all HTTP

	Smart cam	Home automation	Audio	Appliances
Device	- Xiaomi Smart cam - Netatmo Smart Cam - Wans Camera - Xiaomi Security Cam - Yi Camera	- TPLink bulb - Hue hub	-Amazon Echodot - Google home -Amazon Echospot	-Netatmo weather station
Interaction	Trigger movement detection, interact through smartphone companion application	Turn on/off	Voice command	Interact through smartphone companion application

Table 3.1: Devices present in the environment and related interactions tested.

NAME	IP	WI-FI	NETWORK	LAST SEEN
echodot	192.168.10.119	wlan0.1	unctrl	ONLINE
tplink-bubl	192.168.10.108	wlan0.1	unctrl	ONLINE
camera-xiaomi	192.168.10.113	wlan0.1	unctrl	ONLINE
camera-netatmo	192.168.10.111	-	unctrl	ONLINE
meteo-netatmo	192.168.10.114	wlan0.1	unctrl	ONLINE
camera-wans	192.168.10.100	-	unctrl	ONLINE
security-xiaomi	192.168.10.116	wlan0.1	unctrl	ONLINE
googlehome	192.168.10.104	wlan0.1	unctrl	ONLINE
echospot	192.168.12.101	wlan0.3	ctrl2	ONLINE
camera-yi	192.168.10.112	wlan0.1	unctrl	ONLINE
tplink-googlehome	192.168.10.106	wlan0.1	unctrl	ONLINE
tplink-echodot	192.168.10.105	wlan0.1	unctrl	ONLINE
tplink-cameras	192.168.10.107	wlan0.1	unctrl	ONLINE
hue-hub	192.168.10.109	-	unctrl	ONLINE

Figure 3.7: List of devices.

requests/responses are handled separately so that it is possible to reconstruct the dynamics of interactions. The steps followed by the software developed to inspect the data flow is shown in figure 3.8. The analysis at the application level is fundamental

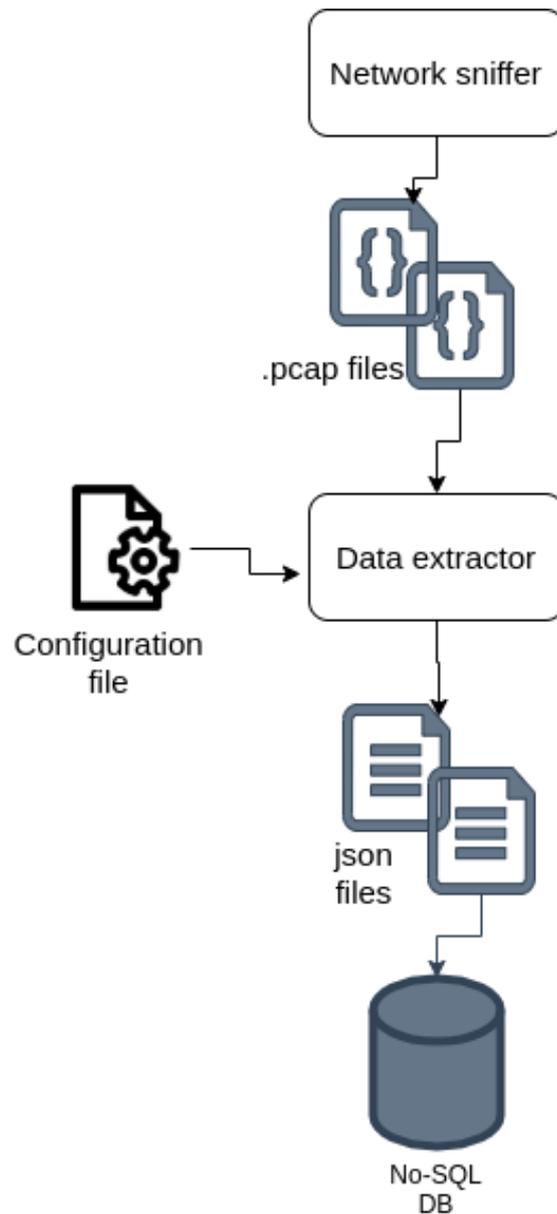


Figure 3.8: Data extraction stateflow.

to determine the messages that will then be received and sent by the honeypot, the ex-

haustive collection, at the transport protocol level, will allow the honeypot to compare the information exchanged in a controlled environment and those exchanged during a possible attack. Furthermore, the choice made leaves the possibility to update the testbed in such a way as to extend the inspection of packets at the application level to the most common protocols for the IoT mentioned in chapter 2. All the information analyzed and processed as explained above are associated with a device and cataloged by the product category.

3.5 Data storage and dictionary structure

The design choices related to the storage of the data acquired in the previous analyzes and therefore the creation of the dictionary that will then be used by the honeypot are illustrated below.

Database: RDBMS, no-SQL

Also in this case the choice of the DBMS was carried out following the requirements defined in the design phase. In particular, the particularities of the systems mainly used for data organization were analyzed. SQL-based DBMSs are aimed at managing and querying highly structured data, one of the characteristics of SQL databases is related to data normalization, reaching the normal Boyce-Codd form. This feature involves the creation of rigid schemes, which are difficult to alter. This characteristic is, perhaps, the main obstacle in the storage of heterogeneous data processed in the analyzes.

	SQL	no-SQL
Storage model	Tables with fixed structure	Document, key-value pair, graph, multivalues
Schema	Fixed	Flexible
Joins	Required	Not always required
Scaling	Vertical	Horizontal
Time to deployment	Medium	Fast

The reasons that led to the choice of a no-SQL database are mainly the following:

- Flexibility of the data model: fundamental for heterogeneous collections and for the possibility of adding new information to allow new functions to the scripts created. In particular, for the case study under examination, the entities to be modeled were similar but differed from each other for some fields. In the case of SQL, this solution could be solved by modeling either a different table for each type of protocol or by creating an all-inclusive table, containing all common and non-common fields, not always valued.
- Horizontal scaling: allows us to increase the volume of data collected without requiring major infrastructure updates. [42]
- Quick data query: The data is stored in an optimized way for the query by linking the necessary information already in the development phase.
- Reduction of the time required during the development and deployment phase.

Among the different types of no-SQL databases, MongoDB[43], a document oriented No-SQL database, was chosen.

MongoDB



Figure 3.9: MongoDB logo.

MongoDB is an open-source NoSQL database management system based on JSON documents, a text format that is agnostic to programming languages. Released in 2009, as NoSQL DBMS does not have a fixed schema, it supports several features, listed below[44]:

- Data query via REST API.

- Indexes: Like other RDBMSs, it supports indexing to optimize queries.
- Natively supports aggregations using the map-reduce pattern.

With reference to the system requirements defined during the design phase, MongoDB was chosen for its support for horizontal scalability, which makes it possible to extend the system to a set of testbeds used as data sources. Furthermore, the built-in support for aggregation operations is certainly interesting in view of further steps relating to the data processing phase. Finally, the possibility of significantly reducing the time required for the development phase was decisive in choosing MongoDB as a DBMS. It is also worth mentioning the ease of database replication, therefore of migration of the same from one server to another.

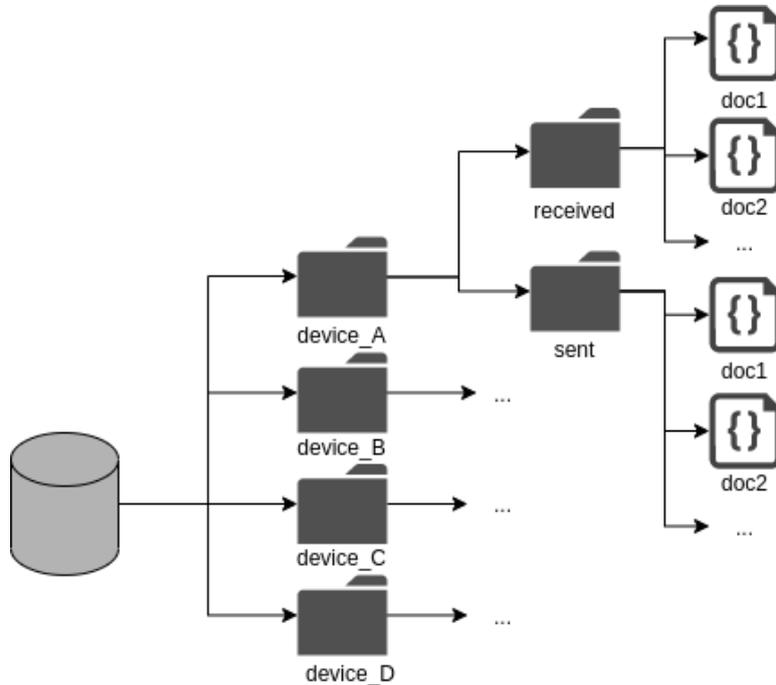


Figure 3.10: Structure of the database.

The database is organized as shown in figure 3.10. Each device constitutes a collection in itself, which in turn contains two collections, one for the messages received and one for those contained. Each document contains the information that has been extracted from the python script used for this purpose, in particular, it is possible to define

the information from which protocol. Since the captured packets are not necessarily structured in the same way in terms of the protocols used, different documents may contain different information. For this reason, the flexible structure of non-relational databases was the choice adopted in the design phase.

Chapter 4

Case study and results

4.1 Overview

In this section are reported the results obtained from the processing of the data collected by the testbed during the period May 2020 - December 2020. Some elaborations are used for information purposes, from these it is possible to obtain the spatial localization of the servers with which the devices communicate, as well as statistical data such as the protocols used by the devices present in the testbed.

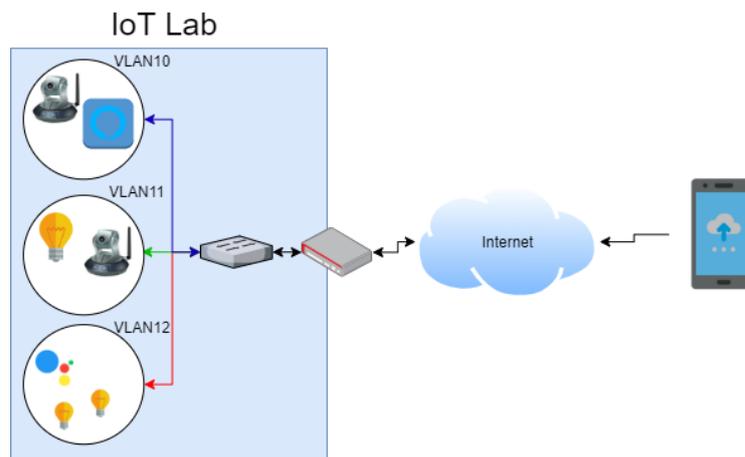


Figure 4.1: Testbed sample setup.

This information will not be used directly for the creation of the message dictio-

nary but is mainly used to get a wider view of the captured data, which can be easily interpreted in this way. Figure 4.2 shows in particular the geographical distribution of the servers with which the various devices communicate. This is an interesting starting point for further analysis that aims to investigate some aspects more closely related to privacy.



Figure 4.2: Spatial information about the servers contacted during the experiments.

The graph shown in figure 4.3 shows on the y-axis the number of packets detected, while on the x-axis the main transport protocols used are grouped by the direction of the traffic generated:

- Local
- From the local network to the outside (*outgoing traffic*)
- From outside to the local network (*ingoing traffic*)

The graph in figure 4.3 shows that with the exception of UDP traffic, most packets are exchanged within the local network. In the case of the UDP protocol, a significant difference can be noted in the direction of the packets sent, this is mainly due to the audio/video streaming activity of IoT devices such as smart-cam, security-cam, etc.

In the graph in figure 4.4, it is evident how the traffic division based on data direction is balanced over the three classes.

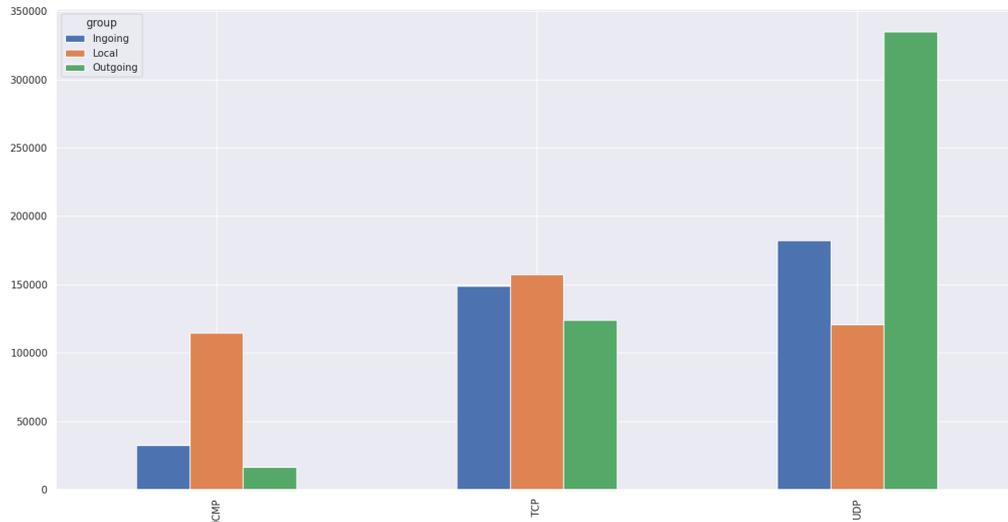


Figure 4.3: Statistical information on the traffic direction of the network divided by protocols used by the devices during the experiments.

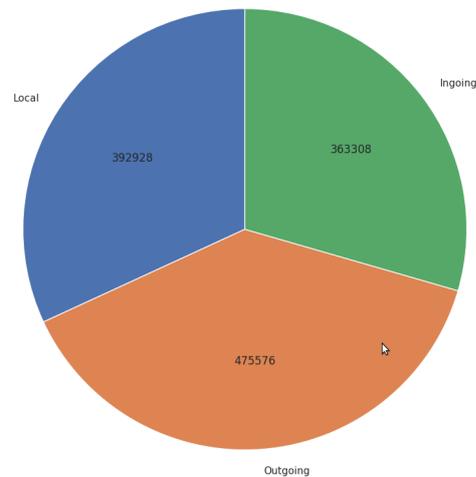


Figure 4.4: Statistical information on network traffic direction.

In the graph in figure 4.5, on the other hand, it can be seen that most of the traffic captured uses the UDP protocol as the transport protocol. This difference, as highlighted above, is due to the considerable volume of data that is transmitted during multimedia streaming, activity due to user interaction via smartphone, cloud, and smart-

camera.

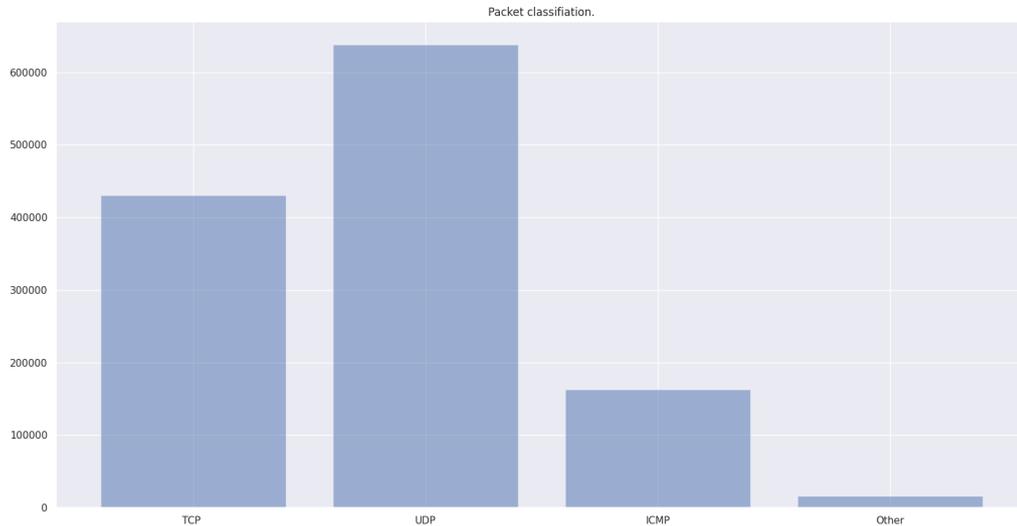
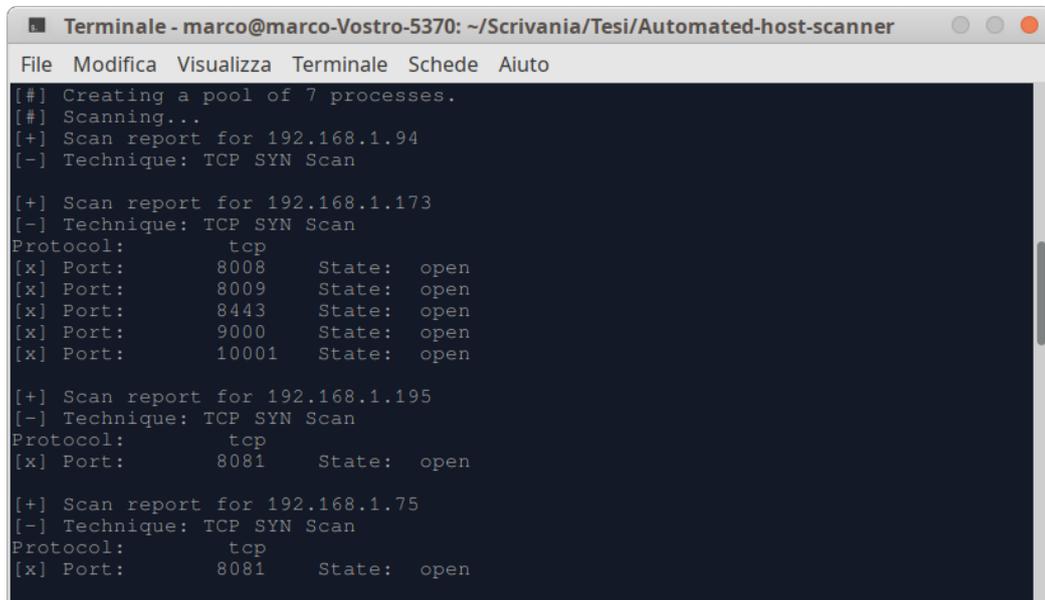


Figure 4.5: Statistical information on the protocols used by the devices within the test environment.

4.2 Active Analysis

The active analysis, as highlighted above, includes a set of techniques that provide for direct interaction with the various devices connected to the network, with the aim of obtaining general information regarding the state and behavior of the latter. This type of analysis is not strictly linked to the realization of the testbed, but provides an additional tool to the honeypot developers, for example being able to determine the open and used ports so as to be able to report the same behavior on the honeypot. The results obtained are also useful for the analysis combined with the data obtained with the passive analysis, thus being able to verify anomalous traffic (eg traffic directed to a closed door) but also to determine the specific use for each port (eg port X used for application data, Y port for updates or messages from/to the cloud). Figure 4.6 shows an example of the output generated by the software module responsible for scanning the ports of the devices connected to the network. In particular, in this example, it is possible to note, for a specific device, the IP address, the status of the ports, and the

related listening protocols. In figure 4.7 instead it is possible to see a possible output



```
Terminale - marco@marco-Vostro-5370: ~/Scrivania/Tesi/Automated-host-scanner
File Modifica Visualizza Terminale Schede Aiuto
[#] Creating a pool of 7 processes.
[#] Scanning...
[+] Scan report for 192.168.1.94
[-] Technique: TCP SYN Scan

[+] Scan report for 192.168.1.173
[-] Technique: TCP SYN Scan
Protocol:      tcp
[x] Port:      8008      State: open
[x] Port:      8009      State: open
[x] Port:      8443      State: open
[x] Port:      9000      State: open
[x] Port:      10001     State: open

[+] Scan report for 192.168.1.195
[-] Technique: TCP SYN Scan
Protocol:      tcp
[x] Port:      8081      State: open

[+] Scan report for 192.168.1.75
[-] Technique: TCP SYN Scan
Protocol:      tcp
[x] Port:      8081      State: open
```

Figure 4.6: Information collected by the automated port scanner.

on a CSV file containing the same information that is displayed on the terminal. This type of analysis is conducted periodically, with the aim of obtaining information from new devices but also of verifying a change in behavior due, for example, to an update of the IoT device firmware. The techniques chosen to conduct the active analysis did not determine a change in the normal operation of the devices, limiting themselves to performing scans that do not require a high computational load in the reception during the processing phase. Furthermore, an attempt was made not to congest the traffic on the local network. Also, in this case, the output obtained, organized in CSV files, reports the information in a labeled way, thus allowing further processing.

<u>IP address</u>	<u>hostname</u>	<u>Protocol</u>	<u>Port</u>	<u>State</u>
192.168.1.87	amazon-796b038e6.lan	tcp	8009	open
192.168.1.126	shelly1-772893.lan	tcp	80	open
192.168.1.129		tcp	80	open
192.168.1.129		tcp	515	open
192.168.1.129		tcp	631	open
192.168.1.129		tcp	5200	open
192.168.1.129		tcp	9100	open
192.168.1.135		tcp	8008	open
192.168.1.135		tcp	8080	open
192.168.1.135		tcp	9091	open
192.168.1.135		tcp	9998	closed
192.168.1.135		tcp	9999	closed
192.168.1.1		udp	1064	closed
192.168.1.1		udp	17185	closed
192.168.1.135		udp	123	open
192.168.1.135		udp	6002	closed
192.168.1.135		udp	49154	closed
192.168.1.87	amazon-796b038e6.lan	udp	902	closed
192.168.1.87	amazon-796b038e6.lan	udp	2002	closed
192.168.1.87	amazon-796b038e6.lan	udp	16938	closed
192.168.1.87	amazon-796b038e6.lan	udp	17417	closed
192.168.1.87	amazon-796b038e6.lan	udp	17468	closed
192.168.1.87	amazon-796b038e6.lan	udp	21333	closed
192.168.1.87	amazon-796b038e6.lan	udp	48078	closed
192.168.1.87	amazon-796b038e6.lan	udp	49192	closed

Figure 4.7: Part of the information gathered by the port scanner organized in CSV files.

4.3 Passive Analysis

The passive analysis involves the acquisition of network packets passing through the proxy. The choice of using a transparent passive capture on a proxy is determined by the need not to alter, in any way, the traffic that is generated by the devices on the network. As previously highlighted, the implementation of techniques such as the man in the middle attack has led to the malfunction of many more advanced devices. Such malfunctions are detected in all secure connections via SSL, due to mechanisms such as certificate pinning or simply certificate validation, controls present in most devices on the market. On the other hand, a real-time display of the content of the encrypted payload would have allowed increasing the size of the dictionary, and for this reason, this point represents one of the aspects to be developed in the future.

The application of more advanced techniques, linked for example to the exploitation of known vulnerabilities in communication protocols, does not meet the scalability requirements defined in the design phase. This study was carried out on devices connected via Wi-Fi, an attempt was made to leave the possibility of extending the data acquisition module to different protocols (e.g. ZigBee, Bluetooth), leaving the captured data processing module unchanged.

```
GET /f9f507e945c526717588d346cccf374a/command/ping HTTP/1.1
Host: 192.168.10.111
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp/4.2.2

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-type: application/json
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Expires: 0
Transfer-Encoding: chunked
Date: Thu, 29 Oct 2020 13:34:15 GMT
Server: lighttpd/1.4.35

{"local_url":"http://192.168.10.111/f9f507e945c526717588d346cccf374a","product_name":"Welcome Netatmo"}
```

Figure 4.8: HTTP stream captured during the experiments.

```
GET /f9f507e945c526717588d346cccf374a/live/files/high/live000439933.ts HTTP/1.1
User-Agent: Netatmo Video Player/3.1.2.2 (Linux;Android 7.0) ExoPlayerLib/2.10.7
Accept-Encoding: identity
Host: 192.168.10.111
Connection: Keep-Alive

...Wj.;B....\...w.G.B...
...% ao...^...{...z...o...6...U"...X...0...Gmx...o...)=...l...0~\...%i5g.K.Z.f...(. .n)
\...yI...n...Y...mh6...k...U...}r...
x.6<...&...E...T...&]e\...pE.]...K...
...*_...Fy...Y)*...z...KW;|u...wk..._H...N...n...]"s...joE...sSm; ;6=C...{...hZ.G...#.N.Rn;...*4k)
p.g...o...);|...8.../...&...T};f...-t...h...U...:...0...l...m...nIs.../...f...u...hg...~...N...
...&...&...u(4..2...Z.../...pC]+...3S...u...[.G...?U...d.*@...Rr.?|.].)....i|Zi\Y9i.zU...=j.B...
...U...b8#...oJ.b...Z...F...W.k.../W.S0... \...m...V...
...9... 'W... "u...w.W...z... \^...r...F...B.tG...{...W+...N.9...wW...w...u...U.V... \...
9...SAI#f...].]...wyy).|]...?V/...U...wu.s...V-#f...;...!d.W-^...V.z...l...Wr.y...z.j...!7t8...
+K...w)...
...3...sk...G...M.NK...pGZ...j...k.o.|...'.^...b... \...0...t...$;
9U...cn..._M...o...wF.t...uk...x...+...U...7...se.J...CI...O.B.7...&t...t.5$.#...I...-?.{...k... \l6...G...s...*.
6.|.wV...=...N..._].M...m...KA..6...y;|oL|rd...(...A&.hu...]=f...N.M.S...U...].)....h.L.b.$... \..._w!<...&[;u...>
```

Figure 4.9: Another fragment of the HTTP stream captured during the experiments.

Figure 4.8 and 4.9 shows an example of an HTTP stream between a smart cam and a smartphone. In particular, in the first image, the command sent by the smartphone as a request on the status of the room is shown in red, the relative response in blue. The following image shows the video/audio streaming request and the relative response in red.

4.4 Data extraction

This last software module is the core of the testbed and is responsible for processing the information captured in the previous analyzes, filtering the data by protocol level or by a specific protocol and by device, labeling the exchanged packets, and finally storing them in the no-SQL database.

The information is processed starting from the pcap files and configuration files generated by the previous modules. By means of appropriate parameters of the script, it is possible to specify individually, from which levels to extract the information that will subsequently be stored. It is possible to extract information from protocol headers such as TCP, UDP, ICMP, but also to process higher-level protocol information such as HTTP/HTTPS.

The module was designed to label the obtained data per device, but this clearly does not limit the ability to design a honeypot that uses messages from all devices stored in the database. This module does not need any additional information other than what is provided by the previous modules. This allows you to start an analysis in an unknown environment and still get to the formation of the dictionary. This particularity allows avoiding a detailed and in-depth study of the documentation (when available) of the various devices present in the test environment. It was possible to verify the considerable lack of technical documentation relating to the communication protocol used by various devices, and often when present not updated. However, when available, the documentation was used in the results validation phase.

Figure 4.10 shows the list of packet collections divided by device and direction (sent/received).

Figure 4.11 shows a document representing an HTTP response message. In particular, in this specific case, the information shown is extracted from different levels (TCP, IP, HTTP). Since different protocols may be present in different packages, it is possible that the collection is heterogeneous, for this reason, we have chosen to use non-relational databases. In this specific case it was of interest the messages exchanged through HTTP but in general, it is possible to obtain information from all the exchanged packets, increasing however the amount of information stored.

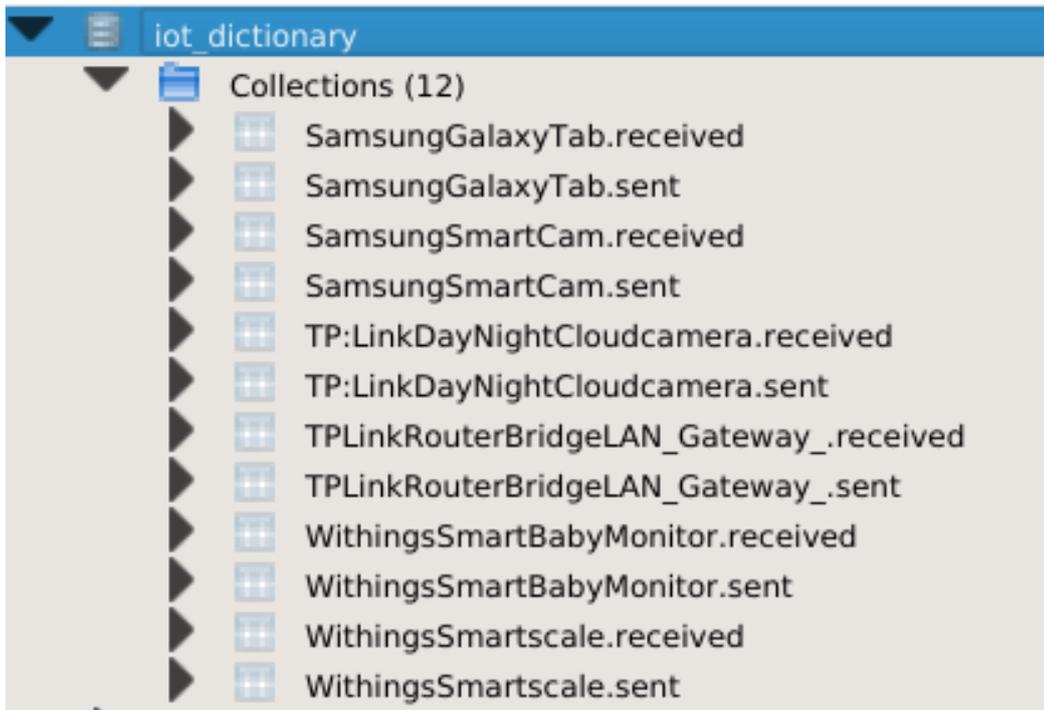


Figure 4.10: List of collections of received/sent messages for each device.

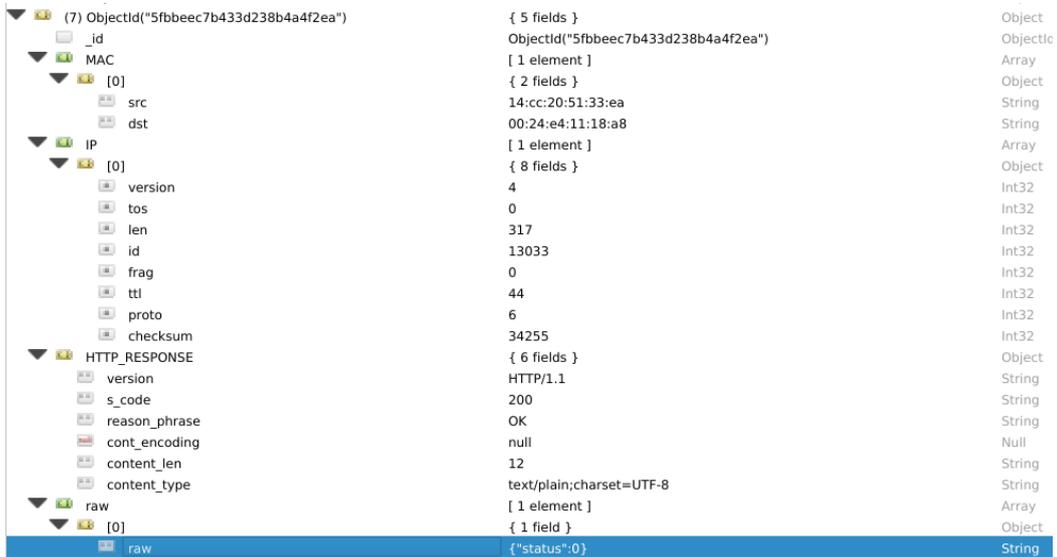


Figure 4.11: Example of a document part of the IoT dictionary.

4.5 Summary

In conclusion, taking into consideration three different devices, of growing complexity in terms of available features and therefore interactions, such as the Shelly-1, the Netatmo camera and the Alexa Echo Dot assistant, it was found that for simple devices such as Shelly-1 i messages can be grouped into two categories: action (from ON to OFF or vice versa) and status request. As for more complex devices such as the Netatmo camera, audio/video streaming and communications are added that are initiated from the camera following environmental events (e.g. motion detection). On the other hand, advanced devices such as Amazon Alexa involve a significant increase in the ports used, protocols used, hosts contacted, also due to the possibility of interacting with different devices belonging to the same network. From all this, it is easy to see that the creation of a dictionary for the first two categories is simpler and more sufficient than in the third case, where the complexity of the interactions makes it necessary not only a greater number of experiments to detect the greatest number of possible interactions but the use of different techniques for the realization of a honeypot. (E.g. machine learning).

Chapter 5

Conclusions and future work

5.1 Conclusion

The study conducted therefore proposes a methodology that can be applied with the aim of obtaining, in an automatic way, a dictionary of messages oriented to a chosen protocol. The added value of the data, once processed, and the possibility of filtering labeled traffic. In particular, the packet capture section uses techniques and tools already widely known to the research community, widely tested, in order to maintain the integrity of the data transmitted and received. Systems and approaches, such as the already mentioned MITMproxy, have been discarded due to interference with the devices under examination with the consequent possibility of data loss. However, there remains a point on which it is possible to improve the proposed model, trying to extract information on packets protected by encryption. Moreover, in this phase no dedicated hardware components were used, which would have limited the contexts of use. Overall, the system has proven capable of capturing, filtering, and cataloging application messages exchanged by different IoT devices. It was also possible to use the data stored by the system to conduct replay attacks, which in most cases were successful. Furthermore, at the moment the approach has been tested exclusively by intercepting traffic conveyed via Wi-Fi or wired network, but it is still possible to extend the work done for devices that use different protocols such as Bluetooth, ZigBee, etc, by modifying the module responsible for capturing network packets. The proposed methodology can be used in different environments and not necessarily for IoT devices. The final result is a NoSQL database from which it is possible to obtain not only information

about the syntax used for exchanging messages from the various devices but also general information relating to the ports, the protocols used, the servers contacted by the devices. All this information will be necessary for the future development of a medium/high interaction honeypot, starting from a dictionary with all the possible messages and responses exchanged by the devices.

5.2 Future work

During the development of this thesis work, several ideas emerged that have not been deepened. These certainly include the possibility of using the information stored in the dictionary to perform behavioral fingerprinting of devices on the network. This can be achieved by using unsupervised machine learning algorithms, such as gradient boosting or k-nearest neighbor. Some studies have already been conducted in this area and the results are encouraging [45] [46] [47]. This investigation can also be conducted without the ability to extract information from encrypted traffic. A possible solution is to use Shannon's entropy as a feature, defined by the formula below, this allows to obtain information about the amount of information contained in the message without necessarily decrypting it. Encrypted or encoded messages will contain more information than a plain text message.

$$H = - \sum_i p_i \log_b p_i$$

Similarly, traffic captured under controlled conditions could be used as a sample to identify variations with respect to the normal behavior of devices, in order to identify the presence of malicious traffic caused by an attack in progress. Another aspect that could be further investigated is that linked to privacy and confidentiality, conducting new statistical analyzes about the servers that are contacted by the devices and the information that is transmitted, the amount of traffic that is protected by encryption, authentication, etc. These are just some of the possible insights that could be conducted once a sufficiently rich dataset of labeled data is available.

Chapter 6

Appendix A

6.1 System setup

6.1.1 Access Point automated setup

```
1 #!/bin/bash
2
3 if [ $# != 3 ]; then
4     echo "[ERROR] Illegal number of parameters."
5     echo "[-] Syntax: recon.sh [interface] [SSID] [PASSPHRASE]"
6     echo '[-] Example: recon.sh wlan0 my_ssid my_passhphrase '
7     exit -1
8 fi
9
10 if [[ $1 == '--help' ]]; then
11     echo "[-] Syntax: recon.sh [interface] [SSID] [PASSPHRASE]"
12     echo '[-] Example: recon.sh wlan0 my_ssid my_passhphrase '
13     exit 0
14 fi
15
16 echo "[INFO] Creating access point through $1"
17
18 # Interface check
19 ip link show $1 > /dev/null 2>&1
20
21 ret_code=$?
```

```
22
23 if [[ $ret_code != 0 ]]; then
24     echo "[ERROR] Device \"$1\" does not exist. Aborting."
25     exit -1
26 fi
27
28 if [[ -d "./tmp" ]]; then
29     sudo rm ./tmp/*
30 else
31     mkdir ./tmp
32 fi
33
34 sudo cp ./conf/hostapd ./tmp/hostapd
35 sudo cp ./conf/dnsmasq.conf ./tmp/dnsmasq.conf
36 sudo cp ./conf/dhcpd.conf ./tmp/dhcpd.conf
37 sudo cp ./conf/dhcpd.conf ./tmp/dhcpd.conf
38
39 sudo apt-get install -y hostapd
40
41 if [[ $? > 0 ]]
42 then
43     echo "[ERROR] Error during hostapd installation. Aborting."
44     exit
45 else
46     echo "[INFO] hostapd successfully installed."
47 fi
48
49 sudo systemctl unmask hostapd
50 sudo systemctl enable hostapd
51
52
53 sudo apt install -y dnsmasq
54
55 if [[ $? > 0 ]]
56 then
57     echo "[ERROR] Error during dnsmasq installation. Aborting."
58     exit
59 else
60     echo "[INFO] dnsmasq successfully installed."
61 fi
```

```
62
63 sudo DEBIAN_FRONTEND=noninteractive apt install -y netfilter-
    persistent iptables-persistent
64
65 if [[ $? > 0 ]]
66 then
67     echo "[ERROR] Error during iptables/netfilter installation.
    Aborting."
68     exit
69 else
70     echo "[INFO] iptables/netfilter successfully installed."
71 fi
72
73
74 echo "interface $1
75     static ip_address=192.168.4.1/24
76     nohook wpa_supplicant" >> ./tmp/dhcpd.conf
77
78 cp ./tmp/dhcpd.conf /etc/dhcpd.conf
79
80
81 sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
82 # Enable ipv4 forwarding
83 sudo sysctl -w net.ipv4.ip_forward=1
84
85 sudo netfilter-persistent save
86
87 sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
88
89 echo "
90
91 interface=$1 # Listening interface
92 dhcp-range=192.168.4.2,192.168.4.20,24h
93             # Pool of IP addresses served via DHCP
94 domain=wlan # Local wireless DNS domain
95 address=/gw.wlan/192.168.4.1
96             # Alias for this router
97 " >> tmp/dnsmasq.conf
98
99 sudo cp tmp/dnsmasq.conf /etc/dnsmasq.conf
```

```

100
101 sudo rfkill unblock wlan
102
103 # Setup hostapd configuration file
104 sudo echo "
105 interface=$1
106 ssid=$2
107 driver=nl80211
108 hw_mode=g
109 channel=7
110 macaddr_acl=0
111 auth_algs=1
112 ignore_broadcast_ssid=0
113 wpa=2
114 wpa_passphrase=$3
115 wpa_key_mgmt=WPA-PSK
116 wpa_pairwise=TKIP
117 rsn_pairwise=CCMP" > tmp/hostapd.conf
118
119 sudo cp ./tmp/hostapd.conf /etc/hostapd/hostapd.conf
120
121 echo "[DONE] Reboot the system to start the AP."
122 echo "[DONE] Run \"sudo sysctl -w net.ipv4.ip_forward=1\" after
    reboot."

```

Listing 6.1: Bash script for Access Point automated setup.

6.2 Active Analysis

6.2.1 Port scanning automation

```

1 #!/usr/bin/env python
2 import sys
3 import nmap
4 import csv
5 import signal
6 from datetime import datetime
7 from multiprocessing import Pool, Lock
8

```

```

9 class TimeoutException(Exception):
10     pass
11
12 global filename, s_timeout
13 s_timeout = 3*60 # 3min timeout for each scan
14 filename = 'recon_report_ports_'+datetime.now().strftime("%d-%m-%Y_%
    H-%M-%S.csv")
15
16
17 def timeout_handler(signum, frame):
18     raise TimeoutException('Timeout')
19
20 def init_worker(l):
21     global lock
22     lock = l
23
24 # Perform pingsweep with DNS resolution
25 # ICMP echo + UDP discovery on port 21,23,80,3389,8081
26 def pingsweep_resolve(subnet):
27     print('[+] Scan report for pingsweep over:', subnet)
28
29     row_list = []
30     header = ['IP address', 'Hostname', 'State']
31     row_list.append(header)
32     nm = nmap.PortScanner()
33     nm.scan(hosts=subnet, arguments='--max-parallelism 100 -sP -PE -
    PA 21,23,80,3389,8081')
34
35     print('Found %d devices' % len(nm.all_hosts()))
36
37     for host in nm.all_hosts():
38         print('-----')
39         print('[-] Host:\t %s (%s)' % (host, nm[host].hostname()))
40         print('[-] State:\t %s' % nm[host].state())
41         row_list.append([host, nm[host].hostname(), nm[host].state()
    ])
42
43     if len(sys.argv) == 3 and sys.argv[2] == '-f': #write output
    into .csv

```

```

44     filename = 'recon_report_'+datetime.now().strftime("%d-%m-%
Y_%H-%M-%S.csv")
45
46     with open(filename, 'w', newline='') as file:
47         writer = csv.writer(file)
48         writer.writerows(row_list)
49
50     return nm.all_hosts()
51
52 # Perform pingsweep without DNS resolution
53 # ICMP echo + UDP discovery on port 21,23,80,3389,8081
54 def pingsweep(subnet):
55     print('[+] Scan report for pingsweep (No DNS resolution) over:',
56           subnet)
57
58     nm = nmap.PortScanner()
59     nm.scan(hosts=subnet, arguments='--max-parallelism 100 -n -sP -
PE -PA 21,23,80,3389,8081')
60
61     print('Found %d devices' % len(nm.all_hosts()))
62
63     for host in nm.all_hosts():
64         print('-----')
65         print('[-] Host:\t %s' % (host))
66         print('[-] State:\t %s' % nm[host].state())
67         print('-----')
68
69     return nm.all_hosts()
70
71 # Perform custom scan
72 def perform_scan(target, args):
73     nm = nmap.PortScanner()
74     nm.scan(hosts=target, arguments=args)
75
76     return nm
77
78 # Perform a TCP SYN scan on a target
79 def tcp_syn_scan(target):
80     output = ''
81     retval = 0

```

```

81 signal.signal(signal.SIGALRM, timeout_handler)
82 signal.alarm(s_timeout)
83
84 try:
85     row_list = []
86     nm = perform_scan(target, '-Pn -sT')
87
88     output += '[+] Scan report for ' + str(target) + '\n'
89     output += '[-] Technique: TCP SYN Scan\n'
90
91     for proto in nm[target].all_protocols():
92         output += 'Protocol:\t %s\n' % proto
93
94         lport = nm[target][proto].keys()
95
96         for port in lport:
97             output += '[x] Port:\t%s\tState:\t%s\n' % (port, nm[
98 target][proto][port]['state'])
99             row_list.append([target, nm[target].hostname(),
100 proto, port, nm[target][proto][port]['state']])
101
102         if len(sys.argv) == 3 and sys.argv[2] == '-f': #write output
103 into .csv
104             with open(filename, 'a', newline='') as file:
105                 writer = csv.writer(file)
106
107                 lock.acquire()
108                 writer.writerow(row_list)
109                 lock.release()
110
111 except TimeoutException:
112     print('[Error] Scan abort - timeout')
113     print('[Error] TCP SYN Scan - timeout for host:', target)
114     retval = -1
115     error_line = '[Error] TCP SYN Scan - timeout for host: '+
116 target
117
118     lock.acquire()
119     error_log.write(error_line+'\n')
120     lock.release()

```

```

117
118     except Exception as exc:
119         print(exc)
120         retval = -1
121     finally:
122         print(output)
123         signal.alarm(0)
124
125     return retval
126
127 # Perform a TCP connect scan on a target
128 def tcp_connect_scan(target):
129
130     print('[+] Scan report for ', target)
131     print('[-] Technique: TCP Connect Scan')
132
133     nm = perform_scan(target, '-Pn -sT')
134
135     for proto in nm[target].all_protocols():
136         print('[x] Protocol:\t %s' % proto)
137
138         lport = nm[target][proto].keys()
139
140         for port in lport:
141             print('[x] Port:\t%s\tState:\t%s' % (port, nm[target][
142                 proto][port]['state']))
143
144 # Perform a SCTP scan on a target
145 def sctp_connect_scan(target):
146
147     print('[+] Scan report for ', target)
148     print('[-] Technique: SCTP Connect Scan')
149
150     nm = perform_scan(target, '-Pn -sY')
151
152     for proto in nm[target].all_protocols():
153         print('[x] Protocol:\t %s' % proto)
154
155         lport = nm[target][proto].keys()

```

```

156     for port in lport:
157         print('[x] Port:\t%s\tState:\t%s' % (port, nm[target][
proto][port]['state']))
158
159 # Perform an UDP scan on a target
160 def udp_scan(target):
161     output = ''
162     signal.signal(signal.SIGALRM, timeout_handler)
163     signal.alarm(s_timeout)
164     retval = 0
165
166     try:
167         row_list = []
168         nm = perform_scan(target, '-Pn -sU')
169
170         output += '[+] Scan report for ' + str(target) + '\n'
171         output += '[-] Technique: UDP Scan\n'
172
173         for proto in nm[target].all_protocols():
174             output += 'Protocol:\t %s\n' % proto
175
176             lport = nm[target][proto].keys()
177
178             for port in lport:
179                 output += '[x] Port:\t%s\tState:\t%s\n' % (port, nm[
target][proto][port]['state'])
180                 row_list.append([target, nm[target].hostname(),
proto, port, nm[target][proto][port]['state']])
181
182         if len(sys.argv) == 3 and sys.argv[2] == '-f': #write output
into .csv
183             with open(filename, 'a', newline='') as file:
184                 writer = csv.writer(file)
185                 lock.acquire()
186                 writer.writerows(row_list)
187                 lock.release()
188
189     except TimeoutException:
190         print('[Error] Scan abort - timeout')
191         print('[Error] UDP - timeout for host:', target)

```

```

192     error_line = '[Error] TCP SYN Scan - timeout for host: '+
target
193     lock.acquire()
194     error_log.write(error_line+'\n')
195     lock.release()
196     retval = -1
197
198     except Exception as exc:
199         print(exc)
200         retval = -1
201     finally:
202         print(output)
203         signal.alarm(0)
204
205     return retval
206
207
208 def standard_scan(subnet):
209     t_start = datetime.now()
210
211     hosts = pingsweep_resolve(subnet)
212     error_log = open('recon_error_log_'+datetime.now().strftime("%d
-%m-%Y_%H-%M-%S.txt"), 'w')
213
214     global n_scanned_port
215
216     if len(sys.argv) == 3 and sys.argv[2] == '-f': #write output
into .csv
217         mkfile()
218
219     print('[#] Creating a pool of '+str(min(8, len(hosts)))+
processes.')
220     print('[#] Scanning...')
221
222     l = Lock()
223     tcp_scan_pool = Pool(initializer=init_worker, initargs=(l,),
processes = min(8, len(hosts))) # Test with 8 processes pool
224     udp_scan_pool = Pool(initializer=init_worker, initargs=(l,),
processes = min(8, len(hosts)))
225

```

```

226     n_failures = 0
227     n_scanned_prot = 0
228
229     res_tcp = tcp_scan_pool.map(tcp_syn_scan, hosts)
230     res_udp = udp_scan_pool.map(udp_scan, hosts)
231
232     for res in res_tcp:
233         n_scanned_prot+=1
234         if res == -1:
235             n_failures+=1
236
237     for res in res_udp:
238         n_scanned_prot+=1
239         if res == -1:
240             n_failures+=1
241
242     tcp_scan_pool.close()
243     udp_scan_pool.close()
244
245     tcp_scan_pool.join()
246     udp_scan_pool.join()
247
248     error_log.close()
249     t_end = datetime.now()
250     delta = t_end - t_start
251
252     print('[X] Job completed. Elapsed time: ' + str(delta.seconds) +
253           ' seconds.')
254     print('[X] Scans:', n_scanned_prot)
255     print('[X] Failures:', n_failures)
256
257 def mkfile():
258     header = ['IP address', 'hostname', 'Protocol', 'Port', 'State']
259     row_list = []
260     row_list.append(header)
261
262     with open(filename, 'w', newline='') as file:
263         writer = csv.writer(file)
264         writer.writerows(row_list)

```

```

264
265 def main():
266     if len(sys.argv) != 2 and len(sys.argv) != 3:
267         print('syntax: python3 recon.py subnet [-f]')
268         print('Use -f to save output into files')
269         print('Example: python3 recon.py 192.168.1.1/24 -f')
270         exit(-1)
271
272     standard_scan(sys.argv[1])
273
274 if __name__ == "__main__":
275     main()

```

Listing 6.2: Python script for host discovery and port scanning automation.

6.2.2 Data extraction

```

1 import json
2 import pymongo
3 import os.path
4
5 from configparser import ConfigParser
6 from scapy.all import *
7 from scapy.layers.http import HTTPRequest, HTTPResponse
8
9 FIN = 0x01
10 SYN = 0x02
11 RST = 0x04
12 PSH = 0x08
13 ACK = 0x10
14 URG = 0x20
15 ECE = 0x40
16 CWR = 0x80
17
18 layers = []
19
20 myclient = pymongo.MongoClient("mongodb://localhost:27017/")
21 mdb = myclient["iot_dictionary"]
22
23 config_object = ConfigParser(delimiters=('='))

```

```

24
25
26 def extract_data(pkt):
27     db_entry = {}
28     global layers
29     global config_object
30     isInLayers = False
31
32     if len(layers) == 0:
33         layers = ["HTTP", "TCP", "UDP", "ICMP", "AH", "ESP", "TLS",
34 "HTTP"]
35
36     if IP in pkt:
37         MAC_src = pkt.src
38         MAC_dst = pkt.dst
39
40         ip_src = pkt[IP].src
41         ip_dst = pkt[IP].dst
42
43         ip_v = pkt[IP].version
44         ip_tos = pkt[IP].tos
45         ip_len = pkt[IP].len
46         ip_id = pkt[IP].id
47         ip_frag = pkt[IP].frag
48         ip_ttl = pkt[IP].ttl
49         ip_proto = pkt[IP].proto
50         ip_chksun = pkt[IP].chksun
51         ip_options = pkt[IP].options
52         ip_flags = pkt[IP].flags
53
54         db_entry['MAC'] = []
55         db_entry['IP'] = []
56
57         db_entry['MAC'].append({
58             'src': MAC_src,
59             'dst': MAC_dst
60         })
61
62         db_entry['IP'].append({
63             'version': ip_v,

```

```

63         'tos': ip_tos,
64         'len': ip_len,
65         'id': ip_id,
66         #     'flags': ip_flags,
67         'frag': ip_frag,
68         'ttl': ip_ttl,
69         'proto': ip_proto,
70         'checksum': ip_chksum,
71         #     'options': ip_options
72     })
73
74     if ESP in pkt and layers.__contains__('ESP'):
75         isInLayers = True
76
77         esp_spi = pkt[ESP].spi
78         esp_seq = pkt[ESP].seq
79         esp_data = pkt[ESP].data
80
81         db_entry['ESP'] = []
82         db_entry['ESP'].append({
83             'spi': esp_spi,
84             'seq': esp_seq,
85             'data': str(esp_data)
86         })
87
88     if AH in pkt and layers.__contains__('AH'):
89         isInLayers = True
90
91         ah_nh = pkt[AH].nh
92         ah_payloadlen = pkt[AH].payloadlen
93         ah_reserved = pkt[AH].reserved
94         ah_spi = pkt[AH].spi
95         ah_icv = pkt[AH].icv
96         ah_padding = pkt[AH].padding
97
98         db_entry['AH'] = []
99         db_entry['AH'].append({
100             'spi': ah_spi,
101             'icv': ah_icv,
102             'padding': ah_padding,

```

```

103         'nh': ah_nh,
104         'payload_len': ah_payloadlen,
105         'reserved': ah_reserved
106     })
107
108     if TCP in pkt and layers.__contains__('TCP'):
109         isInLayers = True
110
111         tcp_seq = pkt[TCP].seq
112         tcp_ack = pkt[TCP].ack
113         tcp_dataofs = pkt[TCP].dataofs
114         tcp_reserved = pkt[TCP].reserved
115         tcp_window = pkt[TCP].window
116         tcp_chksum = pkt[TCP].chksum
117         tcp_flags = pkt[TCP].flags
118         tcp_sport = pkt[TCP].sport
119         tcp_dport = pkt[TCP].dport
120         tcp_urgptr = pkt[TCP].urgptr
121         tcp_options = pkt[TCP].options
122         payload_guess = pkt[TCP].payload_guess
123
124         db_entry['TCP'] = []
125         db_entry['TCP'].append({
126             'seq': tcp_seq,
127             'ack': tcp_ack,
128             'dataofs': tcp_dataofs,
129             'reserved': tcp_reserved,
130             'window': tcp_window,
131             'checksum': tcp_chksum,
132             's_port': tcp_sport,
133             'd_port': tcp_dport,
134             'urg_ptr': tcp_urgptr,
135             # 'options': tcp_options,
136             # 'payload_guess': payload_guess
137         })
138
139         flags = []
140         if tcp_flags & FIN:
141             flags.append("FIN")
142         if tcp_flags & SYN:

```

```

143         flags.append("SYN")
144     if tcp_flags & RST:
145         flags.append("RST")
146     if tcp_flags & PSH:
147         flags.append("PSH")
148     if tcp_flags & ACK:
149         flags.append("ACK")
150     if tcp_flags & URG:
151         flags.append("URG")
152     if tcp_flags & ECE:
153         flags.append("ECE")
154     if tcp_flags & CWR:
155         flags.append("CWR")
156     db_entry['TCP'].append({
157         'flags': flags
158     })
159
160 elif UDP in pkt and layers.__contains__('UDP'):
161     isInLayers = True
162
163     udp_len = pkt[UDP].len
164     udp_checksum = pkt[UDP].len
165     udp_sport = pkt[UDP].sport
166     udp_dport = pkt[UDP].dport
167     payload_guess = pkt[UDP].payload_guess
168
169     db_entry['UDP'] = []
170     db_entry['UDP'].append({
171         's_port': udp_sport,
172         'd_port': udp_dport,
173         'len': udp_len,
174         'checksum': udp_checksum,
175         # 'payload_guess': payload_guess
176     })
177
178 elif ICMP in pkt and layers.__contains__('ICMP'):
179     isInLayers = True
180
181     icmp_type = pkt[ICMP].type
182     icmp_code = pkt[ICMP].code

```

```

183     icmp_chksum = pkt[ICMP].chksum
184     icmp_id = pkt[ICMP].id
185     icmp_seq = pkt[ICMP].seq
186     class_guess = pkt[ICMP].guess_payload_class
187
188     db_entry['ICMP'] = []
189     db_entry['ICMP'].append({
190         'type': icmp_type,
191         'code': icmp_code,
192         'checksum': icmp_chksum,
193         'id': icmp_id,
194         'seq': icmp_seq,
195         # 'class_guess': class_guess
196     })
197
198     if TLS in pkt and layers.__contains__('TLS'):
199         isInLayers = True
200
201         tls_type = pkt[TLS].type
202         tls_version = pkt[TLS].version
203         tls_len = pkt[TLS].len
204         tls_iv = pkt[TLS].iv
205         tls_msg = pkt[TLS].msg
206
207         db_entry['TLS'] = []
208         db_entry['TLS'].append({
209             'type': tls_type,
210             'version': tls_version,
211             'len': tls_len,
212             'iv': str(tls_iv),
213             'msg': str(tls_msg),
214         })
215
216     if pkt.haslayer(HTTPRequest) and layers.__contains__('HTTP')
:
217         isInLayers = True
218
219         host = pkt[HTTPRequest].Host.decode()
220         path = pkt[HTTPRequest].Path.decode()
221         url = host + path

```

```

222     method = pkt[HTTPRequest].Method.decode()
223     req_version = pkt[HTTPRequest].Http_Version.decode()
224     user_agent = pkt[HTTPRequest].User_Agent
225     if user_agent != None: user_agent = user_agent.decode()
226
227     accept = pkt[HTTPRequest].Accept
228     if accept != None: accept = accept.decode()
229
230     accept_lang = pkt[HTTPRequest].Accept_Language
231     if accept_lang != None: accept_lang = accept_lang.decode
232
233     cookie = pkt[HTTPRequest].Cookie
234     if cookie != None: cookie = cookie.decode()
235
236     conn = pkt[HTTPRequest].Connection
237     if conn != None: conn = conn.decode()
238
239     db_entry['HTTP_REQUEST'] = []
240     db_entry['HTTP_REQUEST'] = ({
241         'host': host,
242         'path': path,
243         'url': url,
244         'method': method,
245         'version': req_version,
246         'user_agent': user_agent,
247         'accept': accept,
248         'accept_lang': accept_lang,
249         'cookie': cookie,
250         'conn': conn
251     })
252
253     if pkt.haslayer(HTTPResponse) and layers.__contains__('HTTP
254     '):
255         isInLayers = True
256
257         version = pkt[HTTPResponse].Http_Version.decode()
258         s_code = pkt[HTTPResponse].Status_Code.decode()
259         s_code_s = pkt[HTTPResponse].Reason_Phase.decode()
260         cont_encod = pkt[HTTPResponse].Content_Encoding

```

```

260         if cont_encod != None: cont_encod = cont_encod.decode()
261         cont_len = pkt[HTTPResponse].Content_Length
262         if cont_len != None: cont_len = cont_len.decode()
263         cont_type = pkt[HTTPResponse].Content_Type
264         if cont_type != None: cont_type = cont_type.decode()
265
266         db_entry['HTTP_RESPONSE'] = []
267         db_entry['HTTP_RESPONSE'] = ({
268             'version': version,
269             's_code': s_code,
270             'reason_phrase': s_code_s,
271             'cont_encoding': cont_encod,
272             'content_len': str(cont_len),
273             'content_type': str(cont_type)
274         })
275
276         if pkt.haslayer(Raw):
277             raw = pkt[Raw].load
278             db_entry['raw'] = []
279
280             if (pkt.haslayer(HTTPResponse) or pkt.haslayer(
281 HTTPRequest)):
282                 try: # Try to decode the payload
283                     db_entry['raw'].append({
284                         'raw': raw.decode()
285                     })
286                 except Exception as exc:
287                     db_entry['raw'].append({
288                         'raw': str(raw)
289                     })
290                 else:
291                     db_entry['raw'].append({
292                         'raw': str(raw)
293                     })
294
295         if (isInLayers):
296             json_entry = json.dumps(db_entry, indent=4)
297             print(json_entry)
298
299         devices = config_object['DEVICES']

```

```

299
300     if MAC_src in devices:
301         device_col = mdb[devices[MAC_src]]
302         sent_msg_col = device_col['sent']
303         retVal = sent_msg_col.insert_one(db_entry)
304     if MAC_dst in devices:
305         device_col = mdb[devices[MAC_dst]]
306         received_msg_col = device_col["received"]
307         retVal = received_msg_col.insert_one(db_entry)
308
309
310 def sniff_offline(entry, isdir):
311     load_layer('tls')
312
313     if (isdir):
314         for filename in os.listdir(entry):
315             if filename.endswith(".pcap"):
316                 print("[INFO] Extracting features from: ", filename)
317                 sniff(offline=entry + "/" + filename, prn=
extract_data, store=False)
318                 print("")
319     else:
320         sniff(offline=entry, prn=extract_data, store=False)
321
322
323 def show_usage():
324     print("PCAP data extractor")
325     print("Usage: python dataExtractor.py pcap_file.pcap [flags]")
326     print("Flags:")
327     print("[-h/--HTTP]\tExtract data from HTTP packets.")
328     print("[-t/--TCP]\tExtract data from TCP packets.")
329     print("[-u/--UDP]\tExtract data from UDP packets.")
330     print("[-i/--ICMP]\tExtract data from ICMP packets.")
331     print("[-a/--ESP]\tExtract data from ESP packets.")
332     print("[-a/--AH]\tExtract data from AH packets.")
333     print("[-s/--TLS]\tExtract data from TLS packets.")
334     print("[-help]\tShow usage and options.")
335
336
337 def main():

```

```

338 flags = []
339 global layers
340 global config_object
341
342 # Read config.ini file
343 if os.path.isfile("devices.ini"):
344     config_object.read("devices.ini")
345 else:
346     print("File device.ini not present")
347     exit(-1)
348
349 if (len(sys.argv) < 2):
350     show_usage()
351     exit(-1)
352
353 if (len(sys.argv) > 2):
354     for idx in range(2, len(sys.argv)):
355         flags.append(sys.argv[idx])
356
357 if flags.__contains__("--help"):
358     show_usage()
359     exit(0)
360 if flags.__contains__("--HTTP") or flags.__contains__("--h"):
361     layers.append('HTTP')
362 if flags.__contains__("--TCP") or flags.__contains__("--t"):
363     layers.append('TCP')
364 if flags.__contains__("--UDP") or flags.__contains__("--u"):
365     layers.append('UDP')
366 if flags.__contains__("--ICMP") or flags.__contains__("--i"):
367     layers.append('ICMP')
368 if flags.__contains__("--ESP") or flags.__contains__("--e"):
369     layers.append('ESP')
370 if flags.__contains__("--AH") or flags.__contains__("--a"):
371     layers.append('AH')
372 if flags.__contains__("--TLS") or flags.__contains__("--s"):
373     layers.append('TLS')
374
375 entry = sys.argv[1]
376
377 try:

```

```

378     print("[INFO] Start analyzing pcap: ", entry)
379     if (os.path.isdir(entry)):
380         sniff_offline(entry, isdir=True)
381     elif (os.path.isfile(entry)):
382         sniff_offline(entry, isdir=False)
383     else:
384         raise ValueError("Illegal parameter: " + entry)
385 except Exception as exc:
386     print("[ERROR] ", exc)
387     traceback.print_exc()
388
389
390 if __name__ == "__main__":
391     main()

```

Listing 6.3: Python script for network traffic sniffing with Scapy.

6.3 Passive Analysis

6.3.1 Network sniffing automation

```

1  #!/usr/bin/python
2
3  from scapy.all import *
4  from datetime import datetime
5  import netifaces
6  import sys
7
8  def analyze_packet(packet):
9      ts = datetime.now()
10     print("\n" * 2)
11     print("-----")
12     print(ts.strftime("%m/%d/%Y, %H:%M:%S"))
13     packet.show()
14     print("-----")
15
16     write_packet(packet, sys.argv[2])
17
18 def write_packet(packet, file):

```

```

19     wrpcap(file, packet, append=True)
20
21 def interface_check(interface):
22     try:
23         addr = netifaces.ifaddresses(interface)
24     except ValueError as val_exc:
25         print('[Error] Bad interface name ('+interface+')')
26         return -1
27     except Exception as exc:
28         print('[Error] '+exc)
29         return -1
30     return not netifaces.AF_INET in addr
31
32 def main():
33
34     print('[INFO] Scapy Sniffer')
35     print('[INFO] Author: Marco A. Greco')
36     print('[INFO] Start sniffing on', sys.argv[1])
37     pck_threshold = 0
38
39     if(len(sys.argv) != 3 and len(sys.argv) != 4):
40         print('[Error] Wrong syntax.')
41         print('[Error] Example: python3 sniffer.py wlan0 output.pcap
42 [packet_count]')
43         exit(-1)
44
45     if(len(sys.argv) == 4):
46         if(sys.argv[3].isdecimal()):
47             pck_threshold = int(sys.argv[3])
48             print('[INFO] Threshold: '+str(pck_threshold)+' packages
49 ')
50
51     if interface_check(sys.argv[1]):
52         print('[Error] Interface '+sys.argv[1]+" is not online.")
53         print('[Error] Aborting.')
54         exit(-1)
55
56     sniff(iface = sys.argv[1], prn=analyze_packet, count=
57 pck_threshold)

```

```
56  
57 if __name__ == '__main__':  
58     main()
```

Listing 6.4: Python script for network traffic sniffing with Scapy.

Bibliography

- [1] Jerry Gamblin. Mirai BotNet - <https://github.com/jgamblin/Mirai-Source-Code>.
- [2] Kevin Ashton et al. That ‘internet of things’ thing. *RFID journal*, 22(7):97–114, 2009.
- [3] Lu Tan and Neng Wang. Future internet: The internet of things. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 5, pages V5–376. IEEE, 2010.
- [4] Feng Xia, Laurence T Yang, Lizhe Wang, and Alexey Vinel. Internet of things. *International journal of communication systems*, 25(9):1101, 2012.
- [5] Charith Perera, Chi Harold Liu, Srimal Jayawardena, and Min Chen. A survey on internet of things from industrial market perspective. *IEEE Access*, 2:1660–1679, 2014.
- [6] Awais Ahmad, Murad Khan, Anand Paul, Sadia Din, M Mazhar Rathore, Gwanggil Jeon, and Gyu Sang Choi. Toward modeling and optimization of features selection in big data based social internet of things. *Future Generation Computer Systems*, 82:715–726, 2018.
- [7] Anna Kobusińska, Carson Leung, Ching-Hsien Hsu, S Raghavendra, and Victor Chang. Emerging trends, issues and challenges in internet of things, big data and cloud computing, 2018.
- [8] Ahmed Banafa. Iot standardization and implementation challenges. *IEEE internet of things newsletter*, pages 1–10, 2016.

-
- [9] T. Berners-Lee, MIT/LCS, R. Fielding, and UC Irvine. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, RFC Editor, May 1996.
- [10] R. Fielding, UC Irvine, J. Gettys, J. Mogul Compaq/W3C, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.
- [11] M. Belshe, BitGo, R. Peon, Google Inc, M. Thomsom Ed., and Mozilla. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, RFC Editor, May 2015.
- [12] Larry Masinter, Tim Berners-Lee, and Roy T Fielding. Uniform resource identifier (uri): Generic syntax. *Network Working Group: Fremont, CA, USA*, 2005.
- [13] Z. Shelby, ARM, K. Hartk, C. Bormann, and Universitaet Bremen TZI. The Constrained Application Protocol (CoAP). RFC 7252, RFC Editor, June 2014.
- [14] Message queuing telemetry transport (mqtt). Standard, International Organization for Standardization, Geneva, CH, June 2016.
- [15] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing*, 3(1):11–17, 2015.
- [16] CISCO P. Saint-Andre. xtensible Messaging and Presence Protocol (XMPP): Core. RFC 6120, RFC Editor, March 2011.
- [17] MultiMedia LLC. DDS foundation data distribution service.
- [18] Paolo Bellavista, Antonio Corradi, Luca Foschini, and Alessandro Pernaflini. Data distribution service (dds): A performance comparison of opensplice and rti implementations. In *2013 IEEE symposium on computers and communications (ISCC)*, pages 000377–000383. IEEE, 2013.
- [19] OASIS. Advanced message queuing protocol (amqp) version 1.0, 2012.
- [20] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis

-
- Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.
- [21] Tom Spring, K Carpenter, and M Mimoso. Bashlite family of malware infects 1 million iot devices. *Threat Post*, 2016.
- [22] Sam Edwards and Ioannis Profetis. Hajime: Analysis of a decentralized internet worm for iot devices. *Rapidity Networks*, 16, 2016.
- [23] Zaied Shouran, Ahmad Ashari, and Tri Priyambodo. Internet of things (iot) of smart home: privacy and security. *International Journal of Computer Applications*, 182(39):3–8, 2019.
- [24] Amy B Wang. ‘i’m in your baby’s room’: A hacker took over a baby monitor and broadcast threats, parents say. *The Washington Post*, Dec 2018.
- [25] Iyatiti Mokube and Michele Adams. Honey pots: concepts, approaches, and challenges. In *Proceedings of the 45th annual southeast regional conference*, pages 321–326, 2007.
- [26] Abhishek Mairh, Debabrat Barik, Kanchan Verma, and Debasish Jena. Honey pot in network security: a survey. In *Proceedings of the 2011 international conference on communication, computing & security*, pages 600–605, 2011.
- [27] Niels Provos. Honeyd virtual honeypot - www.honeyd.org.
- [28] Alejandro Guerra Manzanares. Honeyio4: the construction of a virtual, low-interaction iot honeypot. B.S. thesis, Universitat Politècnica de Catalunya, 2017.
- [29] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: A novel honeypot for revealing current iot threats. *Journal of Information Processing*, 24(3):522–533, 2016.
- [30] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices. *Black Hat*, 2017.

-
- [31] The collaboration platform for api development - www.postman.com.
- [32] Aldo Cortesi, Maximilian Hils, and Thomas Kriechbaumer. contributors: mitmproxy: A free and open source interactive https proxy (2010–). URL <https://mitmproxy.org>. [Version 3].
- [33] Mitm proxy - free and open source interactive https proxy. - <https://mitmproxy.org/>.
- [34] Sabeel Ansari, SG Rajeev, and HS Chandrashekar. Packet sniffing: a brief introduction. *IEEE potentials*, 21(5):17–19, 2003.
- [35] Pallavi Asrodia and Hemlata Patel. Analysis of various packet sniffing tools for network monitoring and analysis. *International Journal of Electrical, Electronics and Computer Engineering*, 1(1):55–58, 2012.
- [36] James Forshaw. *Attacking network protocols: a hacker's guide to capture, analysis, and exploitation*. No Starch Press, 2017.
- [37] libpcap, a portable c/c++ library for network traffic capture. - <https://www.tcpdump.org/>.
- [38] Arimon-ng, monitor mode on wireless interfaces. - <https://www.aircrack-ng.org/doku.php?id=airmon-ng>.
- [39] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.
- [40] Marco De Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O de Vivo. A review of port scanning techniques. *ACM SIGCOMM Computer Communication Review*, 29(2):41–48, 1999.
- [41] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proc. of the Internet Measurement Conference (IMC)*, 2019.
- [42] Rick Cattell. Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

-
- [43] MongoDB, a general purpose, document-based, distributed database built for modern application developers and for the cloud era.- <https://www.mongodb.com/it>.
- [44] Alexandru Boicea, Florin Radulescu, and Laura Ioana Agapin. MongoDB vs oracle–database comparison. In *2012 third international conference on emerging intelligent data and web technologies*, pages 330–335. IEEE, 2012.
- [45] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of iot devices. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, pages 41–50, 2018.
- [46] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Iotsense: Behavioral fingerprinting of iot devices. *arXiv preprint arXiv:1804.03852*, 2018.
- [47] Bruhadeshwar Bezawada, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of internet-of-things devices. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1337, 2019.