Tactical Solutions for Mobile Device Forensics in Military Operations

Strategies for Extracting Data in Any Situation

Leclef Thomas

Research and Development project owner:
Cyber Command

Master thesis submitted under the supervision of
Debatty Thibault

the co-supervision of
Croix Alexandre

in order to be awarded the Degree of
Master in Cybersecurity
Cryptanalysis and Forensics

Academic year
2023 – 2024

I hereby confirm that this thesis was written independently by myself without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation.

The author transfers to the project owner any and all rights to this master dissertation, code and all contribution to the project without any limitation in time nor space.

03/06/2024

Title: Tactical Solutions for Mobile Device Forensics in Military Operations

Author: Leclef Thomas
Master in Cybersecurity – Focus Cryptanalysis and Forensics
Academic year: 2023 – 2024

Abstract

During military operations, the need to extract critical intelligence from mobile devices is of paramount importance. The use of this information can be made to determine if an allied soldier's phone has been infected with malware or to extract the maximum amount of information from an enemy soldier's phone. However, traditional forensics techniques are not appropriate for the dynamic and resource-constrained environment of military field. It is therefore necessary to develop an innovative solution to meet the unique challenges of the military environment.

The aim of this master thesis is to provide a digital investigative solution for teams in the field. To achieve this, three main objectives are identified: rapid isolation of mobile devices, portable intrusion detection system and information discovery from a locked phone.

In order to meet these objectives, it is first necessary to have a research method. This method begins with a state of the art that provides a comprehensive summary of mobile device security. Then, in order to meet the various objectives of the project, the idea is to use the content of the state of the art as a basis. In this way, it is possible to combine several existing and widely used tools to meet our objectives. In order to assess the effectiveness of this solution, simulated usage scenarios will be studied in Chapter 5 of this thesis. These scenarios are designed to simulate real situations in the field.

The results of the different scenarios are discussed in Chapter 5. These findings highlight that determining whether a mobile device is free from malware is challenging if the malware only modifies the device locally without engaging with the network. In addition, some very advanced malware are very difficult to detect. However, this master thesis manages to detect most fairly basic malware despite the constraints of the military environment. As far as extracting information from locked phones is concerned, probe requests can be captured easily.

In short, the vast majority of malware can be detected by our solution. As always, a category of very advanced malware cannot always be detected and will require a more in-depth analysis. Despite this, by providing a significant advantage to teams in the field, the result of this thesis is not just a piece of research but a product in its own right.

Keywords: Military digital forensics, secure isolation, portable intrusion detection.

Preface

"Smartphones are not just gadgets. They have become an extension of our being, a key to the world we live in." - Jerry Yang, co-founder of Yahoo!

In our modern age, there's no denying that mobile phones have become an omnipresent extension of our being. They have become much more than simple communication tools; they have become constant companions, windows onto the virtual world that coexist closely with our physical reality. Indeed, these devices are constantly with us, to the point where they almost become an additional member of our person, an extension of our identity and existence. This fusion between the virtual and real worlds has created a new ecosystem in which our physical possessions are extended into cyberspace, often without our minds being fully aware of it.

In this context, the protection of our physical assets, now extended to the virtual world, is of paramount importance. Just as the State has a responsibility to protect the physical integrity of its citizens, it is imperative to guarantee the security of their digital possessions, which have become essential extensions of their identity and their daily lives.

It is with this in mind that this master thesis is written. By focusing on improving protection against mobile malware in military environment, it aims to address an urgent need for security in a world where our lives are increasingly intertwined with technology. By understanding and combating the threats that target our mobile devices, we can hope to safeguard not only our physical assets, but also our autonomy, privacy and security in this ever-changing digital world.

This work is not only an academic contribution, but also a contribution to the preservation of personal integrity and freedom in an increasingly connected world. By exploring the challenges and solutions associated with mobile phone security, we are working to strengthen the foundations of a safe and equitable digital society for all.

Acknowledgements

I would also like to thank Cyber Command who proposed the project to RMA. I also wanted to thank my roommates who kindly connected their phones to my access point to build a whitelist. Thanks also to Mr Croix for being my academic supervisor throughout this work. I hope that this work will be used for useful, ethical and practical purposes.

Table of Contents

List of Figures

List of Abbreviations

ADB	Android Debug Bridge
AP	Access Point
API	Application Programming Interface
APT	Advanced Persistent Threat
BIOS	Basic Input/Output System
BYOD	Bring Your Own Device
C2	Command & Control
C&C	Command & Control
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
EAP	Extensible Authentication Protocol
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
HIDS	Host-based Intrusion Detection System
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IMSI-catcher	International Mobile Subscriber Identity-catcher
IoC	Indicator of Compromise
IP	Internet Protocol
IPSec	Internet Protocol Security
IRC	Internet Relay Chat
LED	Light-Emitting Diode
MAC	Media Access Control
MD5	Message Digest Algorithm 5
NFC	Near Field Contact
NIDS	Network-based Intrusion Detection System
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OS	Operating System
OSI	Open Systems Interconnection
PII	Personally Identifiable Information
PIN	Personal Identification Number
RSA	Rivest–Shamir–Adleman
SHA-1	Secure Hash Algorithm 1
SIEM	Security Information and Event Management
SIM	Subscriber Identification Module
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SSID	Service Set IDentifier
TCP	Transmission Control Protocol

TLS	Transport Layer Security
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtual Private Network
WPA	Wi-Fi Protected Access

Chapter 1
Introduction

1.1 Motivations

The ever-expanding battlefield necessitates efficient and secure methods for extracting critical intelligence from mobile devices. However, classical forensics techniques are impractical for deployment in the dynamic and resource-constrained realities of the military field. This lack of readily deployable solutions leaves digital forensic teams without essential tools to handle situations. Two of these situations stand out.

The first situation is when a soldier suspects that his phone has been infected by malware as a result of suspicious behaviour. In this case, the digital forensic team must isolate the mobile device as quickly as possible and determine whether it is really infected.

The second possible situation is that the mobile device is seized in a sensitive environment such as from an enemy combatant or a captured enemy installation. In this case, the priority is to gather as much information as possible from the phone to fulfil the main intelligence mission. It is important to note that in this case too, the mobile device must be placed in an isolated environment so as not to reveal any information about the soldiers deployed.

At present, there is no effective solution that would enable a digital forensic team in the field to handle correctly these two situations. Indeed, in the military field, there is no isolated physical or network infrastructure into which the mobile device could be placed for investigation. Therefore, there is currently a need to develop a solution that will enable a digital investigation team to work in an isolated environment that can be rapidly deployed and easily transported.

In short, we can take into account the 2 situations in which a digital forensic team may find itself. On the other hand, we can also take into account the fact that this team needs to work in an isolated environment. From this, it is possible to define 3 main objectives that this project will endeavour to meet.

1. Implementing a mechanism for isolating a mobile device from any external communication that is both quick to deploy and easy to transport.

2. Developing a portable intrusion detection system that can be easily deployed anywhere on a military site. This situation arises when a soldier suspects that his phone is infected. In this case, it is possible to unlock the phone because we have its PIN code.

3. Establishing a system that allows to retrieve information from a phone that we do not have the PIN code. This situation arises when a phone has been seized.

1.2 Project statement & contributions

The aim of this project is to fulfil the three above-mentioned objectives in relation to the military context. To do this, we are developing 3 main solutions that will further be put together in order to build a concrete project.

1. All mobile devices that need to be analysed by the digital forensics team will be placed in a Faraday enclosure to prevent any external communication.

2. An intrusion detection system attached to this Faraday enclosure will enable malware to be detected. This system will enable a digital forensic team to identify more easily and quickly whether a mobile device is infected.

3. Some information sent by a locked mobile device within the Faraday enclosure will be collected.

Firstly, given that the use of a Faraday enclosure is not commonplace, we give here 4 main reasons for choosing this innovation.

1. **Protection of Classified Information**: Military operations often involve handling classified or sensitive information. A Faraday enclosure provides a secure environment for forensic analysts to examine the devices, minimizing the risk of data leaks or unauthorized access to classified materials.

2. **Isolation from Remote Threats**: Placing the seized devices inside a Faraday enclosure ensures that they are shielded from external electromagnetic signals, preventing any attempts by adversaries to remotely wipe data or initiate self-destruct mechanisms.

3. **Chain of Custody**: Military legal proceedings require a meticulous chain of custody for digital evidence. A Faraday enclosure helps maintain the integrity of the evidence by ensuring that the devices are protected from external influences during transportation and analysis.

4. **Operational Security**: In military operations, maintaining operational security is crucial. Using a Faraday enclosure helps in preventing unintentional signals or emissions from the devices that could potentially reveal the location or activities of the digital forensics team.

Then, the second solution attached to this project is to develop an intrusion detection system attached to this Faraday enclosure. This system will enable a digital forensic team to identify more easily and quickly whether a mobile device is infected. To do this, we're going to use a Rapsberry Pi that we'll install inside the box. Using this Raspberry Pi, we will create a Wi-Fi access point to which we will connect the phone. This Wi-Fi access point will then be used to analyse the requests sent by the phone to the Internet. However, it is really important to remember that our context requires us to work in a totally isolated environment so as not to reveal any sensitive information. Therefore, all the responses sent by the access point will be the result of an Internet services simulator.

Finally, in the case of the third solution for which the phone cannot be unlocked, we will still try to recover interesting artefacts from the mobile device we have seized. For

example, probe requests are among the interesting artefacts that can be recovered without needing the phone's PIN. A probe request is a frame sent by a Wi-Fi-enabled device seeking available network access points to connect to [81]. In most cases, we can find the SSIDs that the mobile device has already been connected to. In short, even if the mobile device cannot be unlocked, probe requests can be recovered to find out more about the movements of the person to whom it belonged, using tools such as WiGLE[1].

1.3 Organisation of this document

In order to achieve these objectives, it was essential to introduce the subject and the context in this first chapter. Chapter 2 will review the current literature on mobile device security, current detection methods, and data acquisition from locked phones. This literature review will identify all the existing tools in the field to understand their usefulness, advantages, and areas of application. Once these various tools have been identified, the aim of Chapter 3 will be to select the relevant tools for the given context. In addition, this chapter will enable the choice of approach to be validated through the use of recognised frameworks. In short, this chapter will provide the link between the research and the implementation in the following chapter. The aim of Chapter 4 will be to explain how to use the product and how it has been implemented. This will provide users with an explanation of how to use the product and will also serve as a basis for developers wishing to continue the project. Additionally, Chapter 5 will highlight the use and effectiveness of the product in relation to recent threats. It will also highlight Snort's limitations in detecting mobile malware. Furthermore, Chapter 6 outlines possible improvements in terms of research in relation to this work. Finally, Chapter 7 presents a conclusion to this work, highlighting the research that has been carried out in this work, the results obtained and the innovation in the given context.

[1] https://www.wigle.net/

Chapter 2
State of the art

First of all, before getting to the heart of the matter and explaining how to perform network-based malware detection for mobile devices, it is important to review the scientific literature. This is because it is vital to know why mobile device security is so important, but also what solutions have already been explored by scientific research.

This chapter is therefore divided into several sections. The section 2.1 begins by reviewing the basics of smartphone security. It explains the importance of smartphones, their vulnerabilities, the capabilities of malware, and how it exfiltrates data. Once mobile device security has been reviewed, the section 2.2 looks at mobile malware detection. To do this, the section first looks at the tools that exist today to detect malware in general. It then highlights the current state of research by exploring the various scientific methods developed to detect malware specifically for mobile devices. Moreover, it looks at the various tools available to simulate an Internet network. Finally, the section 2.3 presents the different possibilities for extracting artifacts from locked phones.

2.1 Mobile security

Today, the adoption of smartphones by individuals continues to grow. More and more businesses and organisations are operating on the BYOD or Bring Your Own Device model. Most of the time, the employees of these companies carry their mobile device with them when they are within the organisation and even as soon as they leave it. These mobile devices can even be used to access the organisation's data or systems. Unfortunately, these mobile devices are not as secure as traditional computers. Obviously, the user's aim is not to think about the security of their smartphone by installing anti-virus software or other solutions as they might for a computer. All in all, the growing use of smartphones and the lack of security vigilance on the part of users make them a perfect target for malware writers [41]. This same scenario of the increasing use of smartphones and the lack of security vigilance on the part of users can also be transposed to the military world. Soldiers can bring their own mobile devices during their missions. Therefore, the compromise of mobile devices could reveal the location of troops, images or videos of situations on the ground, operational plans, or even confidential communications.

The need to develop effective malware detection techniques is therefore evident. Of course, before building this detection system, it is vital to understand the state of scientific research into mobile security. To begin, it's crucial to establish a taxonomy of the different types of mobile malware that exist. This provides a foundational understanding before delving into the loopholes through which they may infiltrate. Then, when the phone is infected with malware, it is interesting to see what the malware can do, what actions it can take. Finally, when considering the potential threat of malware attempting to extract private information, it becomes imperative to categorize the various techniques used to exfiltrate data. This approach will facilitate the identification of detectable data exfiltra-

tion techniques, those that can be prevented, and those that fall beyond the detection capabilities explored in this study.

2.1.1 Mobile malware taxonomy

To fully understand what mobile malware means, it is important to define the different categories of malware that can be found, as well as their behaviour. Not all malware have the same goals or the same behaviour. It is therefore necessary to understand the different possible behaviours in order to know what we might be dealing with and therefore possibly detect it more effectively.

1. **Virus**: Virus is a type of malware with the ability to replicate itself and spread to other programs or files on the system. As a result, when the user launches an infected program, he or she will launch the virus without the user's knowledge. Most of the time, viruses use Command Control channels to communicate with a remote server. Through these servers, attackers can exfiltrate information from the system or send commands to control the virus remotely. This type of malware arrived very early on mobile devices, so much so that we find traces of it in 2004 with the Dust virus, which was a proof of concept seeking to demonstrate that it was possible to create a virus for smartphones by propagating itself in various files on the device [62].

2. **Rootkit**: When a rootkit is installed on the user's mobile device, the attacker can gain remote access to the phone to control it. The difficulty with rootkits is that they use obfuscation to hide their presence inside the system, making them harder to detect [58].

3. **Backdoor**: This type of malware serves as a means to bypass authentication or other security measures, providing unauthorized access to computer systems or the data stored within. They may manifest at the system level, within cryptographic algorithms, or embedded within applications [35].

4. **Spyware**: The purpose of spyware is to monitor what the user is doing on their smartphone. It can therefore watch what is displayed to the user or retrieve the keystrokes it records . For example, Flexispy[1] is an application that can be installed and hidden to retrieve text messages, emails, photos, GPS location, etc. This type of malware is quite vicious. Indeed, developers of spyware try to commodify and market their products to a general audience. While the consumers of spyware are basically governments and law enforcement [66]. Currently, spyware is therefore sold as security products, especially aimed at businesses, parents and intimate partners. The article [53] highlights the contrast between the social significance attributed by suppliers and the intrusive and potentially non-consensual aspects of these products, as well as the lack of recourse for people monitored without their consent. So spyware can therefore be considered as malware and not as software to increase security as the developers try to claim.

5. **Worm**: This is a type of malware that is capable that can autonomously spread from one computer to another without needing human intervention once it gains access to a system. Usually, worms propagate through a network via internet connections local

[1] https://www.flexispy.com/

area networks. The difference between viruses and worms is that viruses require human activation and rely on host systems for replication. Whereas worms can propagate autonomously without human intervention or the need for a host system [6].

6. **Trojan horse**: This type of malware takes its name from the famous Greek myth of the Trojan horse. A Trojan is any application that masquerades as a legitimate application when it is in fact malware. Like other malware, it can therefore exfiltrate the user's private data and interact with the files on the mobile device [5].

7. **Adware**: The purpose of adware is simply to display advertisements to a user. Some adware also takes advantage of this to steal private information from the user [1].

2.1.2 Mobile malware entry points

Despite the fact that there are many different types of malware, as we have just described, it is interesting to go back to basics and look at the weaknesses and gaps in the mobile devices that can be used by an attacker to infect a device. When dealing with an attacker with sophisticated tools and a complex strategy, one can identify a multi-stage chain of events called the *Cyber Kill Chain*. Looking at this set of steps, this subsection describes the third step which is delivery stage [89]. Indeed, the goal is to analyse the various possible delivery methods that an attacker can use when conducting an attack.

This point may seem a bit long in a work dedicated to malware detection. Nevertheless, this master thesis is part of the digital forensics discipline. Therefore, if we look at the different steps of the incident response process as defined by the SANS institute [61], it is important to consider all the points. This process includes preparation, eradication and lessons learned. This subsection makes it possible to provide significant support to digital forensics analysts for these three key elements. Indeed, this work is not only intended to improve detection techniques but also to be a stepping-stone of knowledge in order to follow the incident response process.

Most of the time, entry points are due to human error allowing malware to enter a system. However, here we try to focus on loopholes that are particularly present on mobile devices. In view of the fact that there is no paper clearly highlighting a taxonomy of all entry points specific to mobile malware. The taxonomy of the resources here comes from well-established frameworks such as the MITRE ATT&CK framework or reliable websites relating to computer security topics. To understand the different entry points, the defined categories contain several elements. This allows the information to be sorted correctly. We can therefore define 6 categories on the Figure 2.1. Nevertheless, we detail each element within the different categories to give a complete explanation of each entry point.

Malicious applications

It is possible that the user downloads applications from his own interaction. There are two possible infection vectors in this category.

- **Official app stores**: These include platforms like Google Play Store for Android and the Apple App Store for iOS. Despite efforts to maintain security, malware can sometimes bypass the cleaning process and end up on these stores [48, 84, 85].

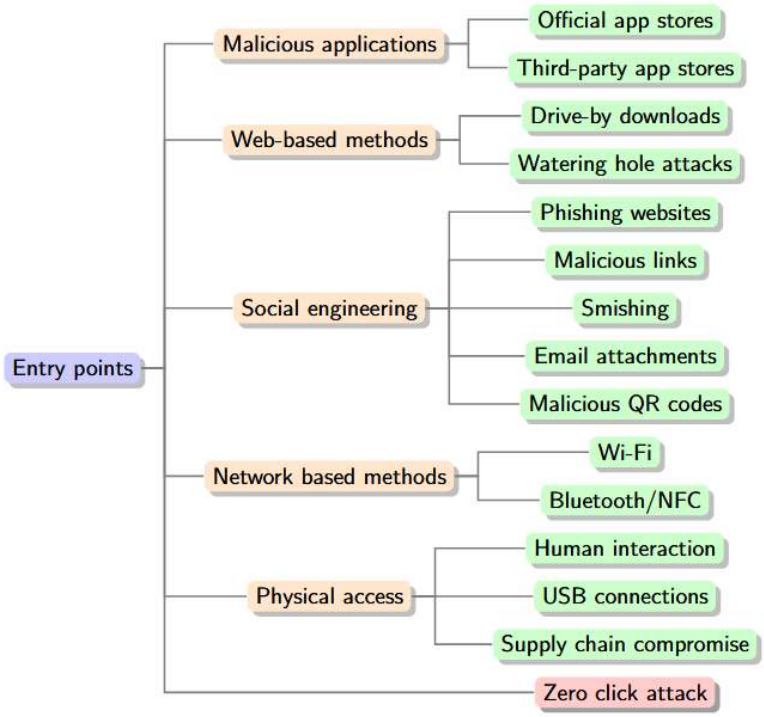


Figure 2.1: Mobile malware entry points taxonomy

- **Third-party app stores**: These are alternative app marketplaces not affiliated with the official app stores of mobile platforms can be installed by users. These stores often have less strict security measures, making them more susceptible to hosting malicious apps [18].

Web-based method

- **Drive-by Download**: A drive-by-download attack is when the user downloads malicious code unknowingly or unintentionally. Such an attack takes advantage of a flaw in the web browser, application or operating system. The user still has to interact with a website or application to be infected [9]. This is where the attack differs from a zero-click attack.

- **Watering hole attacks**: This is a type of targeted attack where an attacker will hack a site where the victim often goes. Once the site is hacked, the attacker will upload malicious code that will be used to hack the victim [22].

Network-based methods

1. **Wi-Fi**: Mobile devices frequently connect to various networks, including public Wi-Fi, which can be insecure. Attackers may set up rogue Wi-Fi hotspots to intercept traffic and launch man-in-the-middle attacks [65].

2. **Bluetooth/NFC**: Malware spread through Bluetooth or Near Field Communication (NFC) connections between devices, often exploiting insecure device settings or protocols [52, 62].

Social engineering

1. **Phishing websites**: An attacker attempts to extract sensitive information or has the victim install malware by manipulating it. This can, for example, involve fake login pages for banking applications [8]. There are different ways of reaching these phishing websites, which we describe here:

   (a) **Malicous links**: Links shared via email, text messages, or social media platforms that lead to websites hosting malware or initiating downloads.

   (b) **Smishing**: The word simply derives from a combination of the words phishing and Short Message Service (SMS). It simply means sending a phishing attack by SMS. To deceive the user by SMS into doing something with malicious intent. Obviously, this loophole affects mobile devices more than traditional computers [91].

   (c) **Email attachments**: Attackers often send emails with malicious attachments disguised as legitimate files. When these attachments are opened on a mobile device, they can execute malware that compromises the device.

   (d) **Malicious QR Codes**: There may be QR codes that link to malicious sites asking the user to download an application. Combined with a repackaging technique, it's easy to get users to install malware by making them believe they are installing a real application [60].

Physical access

1. **Human interaction**: Attackers can physically manipulate or deceive users into installing malware on their devices through techniques like social engineering, exploiting trust or user's lack of vigilance.

2. **USB connections**: Malware can be spread through USB connections when malicious devices are connected to malicious computers or cables[2], transferring the malware to the connected mobile device.

3. **Supply chain compromise**: In the case of highly advanced attackers, opponents can modify product distribution mechanisms or the product itself. This is done prior to the receipt made by the end user in order to compromise the system [17].

Zero click attack

This type of loophole is a sophisticated attack that allows malware to be installed on a device without any action from the user. In order to do so, they exploit zero-day vulnerabilities in software or protocols. These attacks are highly targeted and can have great consequences, often leaving minimal traces, making them difficult to detect and defend against [10]. One popularised example of such malware is Pegasus. Pegasus is spyware developed by the Israeli cyber intelligence firm NSO Group, capable of infiltrating smartphones to access messages, emails, microphones, cameras, and other sensitive data. It has been controversially used by governments to monitor journalists, activists, and political people [67, 76].

[2] https://shop.hak5.org/products/omg-cable

2.1.3 Mobile malware capabilities

Just like on a personal computer, once the malware is installed, it can perform a whole range of operations. For an overall understanding, it is useful to draw up the broad categories of these different operations.

- **Data theft**: Classically, malware can try to steal data present on the user's mobile device. For example, the RedDrop malware was able to steal contacts, photos, Wi-Fi information and even record audio. A rather interesting artefact, which is harder to obtain than on a computer, is the phone's location, making it possible to track an individual [4, 16] . This category of malware therefore poses many problems for the privacy of its victims.
- **Disruption of service**: The attacker can also block use of the phone by making all resources inaccessible to the user [42].
- **Incurring costs**: It is also possible for the malicious person to incur costs for the user and thus extract money from them. The malware can therefore send text messages or make calls to premium rate numbers without the user's knowledge [54].
- **Botnet**: An attacker can also use the phone as a bot. In this way, the mobile device becomes an integral part of a botnet that can be used to carry out distributed denial of service attacks on target computer systems. In order to control the bot, the attacker needs to set up a Command & Control channel where he can send all sorts of instructions [88].

In short, malware can do an enormous amount of things once it has been installed on the victim's phone. One thing to note is that in the majority of cases, network traffic is used to establish and communicate with a Command & Control (C&C) channel. This channel can be used to exfiltrate data or receive commands from the attacker. We can therefore already understand that when a phone is infected, the traffic generated will be very interesting to analyse in order to detect the malware. However, it would be inappropriate to limit ourselves to the mere fact that we know that there is network traffic that could be indicative of malicious activity. It is also important to understand the different techniques used by malware writers to exfiltrate sensitive data.

Definition

Command & Control server A C2 or C&C server is a server that the attacker owns and to which the malware-infected machine can connect. The malware may try to communicate with the server to exfiltrate data, receive commands from the attacker, or launch DDoS attacks. In order to connect to these servers, malware has domain names or IP addresses registered within their code. To avoid detection, the malware mainly communicates using commonly used protocols such as HTTPS. In this way, it is difficult to differentiate malicious traffic from that of a legitimate user [13, 34, 90].

2.1.4 Mobile malware data exfiltration techniques

This Subsection therefore presents the various data exfiltration techniques. Formally speaking, data exfiltration is the movement, theft or deletion of any data from a device. Most of

the time, this data is private to a person or a company and therefore represents a significant market value for an attacker. This data exfiltration can lead to huge financial costs when it comes to a company or a risk to national security when the data comes from government institutions.

Future work

Data exfiltration techniques are very similar for both smartphones and laptops. There are therefore few papers reporting on data exfiltration specific to mobile devices. However, the work of [38] presenting the possibility of exfiltration by SMS and inaudible audio transmission for Android phones and [39] demonstrating the potential for data exfiltration through the pairing function of iOS devices makes an important contribution. This highlights that there are different means of exfiltration for mobile devices compared with computers. Nevertheless, the Table 1 of the article [38] presents a taxonomy of the different data exfiltration techniques from Android phones. On the whole, very few scientific articles present a taxonomy of these different techniques. This lack of scientific analysis is easily explained by the fact that data exfiltration remains similar. Nevertheless, malware writers could potentially take advantage of the fact that defenders assume that data exfiltration is similar to a conventional computer. In short, there may well be other avenues to explore regarding certain data exfiltration techniques that would be possible on smartphones and not on conventional computers.

At first glance, to find data exfiltration techniques possible on smartphones and not viable on computers, it could be interesting to compare the different functionalities. For example, a smartphone has a vibrate function that is not present on a computer. The vibrator could therefore be used to exfiltrate sensitive data. This is just one example of the many possibilities that could be explored to discover data exfiltration from mobile devices.

In any case, the same data exfiltration techniques classically used on computers can also be found on mobile devices. So it's worth exploring the different categories. To draw up a correct taxonomy of what is possible, it is interesting to look at what is described in the MITRE ATT&CK framework describing the different exfiltration methods that have already been used by malware in the past [3]. The choice to rely on this framework to make a taxonomy is not insignificant. Indeed, this framework is developed and maintained by the MITRE corporation, a non-profit organisation that focuses on research in cybersecurity. It is widely recognised by many professionals in the field as shown by different use cases [20, 31, 45]. Moreover, it also provides a specific matrix for mobile malware [15]. In addition to the framework, it's worth looking at the various tutorials available on the internet to see what methods attackers could learn [2, 57].

As a result, it is thanks to these different sources that we can draw up a taxonomy of existing techniques. This taxonomy is illustrated in Figure 2.2, but the different categories are also explained below. This figure will then be used in the Section 3.3.2 to identify how the different means of data exfiltration are covered by this project.

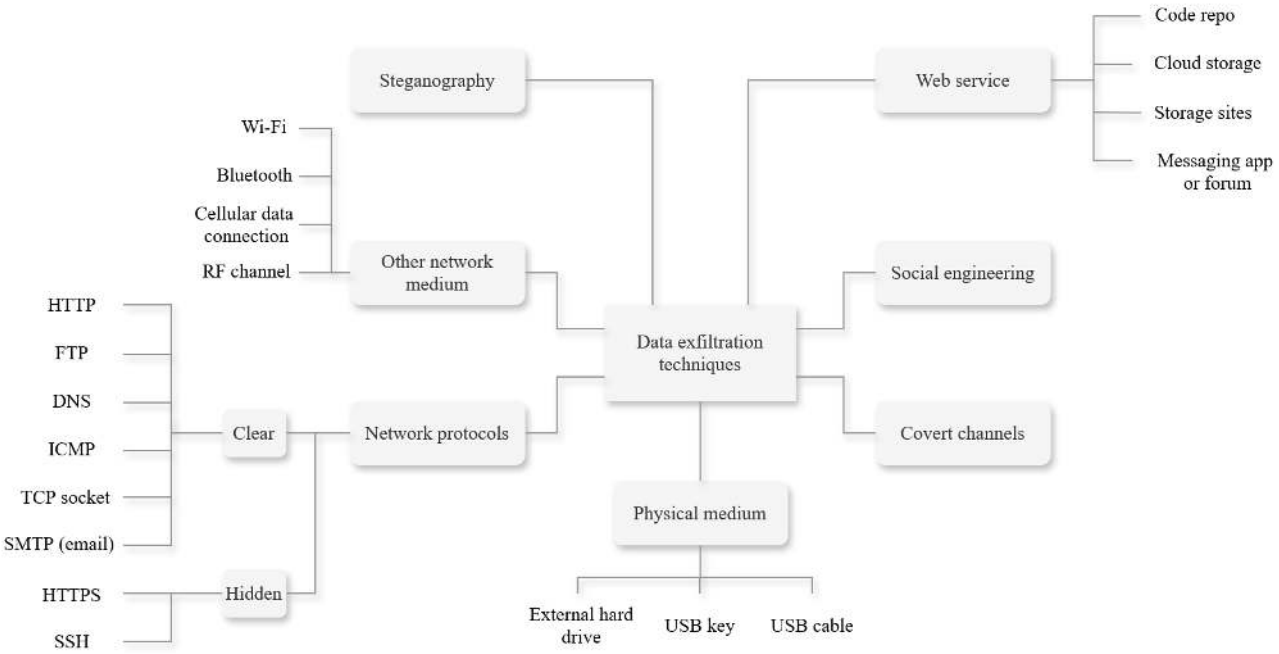


Figure 2.2: Data exfiltration taxonomy

Web service

The very first category highlighted here is the use of web services. It is important to note that malware writers want to avoid detection when stealing data. The best way to do this is to mimic traffic that could be generated by the user. By using common web services, the malware can exfiltrate data without it looking suspicious. For example, we could imagine malware using the discord application to send sensitive data it has recovered to a target machine. In this way, if the user uses discord every day, the traffic will appear legitimate and go undetected by defence systems. A blog post [23] written by Utkhedkar A. shows how this exfiltration can be implemented in practice.

Social engineering

Secondly, another technique that we introduced earlier when talking about loopholes is social engineering. A slightly naive user could give out confidential information by telephone or e-mail if the attacker manipulates him well enough. So this is really a case of data exfiltration using humans as a vulnerability [55].

Covert channels

Covert channels can also be used to recover sensitive data. A covert channel refers to a form of attack in which a capability is established to clandestinely transfer information objects between processes that, according to IT security policy, should not be allowed to communicate. An example of a covert channel attack on Android was studied in the article [38], which highlights the possibility of using an inaudible audio transmission to exfiltrate data.

Physical medium

Another technique that may be more obvious or even simpler is to use a physical medium. If we have full physical access to the mobile device, we can simply plug in a USB stick and download the sensitive information.

Network protocols

Next, the most widely used and simplest technique for data exfiltration is to use classic protocols such as HTTP, TCP sockets, and so on. This category can be linked to the use of web services when the attacker does not implement the service himself. For example, the HTTP protocol can be used as a web service if the exfiltrated data is sent directly to a forum [14]. As things stand, we can assume that malware will use encrypted protocols to exfiltrate data. However, it would be pointless to give up on analysing unencrypted traffic. We therefore need a hybrid approach in which we analyse both encrypted and unencrypted traffic if we want to detect malware on the network.

Other network medium

In the same way, other network media can be used to exfiltrate data. These can include Bluetooth, radio frequency and cellular connections. An example of data exfiltration methods by frequency bands is GSMem. The paper presents how it is possible to build a system exfiltrating data on different frequency bands such as GSM, UMTS or LTE [51]. They obviously require more creativity on the part of attackers and are therefore less widespread.

Steganography

Finally, the last category we are highlighting is steganography. Steganography involves concealing information within another message or object in a way that makes the presence of the hidden information imperceptible to human observation. It is used by malware to extract data discreetly by hiding it in an image, for example. This category can actually be included in all the other categories in the diagram. For example, we could post an image on a forum containing confidential information. In this case, we use a web service with steganography [56].

2.2 Malware detection

By way of context, this is a case where no matter what defence systems have been put in place, the mobile device has been infected and the attacker has managed to get into the system. Once we start from that premise, we have to prepare for the inevitable. In the world of cyber security, when we try to prepare for the inevitable, we try to build systems that are cyber resilient. The aim of a so-called cyber resilient system is to keep essential services running despite the fact that the system is under attack. But that's not exactly the context in which we find ourselves. Our aim here is twofold:

1. Limit information leakage
2. Identify whether the system is infected with malware

To achieve the first objective, we are using a Faraday enclosure to isolate the mobile device from any means of data exfiltration that we have identified. With regard to the second

objective, we can use a pillar of cyber resilience. This pillar on which we will focus in order to achieve this objective is the detection pillar.

Definition

Cyber resilient system A system is said to be cyber resilient if it is built in such a way that it can continue to provide essential services despite the presence of incidents. To achieve cyber resilience in a system, there are 5 pillars: identify, protect, detect, respond and recover [87]. As a reminder, our context means that we are focusing on the detection pillar that we define here.

Detection pillar Given that we know we're going to have to deal with threats, the idea is to be able to identify them so that we can respond to them and recover from any damage they may have caused to our assets. To do this, there is a whole range of tools available, which we cover in the following Section 2.2.

2.2.1 Malware detection techniques

In general, in a computer system, when we need to determine whether a machine is infected with malware, it is possible to implement several techniques. These techniques therefore include logging, intrusion detection systems, honeypots, SIEMs and threat hunting [87]. In addition to malware detection, malware analysis can also be considered even though it does not have the same objective. Indeed, malware analysis aims to understand the behaviour of the malware quite accurately in order to document and eradicate the threat. The will is really to understand how the malware works. One cannot therefore overlook the fact that malware analysis contributes significantly to its detection.

The concepts of malware detection and analysis are therefore not dissociable and can significantly help the work of a digital forensic analyst. Whether in malware detection or malware analysis, these two disciplines have a common goal: the search for indicators of compromise (IoCs) to prove the presence of an intrusion. In short, in addition to conventional detection techniques, dynamic and static malware analysis techniques must be considered in malware detection. This subsection describes all techniques of both disciplines in detail. Both in terms of pure detection and malware analysis. Later in this work, some techniques will be chosen according to the imposed context.

Definition

Indicator of Compromise IoC is an artifact that can be found in a system or network that indicates the intrusion of a threat. Typical IoCs are signatures, malware files, URLs, domain names or IP addresses. Once discovered, they can be integrated into IDS to prevent some attacks from happening again [19].

Intrusion Detection System

Today's organisations are increasingly equipping themselves with different IT components to manage their assets and employees. Absolute security does not exist, given that it is impossible to close all security loopholes, so system auditing is seen as the last line of defence. To this end, as mentioned in the article [64], intrusion detection systems (IDS) have been in place since the early 1990s by the IT security community.

Definition

Intrusion Detection System (IDS) It is a security mechanism designed to identify and respond to suspicious activities occurring within a monitored environment. Based on this monitored environment, there is two mains categories of IDS: NIDS and HIDS.

Network-based Intrusion Detection System (NIDS) In this case, the IDS is a separate device that is connected to a network in such a way that it sees all of the traffic passing by at that location of the network.

Host-based Intrusion Detection System (HIDS): In this case, the IDS consists of an agent that is installed on a specific host and is monitoring everything that is happening on that host [87].

In addition to the different categories of IDS, we can also distinguish between different approaches to intrusion detection. Either a so-called **signature based** approach or a so-called **anomaly based** approach.

Signature-based detection involves employing a predefined set of patterns or signatures, which can vary from straightforward byte sequences indicative of specific malicious activities or objects to intricate templates with non-contiguous sequences and wildcards. This method is utilised by Intrusion Detection Systems (IDS) to compare and identify these signatures within monitored data, such as network traffic or system files [87].

Anomaly-based Intrusion detection systems involve creating a model of what is considered normal activity and identifying anything deviating from this model as suspicious. One method to establish the "normal" model is by initially placing the IDS in a "learning" mode, where it observes and learns the typical behaviour of the specific host or network to be protected. After this learning phase, the system transitions to an "alerting" mode, flagging any deviations from the learned normal behaviour as potentially suspicious. Alternatively, another approach to learning normal behaviour involves using machine learning techniques on large, more generalised datasets. In this method, the IDS learns typical "normal traffic profiles" by analysing common applications and protocols found in various office networks. This broader approach aims to capture generic patterns of normal behaviour that can be applied across different environments [87].

It is therefore easy to draw a diagram (Figure 2.3) showing the different categories of intrusion detection systems that exist today.

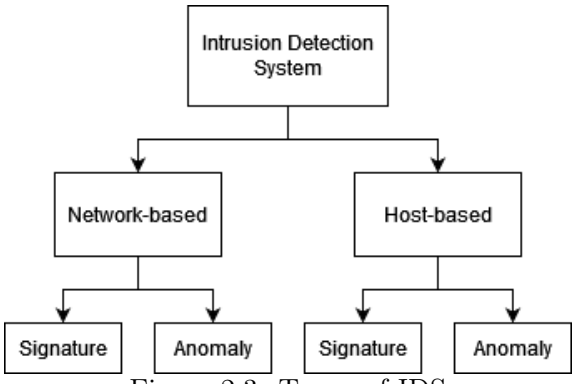


Figure 2.3: Types of IDS

Logging

Monitoring an IT environment involves compiling a comprehensive record of events from diverse sources such as applications, services, hosts, network components and security devices. To achieve this, logging data generated by these sources must be aggregated, converted into a standardised data format and subjected to analysis.

A rather naive approach to the logging method would therefore be to record all the events from all the applications and services. Unfortunately, it's easy to see that such an approach has its limits in terms of storage and analysis. Especially if we focus on network logging, we could have terabytes of data in just one day. So it's vital to first identify the type of event we want to record, and then only record things that are of interest for malware detection purposes. All logging information retrieved must be stored in a log file, which must be structured correctly. Respecting a defined structure means that log file entries can be analysed easily and efficiently. To meet all the requirements of a logging system. The paper written by Kent and Souppaya and published by NIST provides a good explanation of how to manage such a logging system. Explaining how to set up an infrastructure, how to plan logs and also how to have an operational log management process [59].

Finally, the last point that can be highlighted for this mechanism in relation to intrusion detection systems is that it enables activity traces to be analysed afterwards. Whereas an intrusion detection system acts automatically and directly on the system being analysed, the logging mechanism allows the data to be analysed by a human expert or a machine learning algorithm. For example, we could imagine collecting logs from a host over a certain period of time and then saving them for analysis at a much later date.

Honeypots

A honeypot is a deliberately deployed asset in an information environment, devoid of any actual operational use, with the explicit purpose of luring and potentially attracting attackers. This strategic placement allows for minimal false alarms, as the absence of legitimate production activity makes every interaction with the honeypot a significant event from a security standpoint.

One of the popular implementations of honeypot was proposed by Provos N. [71]. The article introduces Honeyd, a virtual honeypot framework designed to deceive fingerprinting tools like Nmap and Xprobe. Applications in security include network decoys, worm detection, countermeasures, and spam prevention. The framework's ability to mimic different operating systems is highlighted.

SIEM

There are therefore a huge number of different sources of data for intrusion detection in systems. It is therefore necessary to set up systems to retrieve all this information and manage it centrally. Such systems are called Security Incident and Event Management or SIEM systems.

There are plenty of SIEM solutions available on the market today. They all have their technical and commercial advantages and disadvantages. Various solutions such as RSA,

SolarWinds, ArchSight, QRadar, McAfee, USM-OSSIM, Splunk and LogRhythm were analysed by [47]. One interesting conclusion drawn by the authors is that most of these tools have a pleasant graphical interface, but are not particularly effective at processing the large number of events collected. It is therefore necessary for SIEMs today to develop visualisation and analysis extensions that allow users to have a correct overview of the state of the system and the various events in order to react as effectively as possible.

Threat hunting

Finally, no matter how many detection measures we put in place through logging, IDS, SIEM, there will always be malware that will not be detected by our systems. That's why we sometimes go on a malware hunt to find out what has slipped through the internal network. Once we've done that, we can improve our intrusion detection systems so that these threats can no longer cause an incident on our network. This process of actively looking for threats in our system is called threat hunting [7].

Static malware analysis

When we talk about static malware analysis, it is when we analyse the malware without running it. We will therefore look at the code or different artifacts that could have been modified on the exploitation system. For example, looking at the presence of certain strings or checking the functions imported into the code. On Windows, we could also look at the windows registry to determine if something interesting is present. Unfortunately malware writers use several techniques such as obfuscation or packing to make the work more complex. This is why malware analysis is not only static but also requires a dynamic component [40].

Dynamic malware analysis

Despite all the obfuscation that a malware code can have, when a malware runs, it will inevitably have some behaviour. So we will try to run the malware in a controlled environment and try to analyse it. For instance, we will be able to investigate if it has modified some elements on the operating system to maintain persistence. We can also analyse the network behaviour to look if it is trying to communicate with the outside world. To do this, we can use sandbox that will isolate the malware and be sure that it does not propagate in the network. A study by Or-Meir et al. [69] makes a comprehensive survey of dynamic malware analysis techniques in the modern era.

2.2.2 Malware detection applied to mobile malware

Now that we know how it is possible to detect malware in a system in general, it is worth presenting a review of the latest work on malware detection applied to mobile devices. To do this, we can draw a parallel with intrusion detection systems. Accordingly, this subsection will first present a review of the various HIDS that have been developed for mobile devices. Then, it will be interesting to talk about NIDS.

HIDS for mobile devices

HIDS for mobile devices perform 3 main types of analysis. Firstly, detection by signature verification. To do this, the HIDS has a list of malware signatures that it compares with

the applications installed on the phone. Secondly, anomaly based detection is often carried out using models that have been trained on malicious application datasets. Finally, the last analysis method, which is specific to mobile devices more than any other system, is the analysis of the permissions given to the mobile application [92]. An application requesting full permissions can be suspect.

So host-based intrusion detection systems do exist for mobile devices, and can offer different levels of performance. However, given our context, the use of such an intrusion detection system is not ideal. To do so, an application would have to be installed on the phone, which is not always possible. Despite the fact that the HIDS do not apply to our context, we can cite some papers that tried to go in this direction [11, 46, 72, 73]. In addition, many anti virus companies offer a version for mobile devices like AVG[3].

NIDS for mobile devices

Network traffic capture point When it comes to detecting malware by analysing network traffic, there are a huge number of solutions proposed in the literature. Most of the proposed methods use machine learning or deep learning algorithms to analyse network traffic and detect the presence of malware. One thing that most scientific papers do not take into account, and which can have an impact on the tool used, is the point of capturing. There are several possible points of capture for recovering and analysing network traffic. Sensibly, the traffic captured will not change, but given that the algorithm or tool is running on a different system, this implies that it has different computing resources. It is therefore difficult to train a deep learning algorithm on an embedded system, for example, but inference becomes possible on small devices. That's why it's worth looking at this point first, before talking about the different tools that can be used. For this, we base ourselves on a comprehensive survey by Conti et al. [36] highlighting these different capture points in several situations.

To talk about the different points where the traffic is captured, we can simply mention the different points:

1. On device: Network traffic is captured directly on the phone via an application.
2. Off-device
   (a) Wired network equipment: All traffic is redirected to a server where it is analysed.
   (b) Access point: The phone is connected to an access point and everything is analysed directly on the AP.

In the first case, where network traffic is recorded directly on the phone, the capture point is directly in the mobile device. This means that an application is installed directly on the phone to retrieve all the network traffic. This approach, like the others, has a number of advantages and disadvantages. There are 2 major advantages to this method:

- **Easy identification of traffic origin**: All the traffic is retrieved from the mobile device, so it is easy to identify that the traffic belongs to that phone. This is not the

[3] https://www.avg.com/en-us/antivirus-for-android#pc

case with network approaches, where traffic is captured from several mobile devices at the same time, and when we want to analyse the traffic, we have to re-identify which phone the traffic comes from.

- **Easy to focus on specific applications or interfaces**: So it's easy to see that we can easily choose to capture traffic from just one application or from an interface. It is therefore easier and cheaper in terms of hardware costs to capture cellular data where, with external equipment, we would have to use an IMSI-catcher.

Despite its advantages, this point of capture has a significant disadvantage that is important to highlight.

- **Lightweight application**: Such an application must be lightweight, i.e. it must not be too greedy in terms of computing demand, must occupy little memory space despite the amount of traffic to be stored and must not consume a lot of battery power.

With this point of capture, several methods have been presented in the literature. One example is Shatbai et al. [79] who used machine learning algorithms on cellular/Wi-Fi packets per application in order to detect a malicious application. These algorithms proved to be very useful for logistic regression and the J48 decision tree with an accuracy of over 0.999. Interestingly, Su et al. [83] also used a J48 decision tree algorithm but only achieved 0.916 accuracy. The difference with the work of Shatbai et al. is that they used data from the second layer of the OSI model. This clearly shows that even if the point of capturing is interesting, the choice of content on which to base the results is very important. Finally, more recent work includes that of Arora et al. [24] who focused on IP packets to run supervised machine learning algorithms. The results in terms of accuracy are fairly poor, with an accuracy of 0.873 for the best case.

In short, this capture point will be useful if we want to focus on mobile malware that exfiltrates private data from the mobile device via cellular data. Otherwise, it still has the significant disadvantage of having to be lightweight and manually installed on the phone. Most of the time these NIDS are also HIDS since they will take advantage of being on the machine to look at more precise stuff like different files, permissions, etc

Then, in the second case (2a), network traffic is captured using a network device located between the access point to which the mobile device is connected and the Internet. These different network devices can be VPN servers, Internet gateways or simply computers whose task is to be dedicated to the recovery and analysis of network traffic. The main disadvantage of this method is that if we find ourselves in a configuration where machines other than telephones are connected, it will be necessary to sort out the network traffic coming from the telephones if we want to concentrate solely on them. Nevertheless, this method has a significant advantage that we don't have in other cases. This important advantage is the fact that we have greater computing power. By redirecting the network traffic to a machine with more power, we will be able to run more demanding machine learning algorithms for traffic analysis.

Thanks to this, some papers in the scientific literature have developed different detection methods based on this capture point. Wei et al. [28] for example used an Independent

Component Analysis algorithm on DNS query data with an accuracy of almost 100%. Clearly, the algorithm can be easily bypassed by malware writers using DNSSEC or IPSec, but it is rarely used. Apart from this work, there is also that of Wang et al. which uses a C4.5 decision tree algorithm that focuses on the content of HTTP requests and TCP headers to estimate whether traffic is malicious or not. This algorithm gives an accuracy of 99.7% in the best case. Unfortunately, when focusing on HTTP traffic, the algorithm can be easily bypassed if the adversary uses an HTTPS channel to exfiltrate the data. Finally, the article [43] by Fend et al. does not identify exactly which capture point they are using, but they present an approach that incorporates deep learning. So, if we want to take advantage of the fact that we have a server with good computing power, this is the kind of algorithm we prefer.

Finally, the last capture point (2b) is directly on the access point to which the phone is connected. Generally speaking, in the context of capturing network traffic from a mobile device, this capture point has been used for PII (Personally Identifiable Information) leakage detection [82] or user action identification [86]. However, among the scientific papers focusing on malware detection for mobile devices, there don't seem to be any papers taking advantage of this capture point for malware detection. Thus, it is interesting to see what advantages and disadvantages this capture point could bring to malware detection. The first advantage is the mobility of such a NIDS. Unlike the other points mentioned above, moving the NIDS means that it can be used in all circumstances, even when the mobile device is not connected to a network. In emergency situations where it is necessary to know whether the mobile device is infected and there is no network nearby, this point of capture directly on the AP is the ideal situation.

When we talk about an access point, we think directly of a Wi-Fi router. However, other solutions with embedded systems can be considered, such as the Raspberry Pi. A Raspberry Pi has the advantage of being mobile, easy to deploy, inexpensive and easy to configure, unlike a router. This makes it easy to deploy portable NIDS. On the other hand, it is true that one of the disadvantages that we had in the first case, where the capture point is on the phone, is partly present: a Raspberry Pi does not have very high computing power, which limits the malware detection algorithms that could be deployed.

In short, there are 3 main capture points for harvesting network traffic: on-device, with wired network equipment or directly on the access point. These different points don't really offer any advantages in terms of traffic captured, since we can see the same network packets in all 3 situations (except in the first case, where we have access to cellular data). Where certain parameters come into play is when we want to choose which machine learning algorithm to use. If we want something more powerful like deep learning or something with a lot of features, we're going to favour harvesting and analysing traffic on the server. In another case, where we want to prioritise mobility and the speed of deployment of a NIDS, we will favour the capture point over the access point. In short, the choice of capture point depends on the specific context and objectives [36].

Review of existing tools for Network Intrusion Detection Systems Now that we've considered the various possible points of capture, it's worth comparing the various existing malware detection tools to understand what they do and what advantages they

have over each other. A list of the various NIDS software tools is provided by NIST [77]. This list includes several commercial NIDS and three better-known open source IDSs: Zeek, Snort and Suricata. To review what exists in terms of tools and to be able to compare them and be sure of what they implement, it is necessary to focus on these open source tools in view of the fact that the tool implementation is freely available.

Snort [74] is a lightweight open source network intrusion detection system (NIDS) that can also be used as an intrusion prevention system (IPS). To fulfil its role as an intrusion detection system, Snort is capable of logging and analysing traffic in real time. In its NIDS role, Snort can identify buffer overflows, stealth port scans, SMB probes, semanting URL attacks, and so on. To do this, Snort has a list of rules to which it compares network traffic, and if one of the rules it has matches what it finds, an alarm is sounded. It is also capable of deep packet inspection (DPI), i.e. examining the data content of each packet, which is crucial for detecting threats that could be hidden at application level. Finally, Snort is quite extensible, with the option of installing third-party modules and plugins.

Zeek (formerly Bro) [70] is also an open source network intrusion detection software, but it focuses more on in-depth network analysis than Snort. Where Snort's approach is essentially based on signatures, Zeek provides logs that are richer in detail, including the protocols used, domain names, IP addresses and HTTP headers. All these details can then be used to carry out a more in-depth analysis of traffic by implementing anomaly-based detection algorithms.

Finally, Suricata[4], like Zeek and Snort, is an open source intrusion detection system. Its big advantage is that it has a multi-threaded implementation, enabling it to perform well in intensive network environments. To fulfil its NIDS function, Suricata has both rule-based and signature-based detection.

In short, Snort is advantageous in that it is not very performance-hungry, but on the other hand, as it only works in single-thread mode, it is less powerful for networks with a heavy load. As for Zeek, its main advantage is that it provides very detailed logs of network activity, facilitating analysis through various algorithms. However, it has the disadvantage of being geared more towards analysis than intrusion detection. It can therefore be slower for real-time detection of specific attacks based on known signatures. Finally, Suricata is multi-threaded, which gives it very good performance, but it has such a huge number of functions that it can be complex to configure for non-expert users. It is more than important to note that these tools do not feature machine learning algorithms for malware detection, whereas most scientific papers seek to develop them. These machine learning algorithms are very powerful for 2 main reasons. Firstly, because they can detect new malware that is not listed in the IDS signature-based databases. But also because some machine learning algorithms can classify traffic even when it is encrypted. Unfortunately, most of the tools used at the moment focus on signatures, because despite the fact that there are a huge number of machine learning algorithms presented in the papers, these methods have too high a false positive rate.

[4] https://suricata.io/

Logging

It is possible to choose to log events directly on the mobile device or from the network. In our context, we are unable to log elements directly on the mobile device. However, the ADB[5] or Logcat[6] tools can be used to analyse system logs on Android. To analyze network logs from an access point multiple tools exist. Wireshark[7] is a tool with a graphical user interface to capture network packets and analyze them. Also, the tcpdump[8] tool exists and is used only on the command line. Finally, if we want to fulfill these objectives of network packet capture and analysis with python code there are different libraries. Scapy[9] is one of the many libraries with which it is possible to do a lot of manipulation with network packets.

Dynamic malware analysis

For dynamic malware analysis on mobile devices, various tools are available. Including tools that allow direct interaction with the application that is investigated through a USB connection. The context prevents us from working with these kinds of tools. Nevertheless, it is still interesting to mention classic tools such as MobSF, JADX, Frida or Objection. A very interesting tool that could be used for dynamic network analysis is Burpsuite. Indeed, it is possible to install our own certificate from Burpsuite in order to read the encrypted traffic that could be sent by the phone [30]. We could also bypass SSL pinning in order to be able to analyze what a malware sends as data [12]. Unfortunately, these techniques require to interact physically with the mobile device. However, this is not what it was in the basic project and can be addressed in the future work. Finally, it can be noted that there are a lot of tools to analyse the traffic of a network generated by malware. All these tools are grouped by the community and available on Github[10].

2.2.3 Developing a stand-alone solution

Once we know where we are going to capture the traffic and what tools are avaiblable, it is worth remembering that we want to limit data extraction. With today's solutions, most NIDSs are just passive and only look at the traffic going out to the Internet. A more effective solution to avoid data exfiltration at the same time as detection is to simulate a network to isolate the mobile device. We can therefore conceptualise a fake access point that, while not connected to the Internet, simulates internet services to deceive the malware into believing it has internet access. To do this, we can take an interest in the tools that already exist, despite the fact that there aren't many of them. Currently, the 2 main tools used to simulate Internet services in order to set up a dynamic malware analysis laboratory are FakeNet-NG[11] and INetSIm[12].

FakeNet-NG was developed by Andrew Honig and Michael Sikorski and the first version was released in 2016. Its aim is to observe the characteristics of certain binaries in secure

[5] https://developer.android.com/tools/adb?hl=en
[6] https://developer.android.com/tools/logcat?hl=en
[7] https://www.wireshark.org/
[8] https://www.tcpdump.org/
[9] https://scapy.net/
[10] https://github.com/rshipp/awesome-malware-analysis?tab=readme-ov-file#network
[11] https://github.com/mandiant/flare-fakenet-ng
[12] https://www.inetsim.org/

environments. By simulating the various services that the Internet comprises, the malware awakens its network signatures by querying certain URLs, requesting certain command & control domains, etc. More specifically, by default, the Internet services that FakeNet-NG can simulate are DNS, HTTP, HTTPS and SMTP. One notable thing about this tool is that it must be run on the same host as the malware. This has the main advantage that fakenet is able to detect the name of the binary generating the network traffic. Unfortunately, this means that fakenet is not applicable to the various capture points cited in Subsection 2.2.2 about capture points. Finally, it should be noted that the tool is developed in Python, which makes it easy to implement new services and features.

INetSim (Internet Services Simulation Suite) is an open source tool developed by Thomas Hungenberg and Matthias and written in Perl. The project was launched in their spare time with the aim of quickly analysing the network behaviour of unknown malware in a laboratory environment. Currently, the various Internet services simulated by the software are HTTP/HTTPS, SMTP/SMTPS, POP3/POP3S, DNS, FTP/FTPS, TFTP, IRC, NTP, Ident, Finger, Syslog. It should be noted that the implementation of the HTTP and HTTPS protocols is quite advanced, since INetSim can respond to most of the files requested by the malware. For example, if the malware requests access to a JPEG file, the software will be able to send back a correctly formatted JPEG image. As a result, the server will not return a 404 error and the malware will continue to run thinking it is really connected to the Internet. Unlike Fakenet, INetSim does not need to be installed on the same host as the malware in order to function correctly and redirect network traffic. Most of the time, INetSim is installed on an initial virtual machine that will capture all the network traffic generated by another virtual machine where the malware is running. However, it seems possible to install it on a network component [75] to simulate an Internet network. Finally, a considerable advantage of INetSim over FakeNet-NG is that it is possible to interact with the malware. For example, one can listen on port 80 thanks to netcat if one knows that the malware communicates in HTTP. In this way, one can pretend to be the attacker's C&C server and send commands that could have been found through static analysis.

Future work

There are not many scientific papers on simulating Internet services. There is one paper in the literature that develops a TRUMANBOX tool [49], but it does not seem to be used in practice by malware analysts. INetSim seems to be the most widespread solution at the moment, but it is written in Perl, the last update dates from 2020 and the developers do this in their spare time. It is therefore quite clear that the tool may have limitations that have not been studied in the scientific literature and that could improve the tool by anticipating malware behaviour more accurately. As a result, there is an opportunity for research in this area where the idea would be to build a powerful tool in a recent programming language that anticipates possible malware escapes and uses current technologies to simulate Internet services.

2.3 Locked mobile device data acquisition

2.3.1 Challenges in acquiring data from locked devices

Acquiring data from locked mobile devices can be very difficult. Indeed, phones today have several security mechanisms. First of all, some Android devices offer file-based encryption since Android 10.0[13]. This makes it difficult to recover data at rest without the decryption key. Then it can be noted that there is a passcode protection used to prevent the authorization of access to the phone. However, some PINs can be bruteforced. Biometric authentications also provide an additional layer of security. Some software can be used to bypass operating system or firmware security mechanisms. For instance, tools like Cellebrite UFED[14], GrayKey[15] or Oxygen forensic detective[16] can exploit vulnerabilities to unlock devices without knowing the passcode. As phone technologies are constantly evolving, experts must constantly maintain or current advanced products. In addition, some reference papers such as the NIST guidelines or the Barmpatsalou et al. survey can be consulted to understand the different methods [27, 29].

In addition to software-based methods, some hardware-based methods can be applied on the mobile device. For example, JTAG and chip-off methods involve accessing the phone's memory directly by physically connecting to it [33]. More sophisticated methods allow to cold boot attacks to recover Android decryption keys by cooling the phone below 10 degrees [68].

Finally, we can do without software and hardware methods to focus on more conventional investigation methods. First, we can simply ask the user for the code during an interview. Then, we can analyse the seized material in order to know if the user has noted the password on a piece of paper or something else. In addition, we can also analyse the screen of the phone with a Smudge attack as [25] to guess the code of the phone. The last investigative method is to ask the provider. It is indeed sometimes possible to ask the PUK code to the provider in order to be able to reset the PIN code and have access to the phone.

2.3.2 Probe requests analysis

Unfortunately, the different techniques available today do not really adapt to our context. Indeed, it is either necessary to buy fairly expensive commercial products, or it is necessary to disassemble the phone to be able to inspect it. So there is an artefact that the papers we discussed did not receive, they are probe requests. The probe request is a type of management frame for Wi-Fi connectivity of devices. When a device is not connected to any network, it will send a series of probe requests to announce the names of the access points it already knows. As said in the intro, this is an interesting artefact on the one hand because it can be used with a tool like WiGLE but also because this data makes it possible to uniquely identify the user. This is because the phone has a more or less unique list of access point names. These probe requests make it possible to track the movement of users in certain scenarios. This shows the interest of analysing probe requests even if

[13] https://source.android.com/docs/security/features/encryption/file-based
[14] https://cellebrite.com/en/ufed/
[15] https://www.magnetforensics.com/products/magnet-graykey/
[16] https://oxygenforensics.com/en/

they do not seem to reveal private information at first [37, 44, 50].

Chapter 3
Bridging research and application

3.1 Purpose and overview

This short chapter aims to make the transition between the state of the art (covered in Chapter 2) and the practical implementation of the project (to be detailed in Chapter 4). Here, we summarise the findings from the state of the art review to inform our selection of tools and methodologies for detecting mobile malware. First, we will highlight the key elements of the state of the art in mobile malware detection. Following this, we will present the different criteria for selecting appropriate tools depending on the IoCs and the specific requirements of our context. Finally, we justify the choice of tools and discuss their applicability in preventing and detecting data exfiltration. The advantage of such an approach will allow a logical flow from theory to practice enabling the reader to correctly understand the methodological choices as well as the implementation in the following chapter.

The state of the art begins with a first part on mobile security. This part is intended to understand the domain before delving into malware detection. It is precisely in the malware detection section that we have highlighted some tools. First, for intrusion detection systems by the network, we had seen several tell tools like Bro, Snort or Suricata. There are also several experimental machine learning algorithms aimed at improving the detection of network patterns. Then, regarding logging tools, we approached Logcat machine side. On the network side, we talked about tools like Wireshark, tcpdump and Scapy. On the dynamic malware analysis, we talked about Burpsuite. Finally, regarding the emulation of Internet services, we mentioned INetSim and FakeNet-NG. The advantages and disadvantages of these different tools have already been compared in the state of the art. Nevertheless, this section is intended to support the tools chosen and the reason for this choice for the sake of clarity.

3.2 Tools and technologies used

3.2.1 IoCs: Context and coverage

In order to correctly choose the tools we have identified, it is important to highlight the different IoCs that exist. Once these IoCs are highlighted, it is interesting to see which ones we can find in our context. By combining the requirements of our context, the IoCs and the tools we have identified, we can have a robust approach to knowing what we can identify and what we cannot highlight. IoCs can be of different types. They can be host-based, network-based, file-based, behavioral or metadata. Our context requiring us to identify the IoCs by an access point, therefore directly implies to focus only on the IoCs passing through the network. Cloudfare identifies these different IoCs networks as malicious IP addresses, domain names, URLs, network traffic patterns, unusual use of ports, connections to hosts known to be malicious or network exfiltration patterns [19]. It is therefore necessary that the tools we select can identify as many of these IoCs as possible, while being able to meet the requirements of our context. Finally, we note that our project will not allow us to

identify all IoCs. For more advanced malware detection, it will be necessary to investigate the various artifacts left on the host.

3.2.2 Tool selection

First, regarding the intrusion detection systems by the network, we have discussed different machine learning algorithms. There is indeed a lot of research that has been done to use machine learning algorithms for intrusion detection. However, the detection of anomalies for the network is not suitable for our project given the high rate of false positives and the lack of robustness. This generation of false alarms would distract a digital forensic analyst from the real threats. This high generation of false alarms and lack of robustness is not unfounded. Indeed, the [26] and [80] papers highlight the success of such systems in operating environments as very limited. Given that our system is supposed to feature IoCs to help the work of a digital forensic analyst, it is easy to understand why no machine learning algorithm will be used in our project. This highlights the distance that can exist between research and practical application. Despite this, it is now interesting to highlight the use of existing tools. For the project we will use Snort. This is explained by the fact that, as identified, Snort is less resource-consuming than Zeek or Suricata. Since we use a Raspberry Pi, it makes sense in view of the limited resources of such a device. So it is thanks to Snort that we can cover the IoC of the suspicious network traffic patterns. Then, at the level of network logging, we will choose Scapy. As mentioned before, using python will allow a better scalability of the project. In addition, Scapy will allow to extract important IoCs such as DNS domain names as well as suspicious IP addresses. Finally, the tool will log all the network packets sent by the phone on the wireless interface. This network log file will be saved in a pcap. This will allow a post-capture analysis to identify IoCs such as URLs, data exfiltration patterns or unusual use of ports.

3.3 Validation of chosen approaches

Now that we have chosen which tools we will use in our project, it is necessary to justify their usefulness. Indeed, we are trying to ensure that we can detect attackers thanks to the chosen tools. If we can do malware detection, we will also justify to what extent we can detect attackers. To do this, we will use the pyramid of pain from the discipline of threat intelligence. Despite the fact that this discipline is quite far from digital forensics, its use may be justified. Moreover, given the fact that we are in a sensitive environment, it is necessary to highlight the exfiltration of data that we are able to block. In addition to being able to block exfiltration of certain data, we can also detect certain types of data extraction. This analysis is highlighted in the Subsection 3.3.2.

3.3.1 Pyramid of pain

The pyramid of pain comes from the discipline of cyber threat intelligence. The purpose of this discipline is to collect as much information as possible about different existing or emergent threats. These threat insights can contain their motivations, attack methods, and targets. The goal of collecting this data for an organization is to better prepare for its threats. This allows it to improve its detection and response systems against an attacker [78]. A concept related to cyber threat intelligence is the pyramid of pain. The different levels of the pyramid describe the difficulty for the origin of a threat to succeed

an attack when certain indicators are blocked by a defender. We can therefore understand that the discipline of cyber threat intelligence is quite different from what can be found in digital forensics. Nevertheless, we can see that these two disciplines have the common point of improving the detection of a threat according to some indicators. In short, we can use the pyramid of pain to understand what "pain" we can do to an attacker but also what type of advanced attacker we cannot cover.

The pyramid of pain was conceived in 2013 by David J. Bianco following the release of a report on APT1. This pyramid of pain was created because professionals struggled to correctly apply indicators of compromise to block this threat. The pyramid of pain illustrated in Figure 3.1 is as follows: inside the pyramid is the IoC and on the right side is the pain that is done to the attacker. For example, if we manage to detect and block the *tools* that the attacker uses then it will be *challenging* for the offender to infect us. This form of pyramid allows us to evaluate the pain we do to the attacker but also to what extent we are able to block an attacker. Also what type of attacker we can block [32].

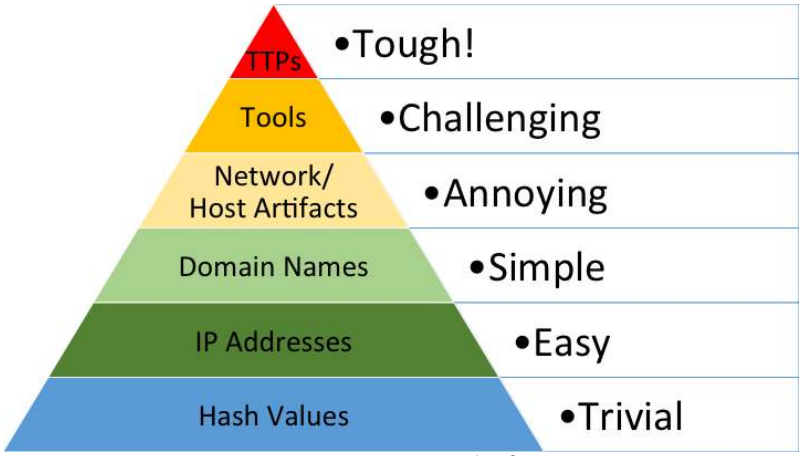


Figure 3.1: Pyramid of pain

We can therefore highlight what we cover thanks to the tools we have selected for our project. For the first level of the pyramid, it is difficult to recover the hash values of the files since our context of does not allow us to have access to the mobile device. Regarding the second and third level of the pyramid, we could use tcpdump. This would allow to output IP addresses that are suspicious compared to the rest of our traffic. Nevertheless, for reasons of scalability of the project, we will favor an approach with Scapy. Developing the project in python will easily improve the project for future work. Then, concerning the fourth level, it is partially covered thanks to the use of Snort and Wireshark. Snort will track malware signatures. Wireshark will highlight URI patterns, server C&C information and all network packets. The uncovered part of this fourth level are host artifacts. This is because we do not have access to the host. Finally, the last two levels are not covered by the tools that could be selected in this project.

In conclusion, the pyramid of pain is a very interesting framework for cyber threat intelligence, which enables us to highlight the different levels of difficulty of attackers when certain indicators are detected. In our project, given that we have limited access to hash values, we are using tools such as Scapy, Snort and Wireshark to cover the different levels of the pyramid of pain. However, some aspects such as host artifacts and advanced attacker techniques remain beyond our current capabilities. This structured approach allows us to

understand the strengths and limitations of the implementation we are going to put in place.

3.3.2 Data exfiltration prevention and detection

After assessing the complexity of the attackers we can detect, we can look at the coverage of data exfiltration. This is important given our context. As highlighted in the introduction of this context, the military context requires us to prevent the exfiltration of sensitive data. It is therefore important to resume the taxonomy of data exfiltration that we had put forward in the state of the art thanks to the advances of the field. A modified version is given in Figure 3.2. In this figure, what is in blue describes the means of exfiltration that we can block during malware analysis. The elements in green are those that we can detect during an analysis thanks to the project. We can therefore note the usefulness of the Faraday enclosure. Indeed, all means of data exfiltration are covered by the Faraday enclosure.

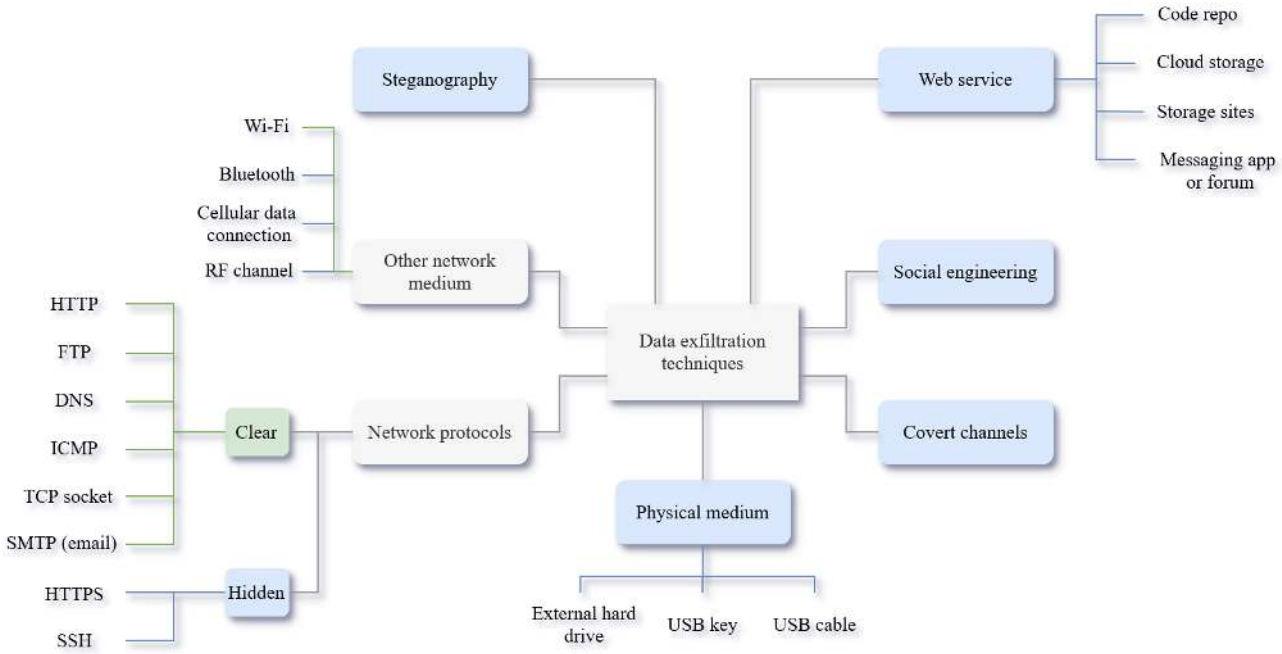


Figure 3.2: Data exfiltration coverage

3.4 Summary

In short, we have succeeded in making a transition between the theoretical framework of Chapter 2 and the practical implementation of Chapter 4. Additionally, we presented arguments against the use of machine learning for our project due to its high rate of false positives and lack of robustness in practical applications. Furthermore, we validated our approach by aligning it with the pyramid of pain, illustrating the levels of detection and prevention we can achieve against potential attackers. Finally, we examined the taxonomy of data exfiltration, highlighting how our chosen tools and methodologies can prevent and detect various exfiltration attempts. Overall, this chapter sets a clear, logical transition from theory to practice, providing a solid foundation for the detailed implementation

strategies to follow in Chapter 4.

Chapter 4
Usage & Implementation

4.1 Global methodology

As a reminder, we have 3 solutions to develop in this project. Firstly, working in an isolated environment thanks to the Faraday enclosure. Secondly, the purpose is to design an intrusion detection system within this box, activated when a soldier suspects malware on their mobile devices. Finally, to recover the probe requests from a captured mobile device in order to recover interesting information.

To create an all-in-one product, the idea is to put a Raspberry Pi inside the Faraday enclosure. This setup will enable to switch between two modes.

1. The first mode will be the intrusion detection system mode, which will make it possible to estimate whether or not the phone is infected by malware.

2. The second mode will capture probe requests sent by the locked mobile device.

4.2 Usage

When the user wants to interact with the product, this is how it looks (Figure 4.1). The Raspberry Pi is located inside the Faraday enclosure. To this Raspberry Pi is connected 3 LEDs and one button as shown in the Figure below. These elements make up the user interface, enabling the user to interact with the various modes of the application.

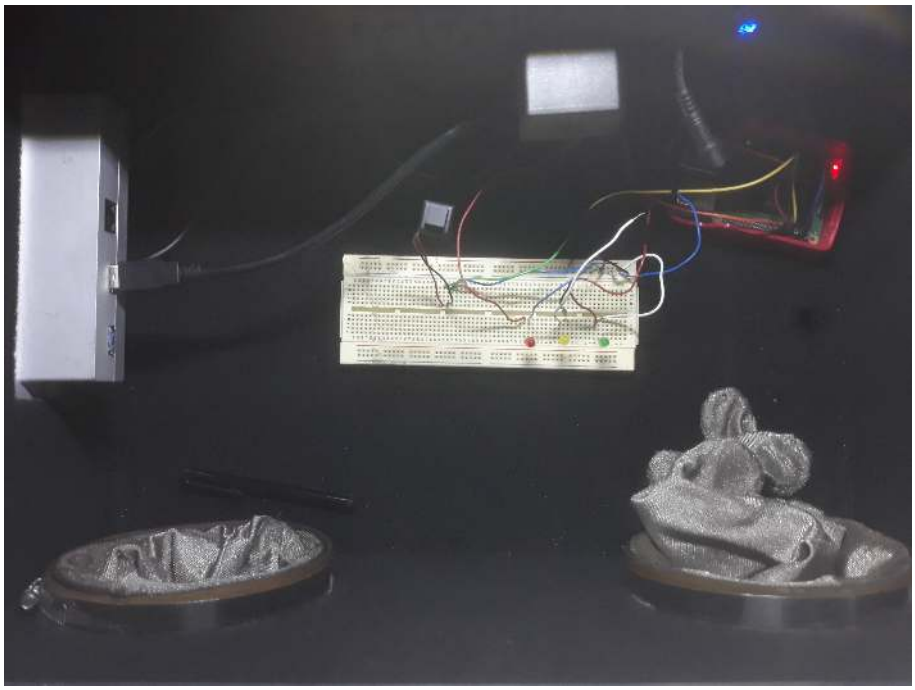


Figure 4.1: Overview of the product

When the user plugs the Raspberry Pi into the socket, the program will launch. For the sake of clarity, we have divided the use of the program into 4 stages.

Side note

For simplicity's sake, a quick reference guide has been included in the Appendix B. The idea behind this guide is to create something simple and quick to understand. It has been made in such a way that it can be stuck to the top of the Faraday enclosure.

4.2.1 Launch the Raspberry Pi

At the first launch, i.e. simply when you plug in the socket, the yellow LED will light up. This indicates that the program has been launched and that you are now able to interact with it.

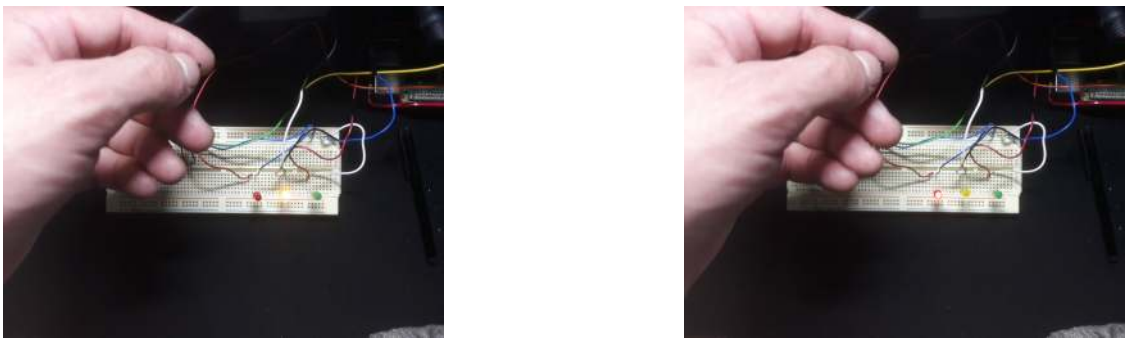
(a) Power plug disconnected (b) Power plug connected

Figure 4.2: Connect the Raspberry Pi plug

4.2.2 Mode selection

Mode switching

Once the yellow LED is on, you can interact with the button to switch between the 2 modes. Pressing the button while the yellow LED is on will activate the red LED and turn off the yellow LED. Conversely, pressing the button while the red LED is on will turn off the red LED and illuminate the yellow LED.



(a) IDS mode selected (b) Probe requests mode selected

Figure 4.3: Mode switching thanks to the button

The meaning of these LEDs is quite simple. They indicate the mode in which the user is going to run the program. In other words, it is possible to choose whether the Raspberry Pi

will behave like an intrusion detection system or whether it will capture the probes requests.

When the yellow LED is selected, this means that the Raspberry Pi is in IDS mode. When the red LED is selected, this means that the Raspberry Pi will capture probe requests.

IDS mode particularity

As mentioned in the introduction, the idea to detect whether the phone is infected by malware is to connect it to a Wi-Fi access point and capture the network traffic. As soon as the program is launched, the Wi-Fi access point is created and the phone can be connected to it. To connect, access Wi-Fi settings on the phone and search for available access points. Look for the network name "MasterThesis", which is the default name of the access point generated by the Raspberry Pi.

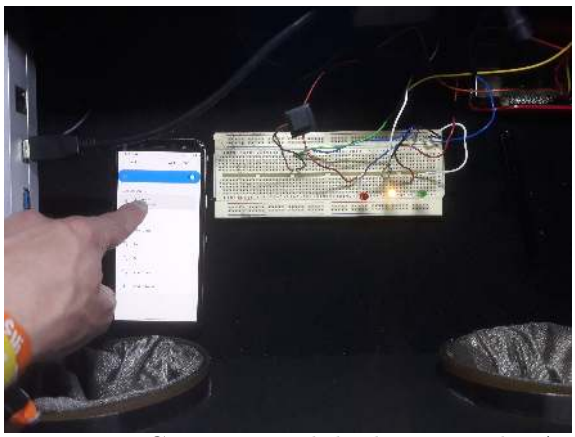


Figure 4.4: Connects mobile device to the AP

Side note

On the image provided, the name of the access point is "FreeWire". Obviously, the name of the access point is subject to change. The default access point name is set to "MasterThesis".

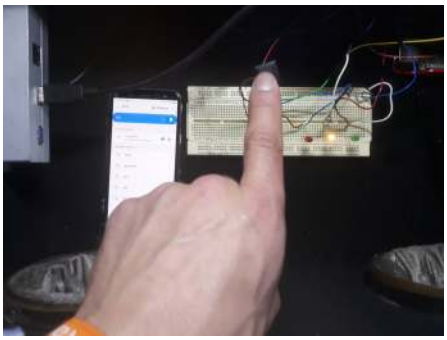
4.2.3 Launch the program

To select the desired mode, follow these simple steps:

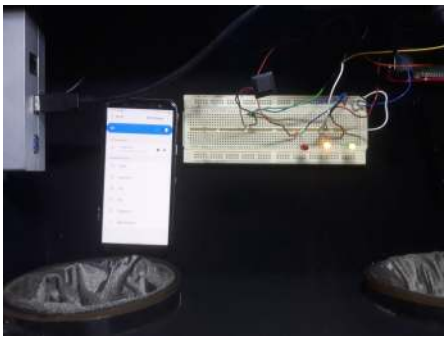
1. Insert a USB key into the designated port on the side of the Faraday enclosure where you wish to export the collected data.

2. Press and hold the button for approximately 5 seconds until the green LED illuminates. This extended press will activate the program.

Once the green LED is on, the program is running. During the operation, the green LED will remain illuminated, signalling that network capture is ongoing.



(a) Long press on the button



(b) Program launched

Figure 4.5: Launch mode thanks to the button

4.2.4 Save the capture on the USB stick

Regardless of the mode you're in, stopping data capture is simple: just press the button. After pressing the button, all three LEDs will blink five times before turning off. This blinking indicates that the data is being saved onto the USB key. Once the LEDs are all off, you can safely disconnect the USB key. The digital forensics team can then connect the USB key to a laptop to analyse the extracted data in more details.

Figure 4.6: Simple press on the button

4.3 Software design methodology

4.3.1 Overview of software components

In order to develop our project, we need several components. First of all, the choice of OS need to be carefully considered in order to capture the probes requests. Next, we need to present the tool we are using to create a Wi-Fi access point. Once the access point has been configured, we need to add the INetSIm tool in order to simulate an Internet network. Finally, once our wireless access point has been configured to simulate an Internet network, we need to detect potential intrusions. To do this, a python implementation is needed to capture traffic, record the domain names and suspicious IP addresses. The Snort tool is also installed to detect network traffic patterns.

The first thing that can be said about the software methodology is the OS chosen. Indeed, the operating system chosen is Kali Linux 2024.1, whose kernel version is Linux 5.15.44-Re4son-V7l+. The choice of such an operating system is not trivial, but is neces-

sary for a specific purpose. In fact, it is necessary to switch the Wi-Fi network card to monitor mode. Without this mode, it would be impossible to capture probes. For this reason, Kali Linux comes with the nexmon[1] utility. Nexmon is a C-based firmware patching framework for Broadcom/Cypress Wi-Fi chips that enables monitor mode. Monitor mode allows to hear all the radio traffic on a specific channel, rather than just the traffic directed to a specific device. Probe requests are broadcasted to all devices in range, not directed to a single recipient. So, in monitor mode, the Raspberry Pi becomes like a radio scanner, passively listening to all traffic on the designated channel.

Next, several tools are required to create our fake Wi-Fi access point. These tools are dnsmasq[2], dhcpcd[3] , hostapd[4] and INetSim[5]. The first 3 are used to create the access point, while the last is used to create a simulation of an Internet connection. The first tool is dnsmasq, which provides a DHCP server and does DNS caching. Dhcpcd will allow us to configure a static IP address for our wireless interface. Hostapd is a daemon that allows a network card to function as an access point. It implements the IEEE 802.11 standard for managing access points and also provides the various conventional authentication methods such as WPA, WPA2, WPA3, EAP, etc. So it's thanks to this tool that we can also define the name of the access point. Finally, INetSim is a software package written in Perl that simulates common Internet services in a laboratory environment in order to analyse the network behaviour of certain malware.

Finally, a python program will capture all the packets on the network and store them in a pcap file. This same python program will also be used to collect all the domain names consulted in the capture. In order not to overload this file of classic domain names, a whitelist is set up to record only unusual domain names. The program also gathers suspicious IP addresses, enhancing its capability to identify IoCs. These three artefacts — PCAP files, domain names, and suspicious IP addresses — can then be analysed by digital forensic experts. In addition, Snort can be used as an intrusion detection system by analysing the network. This will detect suspicious traffic generated by the mobile device.

4.3.2 Installing the OS on the Raspberry Pi

Installing the OS on the microSD card is not a problem, thanks to the Raspberry Pi Imager tool[6]. Once the tool is installed on the user's operating system, simply open it and select the microSD card. Finally, simply select the Kali Linux for Raspberry Pi 4 32-bit image as shown in the figure below.

[1] https://github.com/seemoo-lab/nexmon
[2] https://thekelleys.org.uk/dnsmasq/doc.html
[3] https://roy.marples.name/projects/dhcpcd
[4] https://w1.fi/hostapd/
[5] https://www.inetsim.org/index.html
[6] https://www.raspberrypi.com/software/

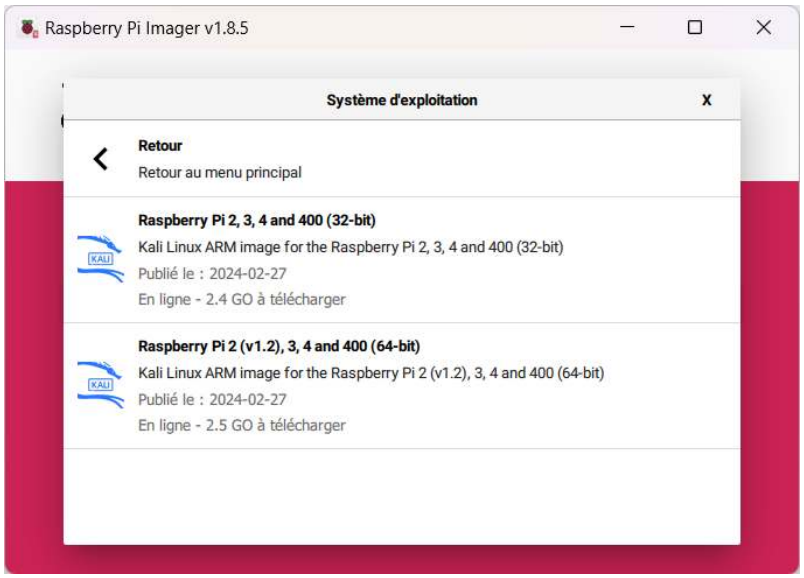


Figure 4.7: Kali Linux OS in Raspberry Pi Imager

4.3.3 Implementation methodology

The following points describing the implementation make up the bulk of this chapter. They aim to explain how the various tools have been installed, as well as explaining the code behind them. This will give the reader a clear understanding of how the project was built. Therefore, the subsection 4.3.4 is divided into 4 points. Firstly, we discuss the way in which the access point is created and configured. Secondly, the installation and configuration of the INetSim tool for simulating Internet services will be presented. Thirdly, there will be a short explanation of how Snort was installed and what rules were put in place. Fourth, the scenario where probe requests need to be captured will be addressed, with a paragraph dedicated to activating monitor mode and explaining probe requests. Finally, once all these tools have been installed and configured, it will be necessary to show in the subsection 4.3.5 how they have been put together to create an all-in-one product using python code.

Side note

To help the reader and developer, source code is provided[a]. We also provide the file structure in Figure 3.8 for ease of reading.

[a] https://gitlab.cylab.be/cylab/mobile-case.git

```
mobile-case
├── conf
│   ├── dhcpcd.conf
│   ├── dnsmasq.conf
│   ├── hostapd.conf
│   ├── inetsim.conf
│   └── rules.v4
├── scripts
│   └── startup
├── whitelists
│   ├── DNSWhitelist.txt
│   └── IPWhitelist.txt
├── main.py
├── config.py
├── captureTraffic.py
├── captureProbes.py
├── mountUSB.py
├── customExceptions.py
└── requirements.txt
```

Figure 4.8: Gitlab file tree

- **conf**: This folder contains a copy of the configuration files required for the various tools. It allows a developer to use it as a basic reference for a default configuration.

- **scripts**: This folder contains all the scripts needed to run the project. For the moment, it contains just one file containing all the commands to be executed at start-up.

- **whitelists**: This folder contains all the domain names and IP addresses that an Android or iOS phone can request for simple routine operations. This file was generated from a capture where three Android phones and one iPhone were connected for 24 hours.

- **main.py**: This file contains the python code that will run the whole software. This python file is called directly and imports the other python files.

- **config.py**: This file contains the python code used to set the basic configurations, such as the pin numbers associated with the LEDs.

- **captureTraffic.py**: This file contains the python code needed to capture the network traffic generated by the phone. It is not run directly but is called from main.py file.

- **captureProbes.py**: This file contains the python code needed to capture probes requests. It is not run directly.

- **mountUSB.py**: This file contains the python code needed to capture and mount the USB key plugged into one of the Rapsberry Pi's ports. It is not run directly.

- **customExceptions.py**: This file contains the exceptions specific to this project.

4.3.4 Tools installation

Create an access point

In order to configure the Raspberry Pi as an access point, we need first to install the dnsmasq and hostapd packages thanks to `apt` package management tool. Next, we'll need to modify the configuration files for these tools. Finally, we need to add these tools to the machine boot and reboot them for the configurations to take effect[7].

Thus, we need to modify the dhcpcd configuration file if we want to give a static IP address to our wlan0 interface. This is necessary for the Raspberry to act as a server. We're assuming here that we're using the 192.168.x.x standard for our wireless network and so assigning the IP address 192.168.4.1 to our Raspberry Pi. This can be done by modifying the dhcpcd configuration file `/etc/dhcpcd.conf`.

```
interface wlan0
static ip_address=192.168.4.1/24
nohook wpa_supplicant
```

Listing 1: `/etc/dhcpcd.conf`

We also need to configure the dhcp service so that we can assign an IP address to the devices that will be connecting to the access point. To do this, we need to edit the dnsmasq configuration file to give it the range of IP addresses we want to give to the devices. The way we configure our tool will allow us to give IP addresses between 192.168.4.2 and 192.168.4.20 to devices connected for 24 hours. This can be done by modifying the configuration file `/etc/dnsmasq.conf`.

```
interface=wlan0
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
```

Listing 2: `/etc/dnsmasq.conf`

In addition, we need to configure the name of the access point itself. To do this it is necessary to modify the hostapd configuration file `/etc/hostapd/hostapd.conf` and add the below information to the file. The channel is used to specify the channel number on which the access point will operate. This ensures that there is not too much interference from other access points. In our case, we can set it to whatever we like, as there will be no other access points in the Faraday enclosure.

```
country_code=BE
interface=wlan0
ssid=MasterThesis
channel=9
```

Listing 3: `/etc/hostapd/hostapd.conf`

We also need to tell hostapd where to find this configuration file. To do this, simply modify the `/etc/default/hostapd` file and find the line with *#DAEMON_CONF* and replace it like this.

[7] https://raspberrypi-guide.github.io/networking/create-wireless-access-point

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Listing 4: `/etc/default/hostapd`

Finally, it is necessary to enable the services when the Raspberry Pi boots. We also need to restart all the services for changes to the configuration files to take effect.. To do this, use the following command sequence.

```
kali@kali:~$ sudo systemctl unmask hostapd
kali@kali:~$ sudo systemctl enable hostapd
kali@kali:~$ sudo systemctl restart hostapd dhcpcd dnsmasq
```

Listing 4.1: Restart service

Simulate Internet services

To simulate an Internet connection to the access point, it is necessary to install and configure the inetsim tool. To do this, two main values need to be changed in the inetsim configuration file `/etc/inetsim/inetsim.conf`. Firstly, change the value of *service_bind_address* to specify the IP address on which the simulated internet services will be bound i.e. 192.168.4.1, typically corresponding to the static IP address we gave to the wlan0 interface previously. The second thing to change is the value of *dns_default_ip* to 192.168.4.1 also in order to specify the default IP address used by the DNS server for domain name resolution. Listing 5 shows how the configuration file should look after these changes. After these changes, don't forget to restart the inetsim tool and activate it to run at boot.

```
...
service_bind_address 192.168.4.1
...
dns_default_ip 192.168.4.1
...
```

Listing 5: `/etc/inetsim/inetsim.conf`

Then, once the tool is correctly configured, you need to redirect some of the traffic to inetsim. In fact, you need to redirect all the traffic apart from the DHCP packets. If we redirected the DHCP packets to inetsim, it would be impossible to assign an IP address to the mobile device we are connecting. Next, we need to record the firewall rules so that they are resistant to each reboot using `iptables-save`.

```
kali@kali:~$ sudo iptables -t nat -A PREROUTING -i wlan0 -p udp --
dport 67:68 -j ACCEPT
kali@kali:~$ sudo iptables -t nat -A PREROUTING -i wlan0 -j REDIRECT
kali@kali:~$ sudo iptables-save > /etc/iptables/rules.v4
```

Listing 4.2: Modify firewall rules to redirect traffic to inetsim tool

What's more, once we've redirected the traffic to inetsim we'll have to modify a tool that we've already installed : `dnsmasq`. Even if we have added firewall rules, dns requests will still be managed by this tool, as it is required to create the access point. So we're going to add a configuration line to the file so that all responses to any DNS request are 192.168.4.1 or the static IP address that we gave to the Rapsberry.

```
address=/#/192.168.4.1
```

Listing 6: The line to add at the end of `/etc/dnsmasq.conf`

Finally, we create a "startup" script which will restore the rules on reboot and which we add to a crontab.

```
#!/bin/zsh

iptables-restore < /etc/iptables/rules.v4
```

Listing 7: startup script

Open a crontab configuration file with `sudo crontab -e` and add the following line at the end of the file.

```
@reboot sudo /home/kali/Desktop/mobile-case/scripts/startup
```

Listing 8: Make firewall rules persistent

Install and configure Snort

Installing Snort is very simple and requires nothing more than being installed by the `apt` package management tool. The official Snort website allows to download detection rules developed by the community that can be added to the project[8]. Once the rule file has been downloaded, simply move it to the `/etc/snort/rules` folder. Once the community rules have been imported, you can modify the Snort configuration file `/etc/snort/snort.lua` to integrate these rules. To do this, modify the ips variable as follows.

```
ips =
{
enable_builtin_rules = true,
include = "/etc/snort/rules",
variables = default_variables
}
```

Listing 9: Make firewall rules persistent

We also want the alerts generated by the Snort rules to be saved on the USB stick. To do this, we need to find the `alert_fast` variable in the same configuration file like this.

```
alert_fast = {file = true, packet = false,}
```

Listing 10: Redirect alert_fast output in a file

Now, if you want to run Snort from the command line and direct it to the USB key. Here's the command we need to use. It's obviously for illustration purposes, as it will be run using the python script.

[8] https://www.snort.org/downloads#rules

```
kali@kali:~$ sudo snort -c /etc/snort/snort.lua -R /etc/snort/rules/
local.rules -i wlan0 -A alert_fast -s 65535 -k none -l /path/to/
USBKEY
```

Listing 4.3: Running Snort in a custom configuration to log on the USB key

Remark

The choice of running Snort on the pcap file and not during program execution is easily justified. Firstly, because running Snort at the same time as a python script results in a certain loss of information. In such a situation, some packets will be lost. What's more, given that we don't need Snort's live feedback and that we look at the output afterwards anyway, there's no point in having live processing. In short, the aim of such a choice is to ensure the efficiency and precision of the software.

Turn on monitor mode

For this part, the tools are installed automatically by the operating system. To activate monitor mode we use the `airmon-ng` tool, which is itself based on the nexmon firmware patching framework. If we take into account that we have already configured all the previous tools for IDS mode, then we need to disable them so that the wlan0 interface is not already occupied. You also need to remove the firewall rules to prevent traffic from being redirected. To do this, we have the following sequence of commands.

```
kali@kali:~$ sudo systemctl stop hostapd dhcpcd dnsmasq inetsim
kali@kali:~$ sudo iptables -F
kali@kali:~$ sudo airmon-ng start wlan0
```

Listing 4.4: Disable tools for IDS mode and enable monitor mode

Shortcut for this subsection

A summary, without explanation, of all the commands executed one after the other can be found in the Appendix A.2.

4.3.5 Python implementation

This subsection presents the Python code that runs the entire project. We're taking a top-down approach here, starting with the main.py file and then looking at which methods it calls. The aim here is not to explain the code line by line but rather the overall logic of each file so that it is more accessible to someone who would like to improve the project.

main.py

If we look closely at the body of the program in Listing 11, it's very short. Most of the code is in the **main()** function. However, it's interesting to note the way we handle exceptions. Since the user doesn't get error feedback on a screen, we make it visual through the LEDs using the **indicates_exception()** function. In addition, for ease of application development, the error traceback is recorded on the USB key or on the desktop if this is

not possible.

Apart from handling any exceptions, we can explain the overall logic of the **main()** function. This function is quite simple; it detects whether the button is pressed and lights the LEDs accordingly. Then, once it detects a long press, it will call the code in the **capturePrObes.py** or **captureTraffic.py** files, depending on the choice made by the user.

```
try:
        while True: # Let the program rerun automatically
                main()
except Exception as e:
        indicates_exception(red_led, yellow_led, green_led)
        try:
                usb_path = get_usb()
                exception_path = f"{usb_path}/exception.txt"
                store_exception(e, exception_path)
                raise e # Print it in the stdout even though
        except Exception as e:
                exception_path = "/home/kali/Desktop/exception.txt"
                store_exception(e, exception_path)
                raise e
```

Listing 11: Body of the `main.py` file

config.py

This file contains everything you need to configure the project globally. It is used for the classic setup of each pin on the Raspberry Pi board. In addition, it is this file that defines the pin number to which the LEDs and button are connected.

captureTraffic.py

The main function of this file is **capture_traffic()**, which is divided in three stages: pre-processing, processing and post-processing. Firstly, the pre-processing stage involves mounting the USB flash drive and defining the paths where the analysis files will be stored. In addition, whitelists from **DNSWhitelist.txt** and **IPWhitelist.txt** are loaded into memory. Next, the processing stage consists of sniffing all the packets received on the `wlan0` interface (see Listing 12). If a packet is a DNS request, the domain name is analysed and if it is not in the whitelist then it is recorded on the USB key. This processing stage is multi-threaded to detect user button presses for ending the sniffing process. as shown in Listing 12. Finally, the post-processing stage runs Snort on the generated pcap file, sorts domain names, and removes Snort's false positives.

```
t = AsyncSniffer(iface="wlan0", store=False, prn=process_packet)
t.start()
detect_button()
t.stop()
```

Listing 12: Processing part of the `capture_traffic()` function

captureProbes.py

This python file contains a main function called **capture_probes()**. This function is quite short but has a pre-processing part as well as a processing part. The pre-processing part consists of stopping all the tools used to create the access point, clearing the firewall rules and putting the wireless interface in monitor mode. Once in monitor mode, the interface is called `wlan0mon`. Listing 13 shows what the main processing of this function looks like. As before, the idea of multi-threading the sniffing comes from the need to stop the program when a button press is detected.

```
t = AsyncSniffer(iface="wlan0mon", store=False, prn=process_packet)
t.start()
detect_button()
t.stop()
```

Listing 13: Processing part of the `capture_traffic()` function

4.4 Fritzing diagram for hardware setup

Not many components are physically connected to the Raspberry Pi. However, it is always useful to provide a Fritzing diagram so that the developer can easily reproduce the connections. This diagram is shown in Figure 4.9.

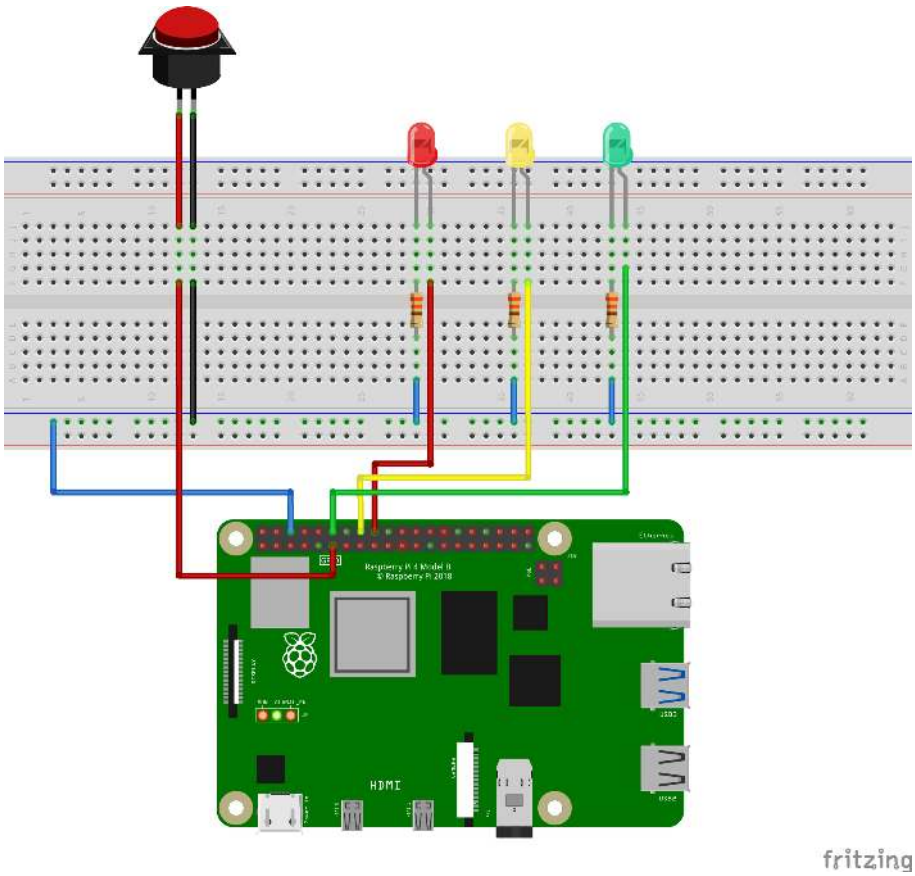


Figure 4.9: Fritzing picture of the connections

4.5 Additional considerations and access

4.5.1 Generating whitelists for DNS, IP addresses and Snort

To simplify the work of the digital forensic analyst, it is useful to sort the domain names and IP addresses that are considered legitimate. This will make it easier to identify IoCs in domain names and IP addresses. If we look at the network traffic of an uninfected mobile device, we can see that an astronomical number of domain names are requested. This subsection therefore explains how the whitelist of domain names in **DNSWhitelist.txt** was generated. In addition, within the code it was necessary to generate a whitelist for Snort rules. The construction of this whitelist is a slightly more subtle choice, since it was necessary for our particular case. Finally, it is also a good idea to whitelist legitimate IP addresses. This will make it possible to identify suspect IP addresses.

First of all, to create the whitelist of domain names, it was necessary to create a short python code. This script is not available on Gitlab but is provided in Listing 14. This simple python script using Scapy adds all the domain names received on the wireless interface to a text file. To retrieve the domain names, the idea was to make the Raspberry Pi a Wi-Fi repeater so that it could provide an Internet connection to the connected devices. In this way, all we had to do was people for their agreement to connect to the access point while they were carrying out their usual activities. We can therefore make the strong assumption that their phones are not infected with malware. We can make this assumption because none of the phones connected to the access point have been rooted or jailbreaked. Furthermore, they have only installed very basic applications. Additionally, the applications installed on the various mobile devices were downloaded from the official stores.

```python
from scapy.all import *

def process_packet(pkt):
    if DNS in pkt and pkt[DNS].qr == 0:
        query_name = pkt[DNSQR].qname.decode("utf-8")
        file = open("DNSWhitelist.txt", "a")
        file.write(f"{query_name}\n")
        file.close()

sniff(iface="wlan0", store=False, prn=process_packet)
```

Listing 14: Python code to generate the domain name whitelist

Next, we needed to create a whitelist for the Snort rules. This is because we have a tool that simulates Internet services. Some packets may therefore be considered suspicious because they do not appear the way a real Internet connection works. So, following the same logic as before, the idea is to connect uninfected mobile devices and look at the numbers of the rules that are generated. Afterwards, it will still be necessary to look at why these rules are activated before hardcoding them in the project. For ease of presentation, the rules we have decided to whitelist are presented in the table below. This table explains what the rule detects and why it is activated in our case.

Rule number	**Rule explanation**	**Reason for their activation**
137:2	An invalid SSL server HELLO was received without an SSL client HELLO having been detected	INetSIm may generate SSL responses in a simplified manner that does not conform to typical SSL/TLS exchanges. This may include sending a 'server HELLO' without the full SSL handshake process having been followed correctly, giving in this alert
116:444	(ipv4) IPv4 option set	INetSIm could add IPv4 options for various reasons, such as simulating certain network behaviours
119:260	The TCP connection was closed before the full HTTP message body was transferred. The length of the full message body was determined by the Content-Length HTTP header field	Transmission times may differ from those encountered in a real network environment. This can lead to connection closures before the full HTTP message is transferred, triggering this rule
116:414	The IPv4 packet has a broadcast destination address	The use of broadcast addresses may seem unusual. This may be due to specific characteristics of the simulation that may trigger this rule
116:408	The IPv4 packet's source address is from the 'current net' (value of zero)	INetSIm may generate packets with non-standard or particular source IP addresses, including addresses with zero values
112:1	ARP request is unicast, not broadcast	This rule is not specific to simulation but can be ignored

Table 4.1: Snort rules added to the whitelist

Finally, it is important to build a whitelist of IP addresses that are accessed for routine tasks. For this whitelist, it should be noted that our project has a significant advantage. Indeed, when an application on the analyzed phone makes a DNS request, it will constantly have as a response the IP address of the Raspberry Pi 192.168.4.1. Therefore, if the IP address of the C2 server is hardcoded into the malware, then it will be different from 192.168.4.1. So we know that if the destination IP address of a packet is different from 192.168.4.1, it means that it is hardcoded in the phone application. However, there are some classic apps like WhatsApp or Facebook which also has hardcoded IP addresses. To avoid marking as suspicious theses IP addresses, we connect a few phones to our access point and execute the following commands to extract legitimate IP addresses.

```
kali@kali:~$ sudo tcpdump -i wlan0 -w sniff.pcap
kali@kali:~$ tshark -r sniff.pcap -T fields -e ip.dst | sort | uniq >
 IPWhitelist.txt
```

Listing 4.5: Commands to generate properly the payload

4.5.2 Image replication for Raspberry Pi deployment

As this project is intended to be replicated on multiple Raspberry Pi for deployment purposes, it needs to be installed quickly. This objective can be achieved by generating an image of the microSD card, which can subsequently be loaded onto fresh microSD cards. The image on the microSD card can be downloaded from the cylab cloud[9]. It can then be easily loaded onto a new microSD card using the Raspberry Pi Imager software.

4.5.3 Failed attempts and lesson learned

Various areas of research were pursued during the course of this work, which proved to be fruitless or even impossible. This subsection describes some of the functionalities that have been considered to improve the project. Unfortunately, these innovations were unsuccessful for various reasons. The aim of this subsection is to inform a future developer taking over this project of the avenues that should no longer be considered. It also highlights all the thinking that that has been made and that could not succeed.

Evil twin

A first idea was to create an evil twin that would answer all the probes requests of the mobile device. An evil twin is an attack that replicates the access point the victim is connected to. Once the twin access point is created, the attacker sends desauthentication packets to the victim to log out of the legitimate access point. Once disconnected, the client goes into its settings to reconnect to the access point and thus connects to the false access point. From there, an attacker finds himself between the victim and the Internet to capture all the traffic of the victim. This classic attack inspired 2 possibilities of integration into the project. The first possibility was to retrieve probe requests, extract as much information about them as possible in order to create a twin access point. From this twin access point, the idea was to let the phone automatically connect to the fake access point. Unfortunately, even on mobile devices, user interaction is required to connect to the twin access point when the access point is not public. We can still note that if the mobile device sends probe requests corresponding to unsecured Wi-Fi, the idea would be functional. The second possibility was to implement the KARMA attack. As described in the [63] paper, the idea of the karma attack is to respond positively to any probe request by sending a probe response. Unfortunately, this attack is also only available for Wi-Fi hotspots without a password.

Recovering AP BSSIDs from probes

A second innovation idea requested by the promoter of this project was to recover the MAC addresses of the access points known by the mobile device. This would have made it even easier to locate the phone's movements thanks to the WiGLE website[10]. Indeed, the

[9] https://cloud.cylab.be/s/Hx2JenWZL4MMKip
[10] https://wigle.net/

name of the access points are subject to certain changes while this is not the case for MAC addresses. Unfortunately, when looking at the standard of a probe request, the content of the MAC address destination is the broadcast MAC address. This does not allow us to retrieve information about the MAC address of the access point.

4.5.4 Log in to the Raspberry Pi

Resolve conflict between NetworkManager and hostapd configuration

To connect to the Raspberry Pi and interact via a terminal, you can connect via SSH. The purpose of such a connection may be to improve or debug the implementation. It can also be used to interact with the malware. By default, an ssh server is automatically managed on this version of Kali. However, the `NetworkManager` service conflicts with those we have installed. We therefore had to make a slight modification to be able to use SSH and assign it a static IP address for eth0. The purpose of this subsection is to explain the change to basic services. If you only want to log in, you can read the next subsection.

```
kali@kali:~$ sudo printf "[keyfile]\nunmanaged-devices=interface-name
:wlan0\n" | sudo tee /etc/NetworkManager/conf.d/99-unmanaged-devices.
conf
kali@kali:~$ sudo printf "interface eth0\nstatic ip_address
=169.254.18.67/16\n" | sudo tee -a /etc/dhcpcd.conf
```

Listing 4.6: Commands to configure proprely SSH service

Connecting to Raspberry Pi via SSH

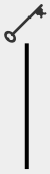
In order to connect to the Raspberry Pi in SSH, password connection is not possible. Instead, you will need to log in with a certificate. In order to activate the certificate connection, it must first be generated on the client side. Then, once the key pair (private key, public key) is created, you need to move the contents of your public key to the `/home/kali/.ssh/authorized_keys` file. Finally it is possible to connect in SSH thanks to your private key. A summary of the commands to be reproduced is below. It is assumed that the client is a Windows computer and that the public key is moved through a USB key.

```
## Client side
C:\Users\John\> ssh-keygen
C:\Users\John\> cd .ssh
C:\Users\John\.ssh\> copy id_rsa.pub D:\

## Raspberry Pi side
kali@kali:~[/media/kali/USB_KEY]$ sudo mv id_rsa.pub /home/kali/.ssh/
authorized_keys
kali@kali:~[/media/kali/USB_KEY]$ sudo systemctl reload ssh

## Client side
C:\Users\John\.ssh\> ssh -i id_rsa kali@169.254.18.67
```

Listing 4.7: Commands to connect to SSH



Shortcut for this subsection

```
C:\Users\John\.ssh\> ssh -i id_rsa kali@169.254.18.67
```

Chapter 5
Case studies & applications

This chapter looks at the program's output in different cases. This output is made up of different files, which will be described later. We therefore study 2 cases. The first is when a soldier suspects his phone is infected with malware. So the phone is placed in the Faraday enclosure and the Raspberry is put into IDS mode. This first case is subdivided into 3 sub-cases representing the 3 different situations that can be faced. Next, we look at a second case in which a phone has been seized from an enemy combatant. In this case, the mobile device is also placed in the Faraday enclosure and we don't have the PIN code. In short, the aim of this chapter is to examine the results of the program in different real-life situations, through the analysis of concrete cases. The purpose is to understand how the system reacts in specific scenarios, using real or simulated data, in order to draw conclusions about its effectiveness, reliability and limitations. It's important to specify that a real Android phone was used for these different cases, not a virtual machine.

5.1 Case study 1: suspected malware infection

When a soldier suspects that his mobile device is infected with malware due to suspicious behaviour, different scenarios can happen. These scenarios include the phone being free of infection, the phone being infected with a basic reverse shell, or the phone being compromised by an Advanced Persistent Threat (APT).

In the first sub-case, where the phone is not infected at all, the program's output will likely show no significant indicators of malicious activity. This outcome would indicate that the soldier's concerns were unfounded, providing reassurance regarding the integrity of the device.

In the second sub-case, if the phone is infected by a basic reverse shell, the program's output may reveal evidence of suspicious network traffic. Common indicators might include unexpected outgoing connections or abnormal data transmission patterns. By identifying these IoCs, the program can confirm the presence of a basic malware infection.

The third sub-case involves the phone being infected by an Advanced Persistent Threat (APT), a sophisticated and sneaky form of malware often associated with nation-state actors or well-resourced cybercriminal groups. Detecting an APT infection presents a greater challenge due to its advanced evasion techniques and ability to remain undetected for extended periods

Side note

In each of these different sub-cases, the program will save 4 files on the USB key.

1. **alert_fast.txt**: This file contains all the alerts generated by the Snort tool.
2. **sniff.pcap**: This file contains all the network packets generated by the mobile

device.

3. **domain_names.txt**: This file contains all the suspicious domain names requested by the mobile device.

4. **suspicious_IP addresses.txt**: This file contains all the suspicious IP addresses requested by the mobile device.

5.1.1 Mobile device not infected

Once the capture has been made in the Faraday enclosure and the contents of the USB key have been recovered, the Figure 5.1 shows what the key contains. For the example, this capture represents 20 minutes of traffic from an Android phone including only the default applications. The different files that are being analysed in subsection can be found on the cylab cloud[1].

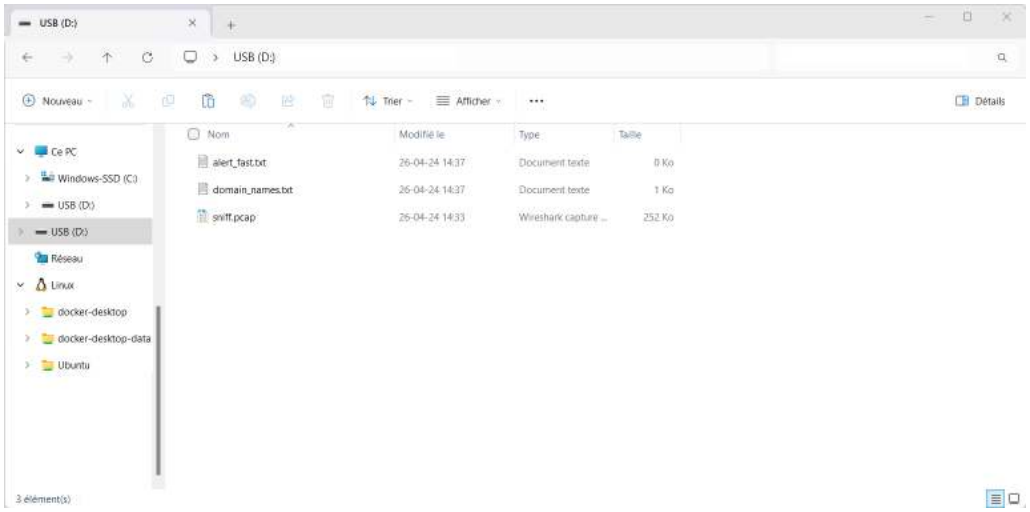


Figure 5.1: Contents of the USB key after capture of the benign case

We can see that the **alert_fast.txt** file is empty. This is expected given that the phone is not infected. So there's no reason for us to have any intrusion alerts from the Snort tool. Also, we can see that the **domain_names.txt** file contains only one domain name `alt2-mtalk.google.com.` which is obviously a false positive. Finally, we can take a quick look at the contents of the Wireshark file.

Thus, we look at the Wireshark capture, we can see that there's nothing in particular, there are no suspicious requests. One important thing to note is that our simulation of Internet services is working well. Firstly, because all the packets from the "Internet" have the IP address of our Raspberry Pi as their source, which means that it is the Raspberry Pi that is responding and not another public address. Secondly, if we look more closely at packets 32 to 51 of the pcap, we can see that INetSim manages to simulate the protocol for establishing a TLS connection between client and server. What's more, once the connection has been established, INetSim manages to respond correctly to requests sent on port 443. In short, this Wireshark capture shows that it is possible for a forensic analyst to investigate the course of events in detail. It also shows that the Internet service simulator is working.

[1] https://cloud.cylab.be/s/C5X79MD6Yn7XdiA

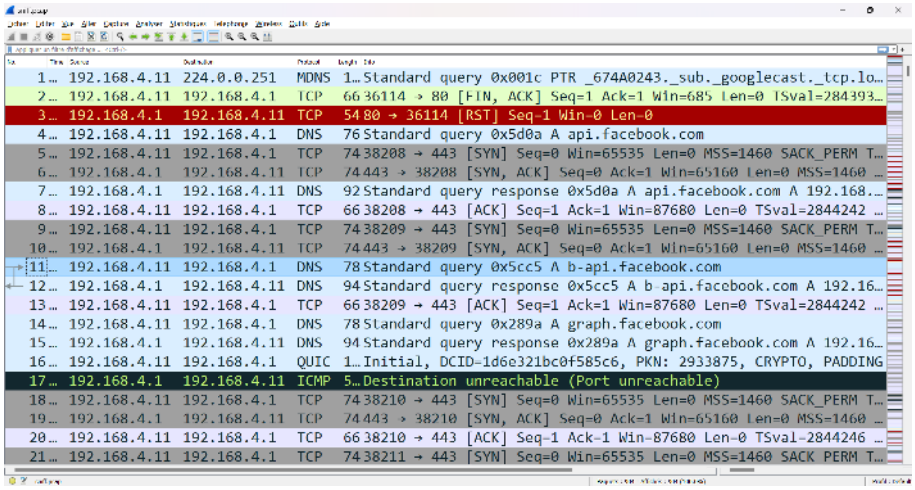


Figure 5.2: Wireshark capture of a benign traffic

5.1.2 Mobile device infected by a basic reverse shell

As before, we analyse the contents of the USB stick. For this example, the capture represents 5 minutes of traffic from an Android phone infected with very basic malware. It is therefore necessary first to explain how this malware was generated in order to better understand the analysis of the contents of the USB key. The malware is crafted using msfvenom[2] which is quite basic, aiming to emulate the typical style of an attack launched by a script kiddie. The different files that are being analysed in sub-section can be found on the cylab cloud[3].

Reverse shell creation

The msfvenom tool that we are using is present by default in Kali Linux. It is used to create payloads that can be operated in the Metasploit framework[4]. These payloads can be used to remotely access systems, exploit vulnerabilities and more. To connect remotely to a victim's machine, we need to enter the IP or domain name of the attacking machine and the port on which we will be listening for incoming connections. All the commands needed to create the reverse shell are available in Listing 5.1. In the following paragraphs, we give a rather formal and complete explanation of the various commands.

In this case, given that our laboratory is local, we will simply use the local address of our attacking machine. Therefore, when we call msfvenom, we'll set the LHOST value to 192.168.129.180. As for the port on which we're going to wait for the connection, just choose a port that isn't currently used. Set the LPORT value to 4444. Additionally, we need to select a payload that corresponds to our operating system. To do this, we can choose a reverse shell tcp for Android: `android/meterpreter/reverse_tcp`. Such a payload allows the compromised machine to initiate a connection back to the attacker's machine. This helps to bypass firewall restrictions and allows the attacker to execute actions remotely. Once the payload has been generated by msfvenom, an attacker can operate it thanks to the Metasploit framework. The capabilities of this payload managed from the framework are wide-ranging, from credential harvesting to lateral movement, as

[2] https://docs.metasploit.com/docs/using-metasploit/basics/how-to-use-msfvenom.html
[3] https://cloud.cylab.be/s/7a3J2G6zraFWoRm
[4] https://docs.metasploit.com/

well as controlling the use of the microphone and webcam remotely.

After creating the payload, it needs to be signed to be executable on an Android device. This stage is different from that for other operating systems. For Android to be able to launch the application, it must be self-signed by the developer. To do this, we can use the keytool[5] tool to generate a certificate and use jarsigner[6] to apply the signature to the generated apk. As regards the use of keytool, we detail the different parameters used. The genkey option is used to specify that a new key pair (private key, public key) should be generated and associated with a self-signed certificate. Next, we use the keystore option to specify the name of the keystore file. Finally, the keyalg, keysize and validity options allow us respectively to use the RSA algorithm for key generation, to define the size of the keys and the validity of the certificate. The key size is expressed as a number of bits and the validity is expressed in terms of days. Finally, to sign the apk, the options used specify the use of SHA-1 with RSA. We also provide the name of the keystore and the name of the application to be signed.

```
kali@kali:~$ sudo msfvenom -platform android -p android/meterpreter/
reverse_tcp LHOST=192.168.129.180 LPORT=4444 R -o android.apk
kali@kali:~$ keytool -genkey -V -keystore key.keystore -alias Android
 -keyalg RSA -keysize 2048 -validity 10000
kali@kali:~$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -
keystore key.keystore android.apk Android
```

Listing 5.1: Commands to generate properly the payload

Output analysis

Once the application has been installed on the phone, simply connect it to the access point and launch the application. The USB key can then be disconnected from the Faraday enclosure and its contents analysed. In Figure 5.3, the **alert_fast.txt** file appears to be empty, and there isn't even a **domain_names.txt** file. The file is missing because the program has not even detected any suspicious domain names. Finally, we have the **sniff.pcap** network packet capture and **suspicious_IP addresses.txt** file.

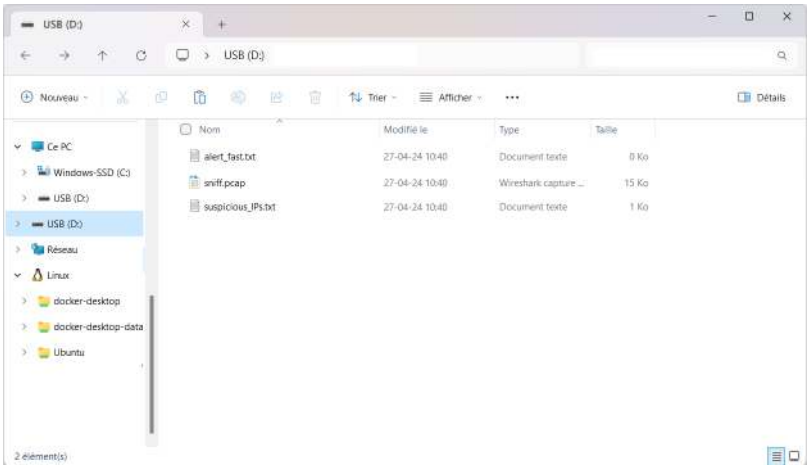


Figure 5.3: Content of the USB key after capture of the reverse shell case

[5] https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html
[6] https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jarsigner.html

Then, if we look at the Wireshark capture, we can see that our malware has tried to open a reverse shell. We can see a set of packets coming from the phone's IP address (192.168.4.11) to the IP address of the attacking machine (192.168.129.180). These packets are equivalent to the malware attempting to connect remotely to the attacking machine. This connection cannot be established because the Raspberry is not connected to the local network. Moreover, if we look at our file of suspicious IP addresses, we can see that an unauthorised IP was called 51 times. In our laboratory environment, this IP does not allow to be sure that the phone is infected. However, in the case of a real threat, we can verify that it is not blacklisted thanks to sites like VirusTotal[7].

In short, given that Snort and the domain name analysis were inconclusive, it is necessary to focus on the IP addresses file to determine whether or not the mobile device is infected by malware. That's why it's so important to capture any suspicious IP addresses requested by the phone. This also makes it possible to link these IP addresses with what is in the Wireshark capture. Using a filter, we can then observe all the packets exchanged with this suspect IP address.

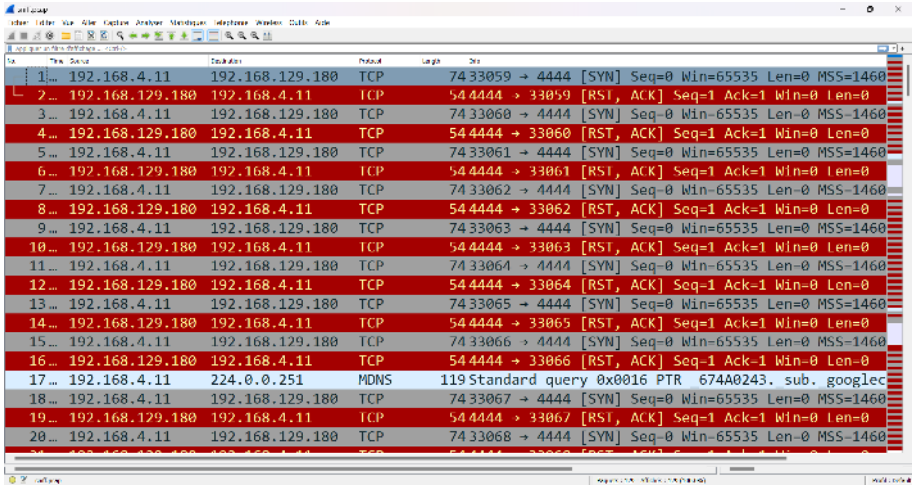


Figure 5.4: Wireshark capture of a reverse shell traffic

5.1.3 Mobile sevice is infected by an advanced persistent threat

In this section the idea is to analyse an android malware known as Safer VPN[8]. Designed to mimic a legitimate VPN service, this malware operates under the guise of providing secure browsing capabilities. Its origins trace back to the APT35 group according to the threat analysis group of Google [21]. In order to assess the effectiveness of the project against such a threat, the idea is to determine what a forensic analyst could conclude. We need to ascertain whether a mobile device is compromised by malware to achieve the project's objectives. Therefore, it's crucial to assess the conclusions drawn from the files generated by our program. These different files that are being analysed in subsection can be found on the cylab cloud[9].

[7] https://www.virustotal.com/
[8] https://bazaar.abuse.ch/sample/5d3ff202f20af915863eee45916412a271bae1ea3a0e20988309c16723ce4da5
[9] https://cloud.cylab.be/s/o9GTJEx4afYpZfn

Output analysis

As before, once the malware has been installed on the phone, all we have to do is connect it to the access point and launch the fake application. As seen in Figure 5.5, the **alert_fast.txt** file is empty. However, the file **domain_names.txt** is not empty and contains interesting artifacts. Finally, we have the usual **sniff.pcap** file containing packets captured on the network.

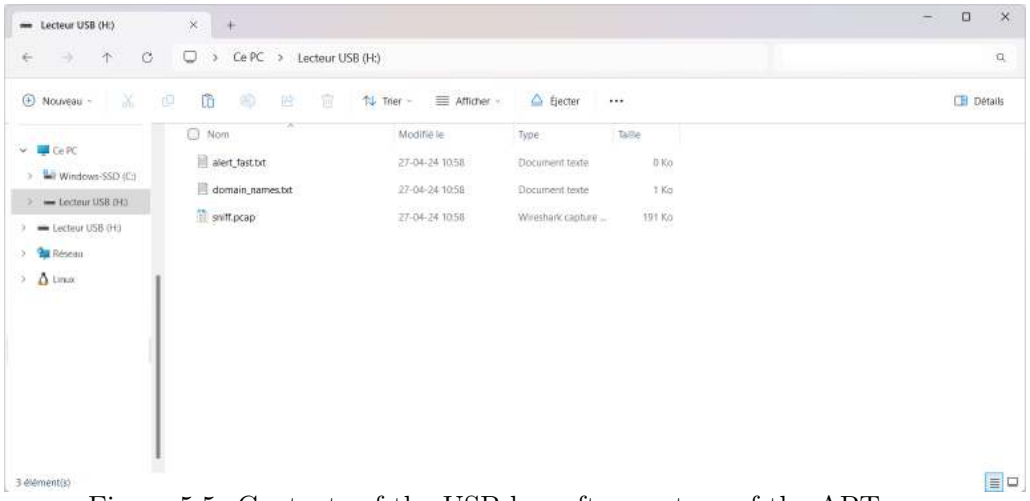


Figure 5.5: Contents of the USB key after capture of the APT case

Therefore, we can ignore the **alert_fast.txt** file since the file is empty. The interesting one is **domain_names.txt** which contains two suspect domain names: cdsa.xyz. and westernrefrigerator.xyz.. These are certainly the domain names of the C&C servers. So if the digital forensic analyst checks domain names on sites such as VirusTotal[10] or URLHause[11] to assess whether the mobile device is infected or not. From examining the results of these sites, one can swiftly determine whether the phone is infected or not.

Finally, if we look at what is contained in the Wireshark capture, we also find very interesting information in the HTTP frames. Moreover, it is also possible to establish a link with the domain names collected in the other file. The capture reveals that package 90 includes a DNS query to the cdsa.xyz domain name. Then, from package 92 we can see a set of requests to an API that can surely extract information. A detail of the URLs is provided below to understand what is extracted. In addition, we can also see in packet 504 a DNS request to our other suspicious domain name westernrefrigerator.xyz. followed by OpenVPN packets. We can therefore assume that this domain name can simulate a VPN connection. This also makes it possible to redirect all the traffic of the phone when this fake VPN is activated.

- `/Api/IsRunAudioRecorder` surely claims that it is possible to activate the audio recorder on the phone. MAC=PPR1.180610.011 is present in the body of the request. PPR1.180610.011 is an Android firmware version code, indicating a specific version of the operating system for an Android phone. Surely allowing the C&C server to uniquely identify the phone.
- `/Api/IsRunClipboard` surely indicated that it is possible to retrieve information

[10] https://www.virustotal.com/gui/home/url
[11] https://urlhaus.abuse.ch/

from the clipboard. The body of the request also contains the MAC address of the phone.

- `/Api/IsRunGPS` surely indicates that the phone has a GPS identification function. Once again the MAC address is sent in the request body.
- `/Api/AndroidTargetLog` is a URL that is called only once. The body of this request still contains several interesting parameters that we detail.
  - "MAC=48:27:EA:BE:7A:0D" It is the MAC address of the device.
  - "Log=Fail to turn on Location because Turn on Location Permission Denied." A log message indicating that the attempt to enable location services failed due to permission denial.
  - "ModuleName=Cell Info Location" The name of the module or feature attempting to enable location services.
- `/Api/Session` is a URL that is called on a regular basis to safely maintain a session with the C&C server. This POST request contains a long string that can be detailed.
  - "PPR1.180610.011" It is once again the Android firmware version code allowing to uniquely identify the infected machine.
  - "Android SDK: 28 (9)" Indicates the SDK version as well as the Android version.
  - "SaferVPN" It is the name of the APK. Its purpose is surely to trick a forensic analyst into making it legitimate.
  - "(0,0)" It is a string that contains information whose purpose is difficult to understand.

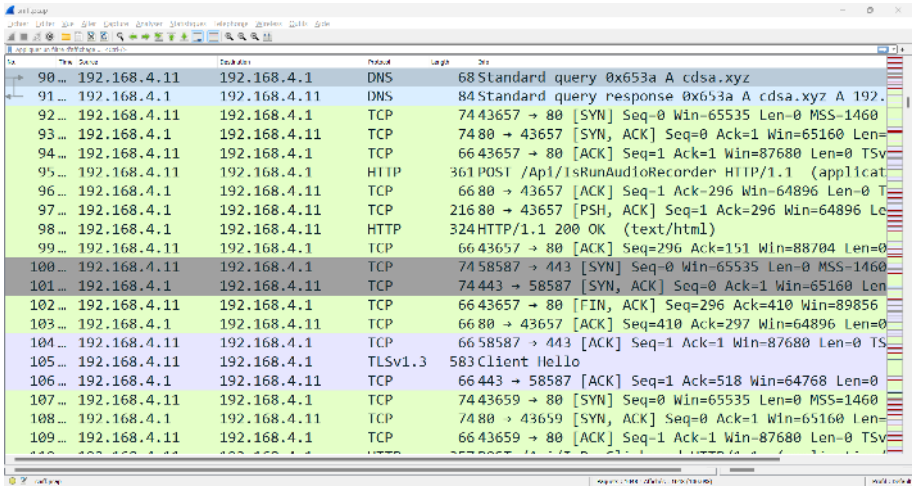


Figure 5.6: Wireshark capture of an APT

In conclusion, it becomes evident that a forensic analyst could have swiftly detected the malware infection on the mobile device. Indeed, simply opening the file containing the domain names and uploading them to VirusTotal would reveal the suspicious nature of these domains. The objective explained at the beginning of this project is therefore fulfilled. In a sensitive military environment, it is possible to detect if a phone is infected with malware. In addition, thanks to Wireshark capture, it is possible to see what the phone seeks to extract as information if the traffic is not encrypted. In addition, in this

case, it is possible to infer what information the malware might have extracted before being placed in our isolated environment. The fact that the malware seeks to know the location of the phone justifies the importance of the Faraday enclosure and the need to work in an isolated environment. However, it's important to acknowledge that despite the productive output of this malware analysis, there are aspects we haven't fully considered. Primarily, we lack access to the information gathered from a static or dynamic analysis that would be executed directly on the mobile device. This means we may we must not ignore the fact that there are other malware that could be able to escape our detection capabilities.

5.2 Summary table of different malware analysed

Once the analysis of different possible behaviours of the product in use, it is interesting to analyse its effectiveness on more malware. Indeed, even if we could analyse 3 different sub-cases, it is interesting to see if the product indeed detects more mobile malware. To do so, the idea is to test recent malware that can be found on MalwareBazaar[12] and install them on our mobile test device. For budgetary reasons, the malware tested are only Android malware. However, given that we mainly analyze the network, we can assume that the network packets generated by an iOS device are similar. The way the table looks is quite simple. The first column is the year the malware was first seen. Next, the second column contains the name of the malware family tested. If the family name was not registered in the database then the apk name is filled in. The third column states whether or not the malware was able to extract IoCs. Subsequently, the "How detected" column indicates which IoC was found. Finally, the last column gives the md5 fingerprint of the malware.

Side note

It is important to note that all files that were generated by the project are available on the cylab cloud[a]. The folder name matches the malware name and the md5 hash of the malware.

[a] https://cloud.cylab.be/s/bkBYj3ratCt6B7b

Table 5.1: Different malware analysed

Year	Mobile malware name	Detected	How detected			MD5 fingerprint
			IP	Domain name	Snort	
2021	Google_Play_Installer.apk	No				dd4596cf68c85eb135f7e0ad763e5dab
2021	WIFI.apk	No				79ba96848428337e685e10b06ccc1c89
2022	Ermac	Yes	✓			35e91deffa2d5392c8d0afa3e83db6a9
2022	IRATA	Yes		✓		ce41d55ee66d509e1e2043d9e238f65a
2023	APT-35	Yes		✓		8a847b0f466b3174741aac734989aa73
2023	tv-latest.apk	Yes	✓			fe9a004870ead6f94ef1a2e09cd6a96a
2023	HookBot	Yes		✓		4002974dbb249748602fe1b4b9da5609
2024	TiSpy	Yes	✓			ce48f58dbae28bcb25677b8add0ddf64
2024	Anubis	Yes	✓			18a3c09ce58b3db05cf248730adb6bd0

[12] https://bazaar.abuse.ch

5.3 Case study 2: seizure of mobile device in sensitive environment

When a mobile device is captured from an enemy installation, it is desirable to recover interesting artifacts. In such a situation, we don't have the phone's PIN code. So we simply place the phone in the Faraday enclosure and set the program to probes mode. We then retrieve the probes requests from the phone. In this case, there are not several possible cases. This section therefore only presents one case. Upon examining the contents of the USB key, we find a file named **probes.txt**, which, in our instance, contains a single line. This line contains the name of the access point requested and also the MAC address of the telephone. Once we have found the name of the access point, we can use the WiGLE[13] website to check if it has been referenced. This makes it possible to track the phone's movements. Obviously, for privacy reasons, the SSID in the file is not a real one. However, the WiGLE website shows that there are network names in Belgium very similar to this one. So it's worth testing the product yourself to see how effective this mode is. The different files that are being analysed in subsection can be found on the cylab cloud[14].

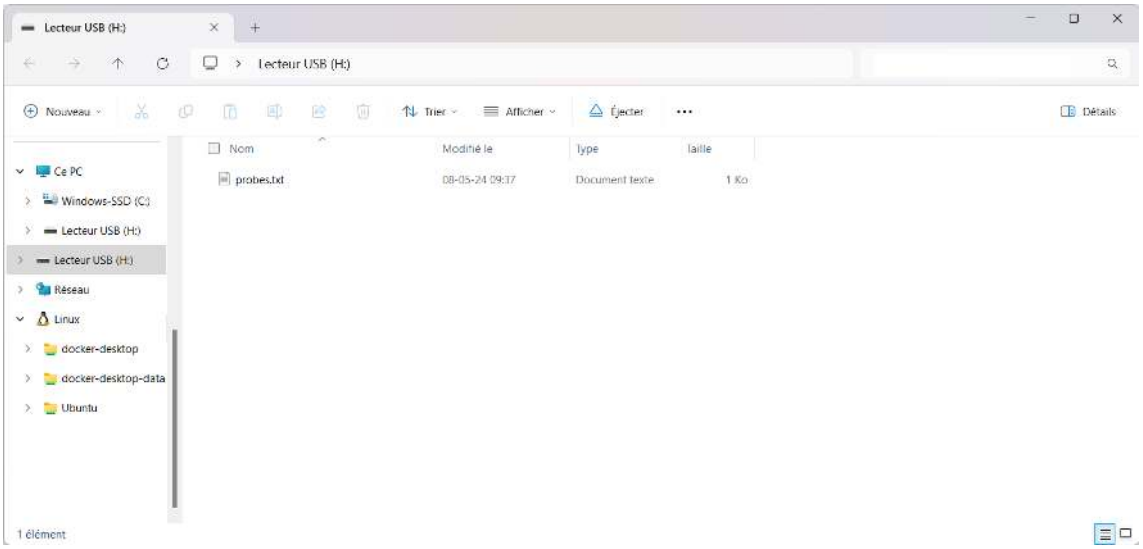


Figure 5.7: Probes captured on USB key

5.4 Lessons learned and best practices

The purpose of this subsection is to sum up key insights and best practices about the efficiency of the product. First, we will assess the effectiveness of malware detection and explore potential improvements. Next, we will address the shortcomings observed with Snort's performance and understand the reasons behind this ineffectiveness. Lastly, we will highlight the differences between our test environment and real-world conditions, highlighting their implications. These slight differences are important to highlight.

In order to evaluate the effectiveness of the product, we can look at the Table 5.1and see that we can easily detect 7 malware out of 9 thanks to the project. At least, our project makes it possible to highlight IoCs significantly helping the work of the forensic

[13] https://wigle.net
[14] https://cloud.cylab.be/s/9ZoxAPmBb43H96E

digital analyst. Indeed, he could simply look at the pcap file but this task would be rather tedious. The second thing that can be noted from this Table is that for the first two malware samples, no IP or domain name was identified. Looking more closely at the capture of these 2 malware, it is difficult to estimate whether the malware have extracted nothing or if they have extracted data in a way that we cannot detect. We therefore understand the importance of providing pcap to an analyst who will surely have more experience in network patterns.

The third thing we can note is that the table clearly highlights that Snort did not work, we must understand why. First, the Snort configuration was tested to verify that the tool was installed. And indeed, if we add a custom rule that detects all ping and run it again on our pcap, the ping is detected. It is therefore necessary to look at some rules of the community to understand when they are activated and why they have not been activated here.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 53 ( msg:"OS-LINUX x86 Linux
 overflow attempt ADMv2"; flow:to_server,established; content:"|89 F7
 29 C7 89 F3 89 F9 89 F2 AC|<|FE|",fast_pattern,nocase; metadata:
ruleset community; service:dns; classtype:attempted-admin; sid:265;
rev:16; )

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS ( msg:"OS-MOBILE
 Android/Fakelash.A!tr.spy trojan command and control channel traffic
"; flow:to_server,established; http_uri; content:"/data.php?action=",
nocase; content:"&m=",distance 0,nocase; content:"&p=",distance 0,
nocase; content:"&n=",distance 0,nocase; metadata:policy max-detect-
ips drop,ruleset community; service:http; reference:url,blog.
fortiguard.com/android-malware-distributed-by-malicious-sms-in-france
/; classtype:trojan-activity; sid:24251; rev:5; )
```

Listing 5.2: Snort community rules example

The first Snort rule detects an attempt to exploit a vulnerability on x86 Linux on port 53. It looks for a specific hexadecimal entry in order to trigger an alert. This rule does not apply to our case because we are only interested in mobile malware. It can be noted that in the community rules, a lot of rules concern Windows or Linux systems that only slow down the analysis of our pcap. Scraping them off might be a good idea. The second rule is much more interesting as it allows us to note an Android malware that we have not encountered in the Table 5.1. Snort can therefore detect some Android malware across the network. Specifically, this rule can detect if the malware has attempted to extract data at the specific URL `/data.php?action=`. Among the malware tested in Section 5.2, none triggered Snort rules. We can therefore conclude from here that Snort is not poorly implemented or non-functional for our context but that it is not fully adapted to mobile malware. The ideal would be to have a feed of Snort rules for recent mobile malware.

In addition, another important thing that can be noted about our testing environment is its limit to adapt to the real case. It should still be noted that in the real case of product use, a much larger number of applications will generate traffic. This will significantly increase the size of the various output files. However, whitelists help significantly reduce the size of these files. We can therefore recommend to disable applications that are not

suspicious on the phone in order to reduce the network traffic generated.

In conclusion, the different results of the product show that it meets its objectives. Nevertheless, the community's Snort rules are not adapted to the recent mobile malware and require a manual update. Snort therefore does not meet its share of efficiency in identifying network artifacts. Fortunately, Snort can be easily updated given the simplicity of writing rules. Also, using the product in a real case will generate much more traffic. It is therefore advisable to disable unsuspecting applications.

Chapter 6
Future work

It will be important that future research investigate practical considerations for time to pack, host artifacts and Snort customisation. Time to pack is important in the military context to achieve objectives. Host artifacts are of paramount importance in dealing with the most advanced threats. Finally, customising Snort could significantly increases the detection of threats by the network.

First of all, time to pack in the military environment describes the time given to the soldiers during which they organise their equipment in preparation for deployment or any operational activity. This concept has crucial implications in terms of efficiency, organisation, logistics and mission success. It is therefore crucial to consider this idea in the future work. For now, the physical product is still in the state of "Proof of concept" being connected to a breadboard and fairly basic components. In order to improve product mobility, it is necessary to solder the cables directly to the Raspberry Pi as well as secure it inside the Faraday enclosure. In this way, digital forensic analysts in the field will just have to take the Faraday enclosure to the deployment location. Once there, they simply put the phone in the Faraday enclosure and perform the analysis. Finally, it can be noted that plugging in a USB display and developing a GUI could increase the usability of the product.

Then, it would be interesting to couple the malware detection with an investigation of artifacts on the host. This would allow a more complete analysis and thus assess with more assurance whether the mobile device is infected with a malware. Indeed, for now, if a malware secretly exfiltrates data through encrypted traffic, it remains difficult, even for an advanced forensic digital analyst, to see it in a Wireshark capture. We highlighted during the state of the art and the Section 3.3.1 the importance of these artifacts of the host. Fortunately, it is indeed possible to search for such artifacts. This can be done through HIDS or malware analysis. It is possible to connect a USB cable to the Faraday enclosure. We therefore highlight the possibility of connecting to the phone in USB and investigate the phone through ADB as well as various analysis tools such as Frida, Objection, etc. On the other hand, there may be data exfiltration by cable.

Finally, as highlighted in the conclusion of Chapter 5, Snort is not adapted to current mobile threats. Nevertheless, Snort takes advantage of the possibility to integrate custom rules. Future research should be devoted to the development of new rules for mobile malware detection in the form of Snort rules. This could potentially identify patterns of encrypted network exfiltration of some advanced malware. The search for such a domain would not only improve the detection of malware in our context but the detection of mobile malware in general.

Chapter 7
Conclusion

The aim of this master thesis was to deal with a series of situations that a digital forensics team in a military context might encounter. Identifying malware infections on mobile devices in the dynamic and resource-constrained military environment was previously challenging. In order to address this challenge, we set up a Faraday enclosure, a portable intrusion detection system and finally the ability to capture probe requests from locked phones seized from an enemy installation. Importantly, our results prove that recent Advanced Persistent Threats can be detected by the developed solution. Therefore, our findings on malware detection in an isolated environment are broadly consistent with the current threats that can be encountered in the field.

The research began with a comprehensive literature review on mobile device security, malware detection and data acquisition for locked phones. After identifying the various existing tools, it was necessary to make a selection and justify the usefulness of these choices using widely-used frameworks. Next, the use and implementation phase provides a comprehensive guide for both the user and the developer. Finally, our findings have demonstrated the effectiveness of network-based malware detection in an isolated environment. This is achieved by highlighting suspicious domain names and IP addresses and by logging network traffic. Snort can also be useful in very specific cases of some malware, although not optimised for mobile malware.

Based on the results of this study, it is recommended that the artifacts left on the host should not be overlooked. Indeed, it was highlighted in the literature review that it is necessary to analyse the artefacts left on the host to be sure of detecting malware. In addition, the results of Chapter 5 showed that not all malware could be found by our product. Practitioners in the field or future developers should therefore focus on implementing a hybrid approach based on both network IoCs and those that may be present on the host.

In conclusion, in order to combat today's threats effectively in a military context, a hybrid approach is more than important. However, our work highlights the effectiveness and potential of network-based threat detection. It is therefore vital to continue innovating in digital investigation methods against advanced threats in dynamic environments such as the military.

Bibliography

[1] Adware - What is Adware and How to Remove Adware — malwarebytes.com. https://www.malwarebytes.com/adware, [Accessed 12-05-2024] [Cited on page 6.]

[2] Data Exfiltration Techniques - YouTube, https://youtube.com/playlist?list=PLqM63j87R5p4cHSmz5UcrujWA64ap9DNW&si=vi9SyefQXJtFTmHc [Cited on page 10.]

[3] Exfiltration, Tactic TA0010 - Enterprise | MITRE ATT&CK®, https://attack.mitre.org/tactics/TA0010/, [Accessed 19-11-2023] [Cited on page 10.]

[4] New RedDrop Android Spyware Records Nearby Audio, https://www.bleepingcomputer.com/news/security/new-reddrop-android-spyware-records-nearby-audio/ [Cited on page 9.]

[5] Trojan Horse Virus | Trojan Horse Malware | What is a Trojan Virus — malwarebytes.com. https://www.malwarebytes.com/trojan, [Accessed 12-05-2024] [Cited on page 6.]

[6] What is a Computer Worm? | Malwarebytes — malwarebytes.com. https://www.malwarebytes.com/computer-worm, [Accessed 12-05-2024] [Cited on page 6.]

[7] What is threat hunting? | IBM, https://www.ibm.com/topics/threat-hunting [Cited on page 16.]

[8] How to analyze mobile malware: a Cabassous/FluBot Case study (Apr 2021), https://blog.nviso.eu/2021/04/19/how-to-analyze-mobile-malware-a-cabassous-flubot-case-study, [Online; accessed 18. May 2024] [Cited on page 8.]

[9] What Is a Drive by Download (Apr 2023), https://www.kaspersky.com/resource-center/definitions/drive-by-download [Cited on page 7.]

[10] What is zero-click malware, and how do zero-click attacks work? (Apr 2023), https://usa.kaspersky.com/resource-center/definitions/what-is-zero-click-malware, [Online; accessed 18. May 2024] [Cited on page 8.]

[11] APE: Intrusion Protection System for Android Devices | MITRE (May 2024), https://www.mitre.org/our-impact/intellectual-property/ape-intrusion-protection-system-android-devices, [Online; accessed 19. May 2024] [Cited on page 17.]

[12] Bypassing Certificate Pinning - OWASP Mobile Application Security (May 2024), https://mas.owasp.org/MASTG/techniques/android/MASTG-TECH-0012, [Online; accessed 19. May 2024] [Cited on page 21.]

[13] Command and Control, Tactic TA0011 - Enterprise | MITRE ATT&CK® (May 2024), https://attack.mitre.org/tactics/TA0011, [Online; accessed 18. May 2024] [Cited on page 9.]

[14] Exfiltration Over C2 Channel, Technique T1646 - Mobile | MITRE ATT&CK® (May 2024), https://attack.mitre.org/techniques/T1646, [Online; accessed 18. May 2024] [Cited on page 12.]

[15] Matrix - Mobile | MITRE ATT&CK® (May 2024), https://attack.mitre.org/matrices/mobile [Cited on page 10.]

[16] RedDrop, Software S0326 | MITRE ATT&CK® (May 2024), https://attack.mitre.org/software/S0326, [Online; accessed 18. May 2024] [Cited on page 9.]

[17] Supply Chain Compromise, Technique T1474 - Mobile | MITRE ATT&CK® (May 2024), https://attack.mitre.org/techniques/T1474, [Online; accessed 18. May 2024] [Cited on page 8.]

[18] The Dangers of Third-Party App Stores: Risks and Precautions - SOCRadar® Cyber Intelligence Inc. (Mar 2024), https://socradar.io/the-dangers-of-third-party-app-stores-risks-and-precautions, [Online; accessed 17. May 2024] [Cited on page 7.]

[19] What are indicators of compromise (IoC)? (May 2024), https://www.cloudflare.com/learning/security/what-are-indicators-of-compromise, [Online; accessed 19. May 2024] [Cited on pages 13 and 25.]

[20] America's Cyber Defense Agency: Best Practices for MITRE ATT&CK Mapping. https://www.cisa.gov/sites/default/files/publications/BestPracticesforMITREATTCKMapping.pdf, [Accessed 12-05-2024] [Cited on page 10.]

[21] Ajax Bash, T.A.G.: Countering threats from Iran — blog.google. https://blog.google/threat-analysis-group/countering-threats-iran/ (2021), [Accessed 07-05-2024] [Cited on page 52.]

[22] Allen, J., Yang, Z., Landen, M., Bhat, R., Grover, H., Chang, A., Ji, Y., Perdisci, R., Lee, W.: Mnemosyne: An effective and efficient postmortem watering hole attack investigation system. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. p. 787–802. CCS '20, Association for Computing Machinery, New York, NY, USA (2020) [Cited on page 7.]

[23] Aman_Utkhedkar: Data Exfiltration with Discord. (Nov 2022), https://medium.com/@rootxaman/data-exfiltration-with-discord-98139e8c6590 [Cited on page 11.]

[24] Arora, A., Peddoju, S.K.: Minimizing network traffic features for android mobile malware detection. In: Proceedings of the 18th international conference on distributed computing and networking. pp. 1–10 (2017) [Cited on page 18.]

[25] Aviv, A.J., Gibson, K., Mossop, E., Blaze, M., Smith, J.M.: Smudge attacks on smartphone touch screens. In: 4th USENIX workshop on offensive technologies (WOOT 10) (2010) [Cited on page 23.]

[26] Axelsson, S.: The base-rate fallacy and its implications for the difficulty of intrusion detection. p. 1–7. CCS '99, Association for Computing Machinery, New York, NY, USA (1999) [Cited on page 26.]

[27] Ayers, R., Brothers, S., Jansen, W.: Guidelines on mobile device forensics (May 2014) [Cited on page 23.]

[28] Barbera, M.V., Epasto, A., Mei, A., Perta, V.C., Stefa, J.: Signals from the crowd: uncovering social relationships through smartphone probes. In: Proceedings of the 2013 conference on Internet measurement conference. pp. 265–276 (2013) [Cited on page 18.]

[29] Barmpatsalou, K., Cruz, T., Monteiro, E., Simoes, P.: Current and future trends in mobile device forensics: A survey. ACM Computing Surveys 51(3), 1–31 (May 2018) [Cited on page 23.]

[30] Beckers, J.: Proxying Android app traffic – Common issues / checklist (2024) (Feb 2024), https://blog.nviso.eu/2020/11/19/proxying-android-app-traffic-common-issues-checklist, [Online; accessed 19. May 2024] [Cited on page 21.]

[31] Beek, C.: Apply MITRE's 'ATT&CK' Model to Check Your Defenses. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/apply-mitres-attck-model-to-check-your-defenses/ (2018), [Accessed 12-05-2024] [Cited on page 10.]

[32] Bianco, D.J.: The Pyramid of Pain (Jan 2014), http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html, [Online; accessed 20. May 2024] [Cited on page 27.]

[33] Breeuwsma, M.: Forensic imaging of embedded systems using jtag (boundary-scan). digital investigation 3(1), 32–42 (2006) [Cited on page 23.]

[34] Chen, P., Desmet, L., Huygens, C.: A study on advanced persistent threats. In: De Decker, B., Zúquete, A. (eds.) Communications and Multimedia Security. pp. 63–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2014) [Cited on page 9.]

[35] Chris Wysopal, C.E.: Static detection of application backdoors (2015) [Cited on page 5.]

[36] Conti, M., Li, Q.Q., Maragno, A., Spolaor, R.: The Dark Side(-Channel) of Mobile Devices: A Survey on Network Traffic Analysis. IEEE Communications Surveys & Tutorials 20(4), 2658–2713 (2018) [Cited on pages 17 and 19.]

[37] Di Luzio, A., Mei, A., Stefa, J.: Mind your probes: De-anonymization of large crowds through smartphone wifi probe requests. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications. pp. 1–9 (2016) [Cited on page 24.]

[38] Do, Q., Martini, B., Choo, K.K.R.: Exfiltrating data from Android devices. Computers & Security 48, 74–91 (Feb 2015) [Cited on pages 10 and 11.]

[39] D'Orazio, C.J., Choo, K.K.R., Yang, L.T.: Data exfiltration from internet of things devices: ios devices as case studies. IEEE Internet of Things Journal 4(2), 524–535 (2017) [Cited on page 10.]

[40] Enterprise, B.: The Differences Between Static and Dynamic Malware Analysis (May 2024), https://www.bitdefender.com/blog/businessinsights/the-differences-between-static-malware-analysis-and-dynamic- malware-analysis [Cited on page 16.]

[41] Eslahi, M., Naseri, M.V., Hashim, H., Tahir, N., Saad, E.H.M.: Byod: Current state and security challenges. In: 2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE). pp. 189–192 (2014) [Cited on page 4.]

[42] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M., Rajarajan, M.: Android security: A survey of issues, malware penetration, and defenses. IEEE Communications Surveys & Tutorials 17(2), 998–1022 (2015) [Cited on page 9.]

[43] Feng, J., Shen, L., Chen, Z., Wang, Y., Li, H.: A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic. IEEE Access 8, 125786–125796 (2020) [Cited on page 19.]

[44] Freudiger, J.: How talkative is your mobile device? an experimental study of wi-fi probe requests. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks. WiSec '15, Association for Computing Machinery, New York, NY, USA (2015) [Cited on page 24.]

[45] Ganacharya, T.: Microsoft 365 Defender demonstrates 100 percent protection coverage in the 2023 MITRE Engenuity ATT&CK® Evaluations: Enterprise | Microsoft Security Blog — microsoft.com (2021), https://www.microsoft.com/en-us/security/blog/2023/09/20/microsoft-365-defender-demonstrates-100-percent-protection-coverage-in-the-2023-mitre-engenuity-attck-evaluations-enterprise/ [Cited on page 10.]

[46] Ghorbanian, M., Shanmugam, B., Narayansamy, G., Idris, N.B.: Signature-based hybrid intrusion detection system (hids) for android devices. In: 2013 IEEE Business Engineering and Industrial Applications Colloquium (BEIAC). pp. 827–831 (2013) [Cited on page 17.]

[47] González-Granadillo, G., González-Zarzosa, S., Diaz, R.: Security information and event management (siem): Analysis, trends, and usage in critical infrastructures. Sensors 21(14) (2021) [Cited on page 16.]

[48] Goodin, D.: Legit app in Google Play turns malicious and sends mic recordings every 15 minutes (May 2023), https://arstechnica.com/information-technology/2023/05/app-with-50000-google-play-installs-sent-attackers-mic-recordings-every-15-minutes [Cited on page 6.]

[49] Gorecki, C., Freiling, F.C., Kührer, M., Holz, T.: TrumanBox: Improving Dynamic Malware Analysis by Emulating the Internet. In: Défago, X., Petit, F., Villain, V. (eds.) Stabilization, Safety, and Security of Distributed Systems. pp. 208–222. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2011) [Cited on page 22.]

[50] Gu, X., Wu, W., Gu, X., Ling, Z., Yang, M., Song, A.: Probe Request Based Device Identification Attack and Defense. Sensors 20(16), 4620 (Aug 2020) [Cited on page 24.]

[51] Guri, M., Kachlon, A., Hasson, O., Kedma, G., Mirsky, Y., Elovici, Y.: GSMem: Data exfiltration from Air-Gapped computers over GSM frequencies. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 849–864. USENIX Association, Washington, D.C. (Aug 2015) [Cited on page 12.]

[52] Haataja, K.: Security Threats and Countermeasures in Bluetooth-Enabled Systems (Turvallisuusuhkat ja vastatoimet Bluetooth-yhteensopivissa järjestelmissä). Ph.D. thesis, Kuopion yliopisto (2009) [Cited on page 7.]

[53] Harkin, D., Molnar, A., Vowles, E.: The commodification of mobile phone surveillance: An analysis of the consumer spyware industry. Crime, Media, Culture 16(1), 33–60 (2020) [Cited on page 5.]

[54] Intelligence, M.T.: Toll fraud malware: How an Android application can drain your wallet | Microsoft Security Blog. Microsoft Security Blog (Sep 2023) [Cited on page 9.]

[55] Jakobsson, M., Myers, S.: Phishing and countermeasures: understanding the increasing problem of electronic identity theft. John Wiley & Sons (2006) [Cited on page 11.]

[56] Johnson, N.F., Jajodia, S.: Exploring steganography: Seeing the unseen. Computer 31(2), 26–34 (1998) [Cited on page 12.]

[57] Kalidas, A.: Android-Data-Exfiltration (Aug 2023), https://github.com/Arjunkalidas/Android-Data-Exfiltration, original-date: 2020-03-19T02:41:57Z [Cited on page 10.]

[58] Kaspersky: What is rootkit – definition and explanation (Apr 2023), https://www.kaspersky.com/resource-center/definitions/what-is-rootkit [Cited on page 5.]

[59] Kent, K., Souppaya, M.: Guide to Computer Security Log Management. Tech. Rep. NIST Special Publication (SP) 800-92, National Institute of Standards and Technology (Sep 2006) [Cited on page 15.]

[60] Kharraz, A., Kirda, E., Robertson, W., Balzarotti, D., Francillon, A.: Optical delusions: A study of malicious qr codes in the wild. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 192–203. IEEE (2014) [Cited on page 8.]

[61] Kral, P.: The incident handlers handbook. Sans Institute (2011) [Cited on page 6.]

[62] La Polla, M., Martinelli, F., Sgandurra, D.: A survey on security for mobile devices. IEEE Communications Surveys & Tutorials 15(1), 446–471 (2013) [Cited on pages 5 and 7.]

[63] Lovinger, N., Gerlich, T., Martinasek, Z., Malina, L.: Detection of wireless fake access points. In: 2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). pp. 113–118 (2020) [Cited on page 45.]

[64] Lunt, T.F.: A survey of intrusion detection techniques. Computers & Security 12(4), 405–418 (Jun 1993) [Cited on page 13.]

[65] Makhlouf, A.M., Boudriga, N.: Intrusion and anomaly detection in wireless networks. In: Handbook of Research on Wireless Security, pp. 78–94. IGI Global (2008) [Cited on page 7.]

[66] Marczak, B., Scott-Railton, J., McKune, S.: Hacking team reloaded? us-based ethiopian journalists again targeted with spyware. The Citizen Lab (2015) [Cited on page 5.]

[67] Marczak, B., Scott-Railton, J., McKune, S., Abdul Razzak, B., Deibert, R.: Hide and seek: Tracking nso group's pegasus spyware to operations in 45 countries. Tech. rep. (2018) [Cited on page 8.]

[68] Müller, T., Spreitzenbarth, M.: Frost: Forensic recovery of scrambled telephones. In: Applied Cryptography and Network Security: 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings 11. pp. 373–388. Springer (2013) [Cited on page 23.]

[69] Or-Meir, O., Nissim, N., Elovici, Y., Rokach, L.: Dynamic malware analysis in the modern era—a state of the art survey. ACM Comput. Surv. 52(5) (sep 2019) [Cited on page 16.]

[70] Paxson, V.: Bro: a system for detecting network intruders in real-time. Computer Networks 31(23), 2435–2463 (1999) [Cited on page 20.]

[71] Provos, N., et al.: A virtual honeypot framework. In: USENIX Security Symposium. vol. 173, pp. 1–14 (2004) [Cited on page 15.]

[72] Ribeiro, J., Saghezchi, F.B., Mantas, G., Rodriguez, J., Shepherd, S.J., Abd-Alhameed, R.A.: An Autonomous Host-Based Intrusion Detection System for Android Mobile Devices. Mobile Netw. Appl. 25(1), 164–172 (Feb 2020) [Cited on page 17.]

[73] Ribeiro, J., Saghezchi, F.B., Mantas, G., Rodriguez, J., Abd-Alhameed, R.A.: Hidroid: Prototyping a behavioral host-based intrusion detection and prevention system for android. IEEE Access 8, 23154–23168 (2020) [Cited on page 17.]

[74] Roesch, M.: Snort – Lightweight Intrusion Detection for Networks (1999) [Cited on page 20.]

[75] Ruberg, B.: Small-scale honeynet with Raspberry Pi (Dec 2016), https://www.redpill-linpro.com/techblog/2016/12/19/raspberry-pi-honeynet.html [Cited on page 22.]

[76] Rudie, J., Katz, Z., Kuhbander, S., Bhunia, S.: Technical analysis of the nso group's pegasus spyware. In: 2021 International Conference on Computational Science and Computational Intelligence (CSCI). pp. 747–752 (2021) [Cited on page 8.]

[77] Scarfone, K., Mell, P.: Guide to Intrusion Detection and Prevention Systems (IDPS). Tech. Rep. NIST Special Publication (SP) 800-94, National Institute of Standards and Technology (Feb 2007) [Cited on page 20.]

[78] Search, C.: What is Threat Intelligence and Why is it Important? (Nov 2023), https://www.cyberneticsearch.com/blog/what-is-threat-intelligence-and-why-is-it-important-, [Online; accessed 20. May 2024] [Cited on page 26.]

[79] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: "andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems 38(1), 161–190 (2012) [Cited on page 18.]

[80] Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: 2010 IEEE Symposium on Security and Privacy. pp. 305–316 (2010) [Cited on page 26.]

[81] Stefan: Wifi probe requests explained (Jul 2022), https://blog.spacehuhn.com/probe-request [Cited on page 3.]

[82] Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen, H.: Investigating user privacy in android ad libraries. In: Workshop on Mobile Security Technologies (MoST). vol. 10, pp. 195–197 (2012) [Cited on page 19.]

[83] Su, X., Chuah, M., Tan, G.: Smartphone dual defense protection framework: Detecting malicious applications in android markets. In: 2012 8th International Conference on Mobile Ad-hoc and Sensor Networks (MSN). pp. 153–160. IEEE (2012) [Cited on page 18.]

[84] Titterington, A.: Google Play malware clocks up more than 600 million downloads in 2023. Kaspersky (Nov 2023) [Cited on page 6.]

[85] Tung, L.: Apple: These are the sorts of apps we blocked from our App Store last year, https://www.zdnet.com/article/apple-these-are-the-sorts-of-apps-we-blocked-from-our-app-store-last-year/ [Cited on page 6.]

[86] Watkins, L., Corbett, C., Salazar, B., Fairbanks, K., Robinson, W.H.: Using network traffic to remotely identify the type of applications executing on mobile devices. Johns Hopkins University Applied Physics Laboratory Laurel, MD USA (2013) [Cited on page 19.]

[87] Wim, M.: Pragmatic cybersecurity. Independently published (2020) [Cited on pages 13 and 14.]

[88] Xiang, C., Binxing, F., Lihua, Y., Xiaoyi, L., Tianning, Z.: Andbot: towards advanced mobile botnets. In: 4th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 11) (2011) [Cited on page 9.]

[89] Yadav, T., Rao, A.M.: Technical Aspects of Cyber Kill Chain. In: Abawajy, J.H., Mukherjea, S., Thampi, S.M., Ruiz-Martínez, A. (eds.) Security in Computing and Communications. pp. 438–452. Springer International Publishing, Cham (2015) [Cited on page 6.]

[90] Yasar, K.: command-and-control server (C&C server). WhatIs (Oct 2022) [Cited on page 9.]

[91] Yeboah-Boateng, E.O., Amanor, P.M.: Phishing, smishing & vishing: an assessment of threats against mobile devices. Journal of Emerging Trends in Computing and Information Sciences 5(4), 297–307 (2014) [Cited on page 8.]

[92] Zarni Aung, W.Z.: Permission-based android malware detection. International Journal of Scientific & Technology Research 2(3), 228–234 (2013) [Cited on page 17.]

Appendix A

Source code

A.1 Configuration files

```
interface=wlan0
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
address=/#/192.168.4.1
```

Figure A.1: Configuration file for `dnsmasq.conf`

```
hostname
clientid
persistent
option rapid_commit
option domain_name_servers, domain_name, domain_search, host_name
option classless_static_routes
option interface_mtu
require dhcp_server_identifier
slaac private

interface wlan0
        static ip_address=192.168.4.1/24
        nohook wpa_supplicant

interface eth0
        static ip_address=169.254.18.67/16

```

Figure A.2: Configuration file for `dhcpcd.conf`

```
country_code=BE
interface=wlan0
ssid=MemoryThesis
channel=9
```

Figure A.3: Configuration file for `hostapd.conf`

A.2 Python code

A.3 Tools installation not in details

```
sudo apt install dnsmasq hostapd
sudo nano /etc/dhcpcd.conf
# Modify the configuration file
sudo nano /etc/dnsmasq.conf
# Modify the configuration file
sudo nano /etc/hostapd/hostapd.conf
# Modify the configuration file
sudo nano /etc/default/hostapd
# Indicates where the configuration file is located
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl restart hostapd dnsmasq dhcpcd
sudo apt install inetsim
sudo nano /etc/inetsim/inetsim.conf
# Modify the configuration file
sudo iptables -t nat -A PREROUTING -i wlan0 -p udp --dport 67:68 \
-j ACCEPT
sudo iptables -t nat -A PREROUTING -i wlan0 -j REDIRECT
sudo iptables-save > /etc/iptables/rules.v4
sudo systemctl restart inetsim
sudo systemctl enable inetsim
sudo nano startup
# Create a script that restores iptables rules at boot
sudo crontab -e
# Make the script launched at boot
sudo nano /etc/dnsmasq.conf
# Make all the dns responses containing the address of the Raspberry Pi
sudo systemctl restart dnsmasq
# Install and configure SNORT
sudo apt install snort -y
sudo snort -c /etc/snort/snort.lua
sudo ip link set dev wlan0 promisc on
ip address show wlan0
sudo ethtool -K wlan0 gro off lro off
sudo snort -c /etc/snort/snort.lua -R /etc/snort/rules/local.rules \
-i wlan0 n-A alert_fast -s 65535 -k none -l </path/to/USBKEY>
# To turn on monitor mode
sudo systemctl stop hostapd dhcpcd dnsmasq inetsim
sudo iptables -F
sudo airmon-ng start wlan0
```

Figure A.4: Command sequence to install the tools

Appendix B
Quick Reference Guide

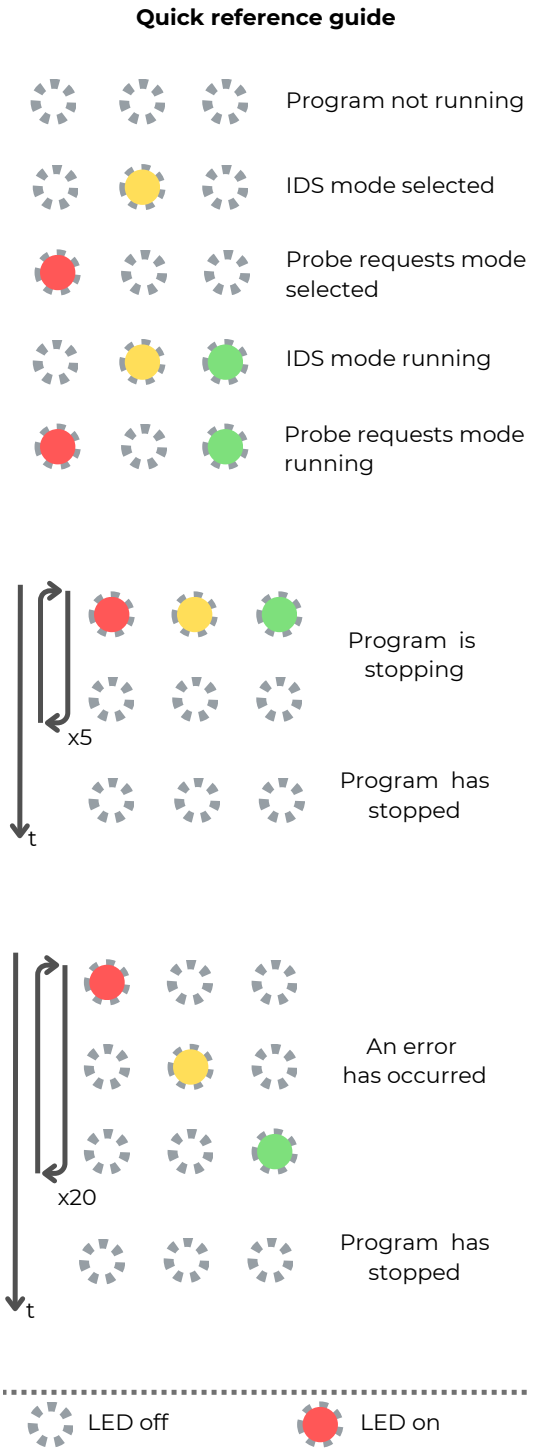


Figure B.1: Quick reference guide to print and display on the Faraday enclosure[1]

[1] https://cloud.cylab.be/s/9tAoemq2nTmmxzK