Mémoire de Projet de Fin d'Études

Pour l'obtention du Titre D'Ingénieur d'État en

Télécommunications et Technologies d'Information

Filière: Cybersécurité et Confiance Numérique

Sous le thème:

Présenté par :

- M. XX

Membre de jury:

- Mme AYACHE Meryeme: Encadrante Interne
- M.XX: Président du Jury
- M. XX: Examinateur
- M. XX: Encadrant Externe

Logo d'entreprise

Année Universitaire 2021-2022

DEDICATIONS

“

hello word

”

- SABRI Siham

ACKNOLEGEMNETS

Remerciement

RÉSUMÉ

Résumé

Mots clés: Cybersecurité, Collecte d'informations, Reconnaissance, Vulnerabilité, Automatisation.

ABSTRACT

Abstract

Keywords : Cybersecurity, Information Gathering, Reconnaissance, Vulnerability, Automation.

Contents

DEDICATIONS 2

ACKNOLEGEMENTS 3

RÉSUMÉ 4

ABSTRACT 5

TABLE OF CONTENTS 7

TABLE OF FIGURES 8

LIST OF ACRONYMES 9

General Introducation 11

1 Host Organization and General Context 12
- 1.1 Introduction . . . 13
- 1.2 Host Organization : HENCEFORTH . . . 13
  - 1.2.1 HENCEFORTH presentation . . . 13
  - 1.2.2 Main activities and Cyber security services of HENCEFORTH . . . 13
  - 1.2.3 Organizational chart . . . 14
- 1.3 General Context . . . 14
  - 1.3.1 Motivation . . . 14
  - 1.3.2 Problem statement . . . 15
  - 1.3.3 Objectives . . . 15
- 1.4 Project Management . . . 16
  - 1.4.1 Working Method . . . 16
  - 1.4.2 Management method and tools . . . 16
  - 1.4.3 Project planning . . . 16
- 1.5 Conclusion . . . 17

2 BACKGROUND 18
- 2.1 **Introduction** . . . 19
- 2.2 **Digital Forensics** . . . 19
  - 2.2.1 Definition . . . 19
  - 2.2.2 Process . . . 19
  - 2.2.3 ? . . . 19
- 2.3 **Android systems** . . . 19
  - 2.3.1 Android architecture . . . 19
  - 2.3.2 Boot processing . . . 22
  - 2.3.3 Storage structure . . . 23

2.3.4 Android applications . . . 24
2.3.5 Security and malware in Android . . . 26
2.4 **Android Malware Analysis** . . . 29
2.4.1 Definition . . . 29
2.4.2 Static analysis and Obfuscation . . . 29
2.4.3 Dynamic analysis and its limits . . . 31
2.4.4 Tools . . . 33
2.4.5 Lab . . . 33
2.5 **Conclusion** . . . 33

3 Android Forensics Process **34**
3.1 Introduction . . . 35
3.2 Android forensics guide and configuration . . . 35
3.2.1 General steps . . . 36
3.2.2 Seizure . . . 36
3.2.3 Extraction . . . 36
3.2.4 Analysis . . . 36
3.2.5 Configuration . . . 36
3.2.6 Logical acquisition . . . 36
3.2.7 Physical acquisition . . . 36
3.3 Conclusion . . . 36

4 Design and Implementation **37**
4.1 Introduction . . . 38
4.2 Design . . . 38
4.2.1 Définition et concepts . . . 38
4.3 Implementation . . . 38
4.3.1 Réalisation . . . 39
4.4 Conclusion . . . 39

CONCLUSION **40**

List of Figures

1.1 HENCEFORTH's logo 13
1.2 HENCEFORTH's organization 14
1.3 Jira's logo 16
1.4 Github's logo 17
1.5 GANTT Diagram 17

2.1 Linux Kernel 19
2.2 Hardware Abstraction Level (HAL) 20
2.3 Libraries 20
2.4 Android Runtime 20
2.5 Comparaison 21
2.6 Java API framework layer 22
2.7 System Apps 22
2.8 Supported filesystems by Android 25

LISTE OF ACRONYMES

API	*Application Programming Interface*
CVE	*Common Vulnerabilities and Exposures*
DNS	*Domain Name System*
DRF	*Django Rest Framework*
DRY	*Don't Repeat Yourself*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
IT	*Information Technology*
JSON	*JavaScript Object Notation*
MVC	*Model, View, Controller*
MVT	*Model, View, Template*
NMAP	*Network Mapper*
ORM	*Mapping objet-relationnel*
OS	*Operating System*
OSINT	*Open-source intelligence*
REST	*Representational State Transfer*
SI	*Système d'information*
SQL	*Structured Query Language*
SSRF	*Server Side Request Forgery*
UML	*Unified Modeling Language*

XML *Extensible Markup Language*

XSS *Cross-site scripting*

XXE *XML external entity*

General Introduction

Intro generale ch

Chapter 1

Host Organization and General Context

1.1 Introduction . . . 13
1.2 Host Organization : HENCEFORTH . . . 13
1.2.1 HENCEFORTH presentation . . . 13
1.2.2 Main activities and Cyber security services of HENCEFORTH . . . 13
1.2.3 Organizational chart . . . 14
1.3 General Context . . . 14
1.3.1 Motivation . . . 14
1.3.2 Problem statement . . . 15
1.3.3 Objectives . . . 15
1.4 Project Management . . . 16
1.4.1 Working Method . . . 16
1.4.2 Management method and tools . . . 16
1.4.3 Project planning . . . 16
1.5 Conclusion . . . 17

1.1 Introduction

The first chapter of our report is dedicated to the general context in which our project fits. Therefore, we will start by a presentation of the host organization, namely HENCEFORH (researcH and developmENt in advanCEd inFORmation TecHnology), its activities and its architecture.Secondly, we will move to the main problem, motivation, and objectives of the project.Finally, we will conclude this chapter, by the method adopted for project management.

1.2 Host Organization : HENCEFORTH

1.2.1 HENCEFORTH presentation

HENCEFORTH (researc**H** and develop**ENT** in advan**CE**d in**FOR**mation **T**ec**H**nology) is a young company, that offers digital services specialized in Big Data and Cyber Security. Those services that HENCEFORTH offers to its clients are developed based on deep studies and scientific researches. [**alex2017forensics**]

Figure 1.1: HENCEFORTH's logo

1.2.2 Main activities and Cyber security services of HENCEFORTH

Main mission

HENCEFORTH aims to meet new technological challenges by focusing on research and development in new areas related to information and communication technologies in general and to the security of information systems in particular and Big Data. Thanks to this vision, HENCEFORTH offers its clients advice and support in the implementation of solutions related to the field of Big Data with all its facets and to the field of information systems security. It offers integrated hardware, software,training and consulting solutions related to these areas.

Cybersecurity services

Cyber security is one of the main activities of HENCEFORTH, by which it offers to its clients many high quality services:

- **Security Audit** : HENCEFORTH brings unique value through its experience in the security field by delivering comprehensive security audit services, to test the security configuration of clients' systems and check their compliance with security standards and good practices. After the test, our consultants provide a set of recommendations to correct the discovered vulnerabilities and enhance the infrastructure's security.

- **Forensics and Incident Response** : The team resposible of Digital Forensic and Incident Response Department, is a highly skilled one, that works on diferent situations of forensics to guide the clients step by step through the process of an incident. This department offers many types of forensics services such as Computer forensics, database forensics, application forensics, etc.
- **Integration** : HENCEFORTH guides its clients to implement their infrastructure, by providing design facilitation, network planning, network infrastructure optimization, installation, and integration services. Network integration tean is commited to offer a series of valuable features to upgrade and develop client's network infrastructures with advanced security features, which helps to improve adaptability to emerging technologies and changing business requirements.

1.2.3 Organizational chart

HENCEFORTH is devided into two main parts, the first one is for general direction, and the second one concerns research and developement. It thus offers its executives and engineers a pleasant working environment to carry out their research and development activities in collaboration with renowned researchers both nationally and internationally.

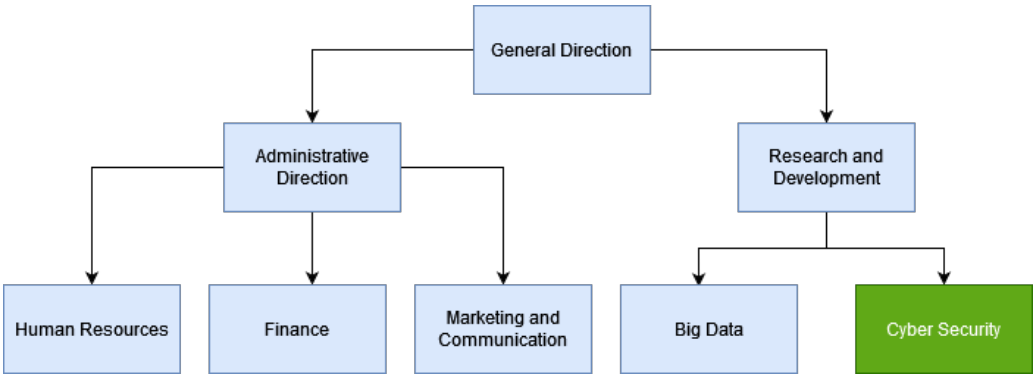

Figure 1.2: HENCEFORTH's organization

1.3 General Context

1.3.1 Motivation

Recently, Android devices usage has increased and become an integral part of our lives, and Android OS has over 70which become a target for attackers. The attack is more easy in Android than other OS such iOS, because Android devices are less secure than iOS phones. There are many ways to attack mobile phones, the most used one is by malicious applications; attackers are mastering their abilities to create persistent malware to be installed by the user without being aware of that. For iOS for example, there is one single controlled and verified applications store from which users can download applications, while for Android we find the official store but it can also accept applications from other sources even if there are unknown or malicious, the thing that facilitates the job for attackers to distribute their malware. Mobile security can be seemed as a personal issue, but it is not the case because now with the BYOD (Bring Your Own Device) approach that we find in organizations, users can bring their mobile

phones and connect them to the organization network, and that is a big deal for the security of the information system, because malware can spread through network, target and destroy the whole system. So mobile security now are more than a personal problem, it concerns everyone. That is why we should pay attention to the applications we download and install in our mobile phones.

Malware analysis is a powerful way that we use to parse software to check their credibility, it is a part of forensic analysis which is a detailed investigation(storage, CPU, memory, etc) analysts do to extract as much as possible of data from a given device with abnormal symptoms(for example: unknown open services, abnormal behaviour, weird communication with unknown entity, etc). For mobile devices, it will be interesting to have a detailed process to follow to analyse a phone and extract different artifacts. To do that, a research in depth needs to be done in order to look for tools and techniques that are available in the market.

At Henceforth organization, we find RD department, which is dedicated to research. Its main focus is on malware analysis in mobile devices. Actually, Henceforth is building a team of researchers to work in this department. So the first thing to start by is a general framework that describes the whole process of forensics, it should contain steps, good practices, and different tools. So, once we had a general architecture of Android forensics we can focus on each part. My internship subject will be the base and the guide of this department.

1.3.2 Problem statement

Once a phone got attacked by a malware, the first thing to do is looking for the causes and the sources of this malware, to identify how this malware infects the system, and to look if there is a leakage of information.All that will help in understanding the malware to take necessary precausions next time and to enhance the mobile security. To acheive this goal, we should have a detaild forensics guide that will help the analyst to extract as much as possible of data and artifacts from the phone with the most possible reliable way.

1.3.3 Objectives

The target of this internship is to provide a **framework** that will help the examiner, during the process of mobile forensics, to use the right tools and techniques to extract different data and artifacts from mobile phones. Using this framework, the analyst will be able to follow a set of good practices to treat the mobile phone with the most secure way to avoid data deletion or modification. As well as, the framework should cover workflows and techniques used to analyse mobile malware.

To acheive these goals, we are commited to do the following taks:

- Studying the state of the art on digital forensics, mobile malware, malware analysis, and obfuscation techniques.
- A study about available forensics tools
- Develop a framework that describes mobile forensics process in detail
- Forensic analysis of a memory dump, and mobile application.

1.4 Project Management

1.4.1 Working Method

To follow and control our project commitment, we were adopting the following method:

- Deliver daily, weekly, and mounthly reports to my supervisors to ensure their permanent feedback.
- Welcome change and crtics, keep a high level of motivation and question ourselves regularly.
- Ensure strong cooperation with my supervisors
- Aim for technical excellence and simplicity

During the period of my internship, I had two kinds of meetings:

- From one to three meetings per week with my industrial supervisor at HENCEFORTH Mr. **CHAKIR El mostapha** to follow the progress of the work, discuss ideas and concepts and find solutions for complex problems I faced during the development process.
- Meetings with the manager and the General director to follow progress of the general work and the achieved results.

1.4.2 Management method and tools

During the internship, we were working based on **Agile method** that consists of delivering work in small, and as things progress, requirements and results can be evaluated continuously so we will have a natural mechanismy to respond to change quickly.

To plan and manage our objectives, we used **Jira**, which is a tool developed by **Atlassian**. It helps teams to plan , track, and deliver software. Using Jira dashboard, we were able to easily plan tasks with their deadline, which facilitated the progress tracking by our supervisors.

Figure 1.3: Jira's logo

To upload delivered products, I used **GitHub** platform:

1.4.3 Project planning

Planning is the most significant phase that we should carefully take into consideration during the project process. It allows us acheive our targets with the most efficient use of time and other resources. By planning, we determine the way in which our goals will be acheived, we decide what is should be done and when.

Figure 1.4: Github's logo

To ensure an effective planning for our project, we used **GANTT** diagram, that helps to schedule and set tasks with deadlines. it is effective for visually representing the different tasks progress.

The following figure reprents the GANTT chart of our project planning:

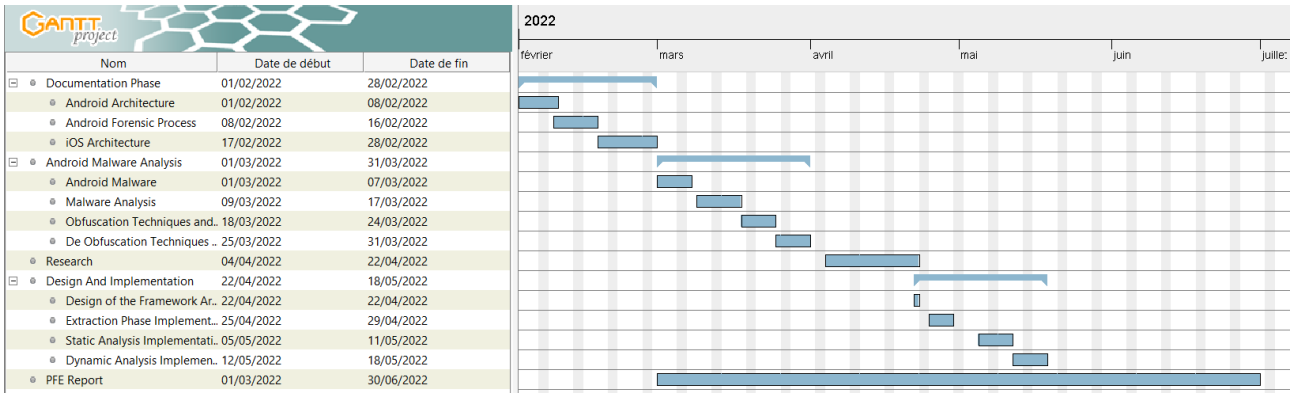

Figure 1.5: GANTT Diagram

1.5 Conclusion

In this chapter we gave an overview about host organization, its structure, and its business areas, as well as we presented a detailed description of the project and its objectives. The next chapter will be dedicated to the background and the general knowledge about digital forensics and mobile malware.

Chapter 2

BACKGROUND

2.1 **Introduction** . . . 19
2.2 **Digital Forensics** . . . 19
2.2.1 Definition . . . 19
2.2.2 Process . . . 19
2.2.3 ? . . . 19
2.3 **Android systems** . . . 19
2.3.1 Android architecture . . . 19
2.3.2 Boot processing . . . 22
2.3.3 Storage structure . . . 23
2.3.4 Android applications . . . 24
2.3.5 Security and malware in Android . . . 26
2.4 **Android Malware Analysis** . . . 29
2.4.1 Definition . . . 29
2.4.2 Static analysis and Obfuscation . . . 29
2.4.3 Dynamic analysis and its limits . . . 31
2.4.4 Tools . . . 33
2.4.5 Lab . . . 33
2.5 **Conclusion** . . . 33

2.1 Introduction

In this chapter, we will present some basic knowledges in order to facilitate the uderstanding of the upcoming work. Therefore, we will firstly start by a deep

2.2 Digital Forensics

2.2.1 Definition

2.2.2 Process

2.2.3 ?

2.3 Android systems

As we are interested in Android forensics, the first thing to start by, is a deep overview of Android systems, because we cannot do mobile forensics, either malware analysis, if we do not have enough knowledge about Android internals. The following subsections will deeply present different parts of Android devices.

2.3.1 Android architecture

Architecture

The Android OS consists of several layers running one above the other. Those layers are integrated in a special way to provide an effective execution environment for mobile devices.

- **Linux Kernel**

  The Android OS is built based on Linux kernel with some changes malde by Google. The choice of Linux was made because Linux is considered as a portabe OS that can be compiled easily on different hardware types. The kernel part is the one that separates the device hardware from the upper layers, as well as , it is considered as an abstraction layer between the software and the hardware of the device.Fo example, when we click on the camera button, the hardware instruction will be converted into a software instruction to take a picture and store it. This is done thanks to kernel drivers. When the device detects the camera button click, the hardware instruction goes to the responsible kernel in the kernel, that will send the necessary commands to the camera hardware. The drivers that we can find in the kernel are various such as Wi-Fi, Bluetooth, USB, audio, etc.

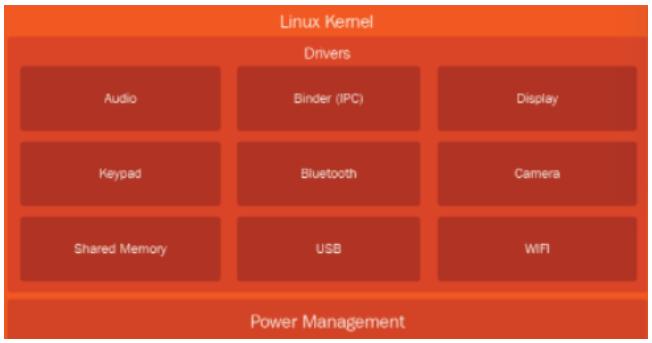

Figure 2.1: Linux Kernel

- **Hardware abstraction level (HAL)**

  The hardware abstraction level is used by Java API framework and the standard interfaces to work and interact with mobile device's hardware. This process is achieved through a set of library modules, that provide the necessary interfaces for each type of hardware component such as Bluetooth and camera.

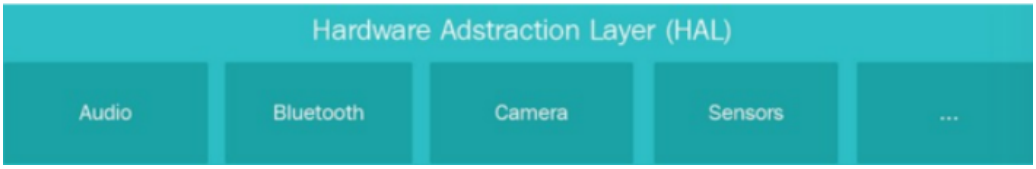

Figure 2.2: Hardware Abstraction Level (HAL)

- **Libraries**

  These libraries are written in C or C++ language, their purpose is to help the device to deal with diferent types of data. For example, the phone uses SQLite libraries to store and retrieve data from a database.

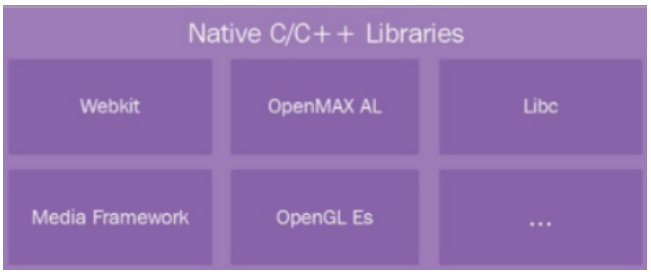

Figure 2.3: Libraries

- **Android Runtime**

  For Android systems, Java programming language is used to create all the applications, and after the compilation of a java program, we get bytecode that will be executed in a JVM(Java Virtual Machine). But for versions under 5.0, there was a special VM type called DVM(Dalvik Virtual Machine).

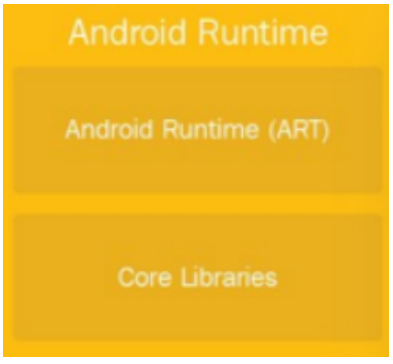

Figure 2.4: Android Runtime

The Dalvik Virtual Machine runs the Java bytecode, so we get .classe files the will be converted to .dex files using dx tool. Each application in Android runs its own instance

Figure 2.5: Comparaison

of DVM independly of the other DVM. The figure 2.5 gives an isight about the difference between DVM and JVM.

Before version 5.0, Android was using JIT compilation(Just In Time), that idetifies frequently executed operations and dynamically compiles them to native machine code that is called traces. But after version 5.0 has been released, Dalvik was replaced by ART which is Android Run Time taht uses AOT copilation (Ahead Of Time), that cmopiles the whole applications into native machine code upon their installation. Which optimizes the install time for Android applications.

- **Java API framework layer**

  This layer helps developere to create their Android applications by using system components, for example:

  - **View System** : is used to build user interface for applications, such as boxes, buttons, and so on.
  - **Content Provider** : is used to allow applications to share their data, and access other applications data.
  - **Resource Manager** : allows aplications to get access to non-code components of an application, such as localized strings.
  - **Notification Manager** : is used by applications to display their notifications and alerts.
  - **Activity Manager** : is used by applications to manage their lifecycle, the order in which an activity is opened, and so on.

- **The application layer**

  This layer consists of applications that users interact with. In general, there are two type of applications:

  - **System applications:** are pre-installed application that we find in the phone such as default browser. There are in read only mode, that we can find under the */system* partition. The user can not uninstall or change them.

Figure 2.6: Java API framework layer

– **User installed applications:** are all the applications installed by the user themselves from applications stores such as Play Store. We can find them under the */data* partition.

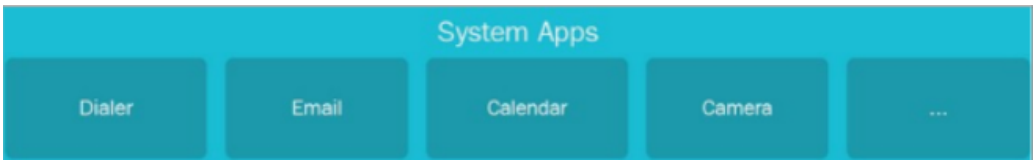

Figure 2.7: System Apps

Hardware

Android systems are supported by a set of hardware components that differ from a manufacturer to another. Here are some example of hardware components:

- **CPU (Central Processing Unit) :** is the part that is responsible for the xecution of anything that happens on the phone. It guides the device in what to do and how to do it. The number of tasks executed per second is the factor used to measure its performance.
- **Baseband processor :** is a processing unit used by network protocol supported by mobile phones like GSM, 3G, 4G, 5G, etc. Because these network protocols require a large amount of CPU power to treat data, generate, receive and transmit packets to the network provider. It is a separate unit that communicates with the main processor, and manages all network services functions such as signal generation, modulation, encoding, and transmission.
- **Memory :** we generally separate two types of memory:
  - **RAM (Random Access Memory)** :contains volatile data, that are deleted once the phone is powered off. It is used to load and execute applications.
  - **ROM (Read Only Memory)** : is a non volatile memory, so its content is retained even if the phone was powered off. It contains the boot loader, OS, downloaded applications and their data.
  - **SD Card :** an externe non volatile memory that is used to store multimedia and user data.
- **Battery, USB, Speaker, Microphone, and so on.**

2.3.2 Boot processing

Once the phone is powered on, there is a series of steps that are executed in order to load the necessary resources needed by the phone to start working.

- **Boot ROM code execution:** When the mobile phone is powered on, the boot ROM code ,that is specific to the used CPU by the phone, starts its execution. This execution is done through two phases:
  - Phase 1 : after execution, the device hardware is initialized and starts scanning to detect the boot media until it finds it.
  - Phase 2 : when the boot sequence is established, the boot loader is copied to the RAM memory to be executed.
- **The bootloader:** Before the OS starts it function, the bootloader is executed first. The execution is done in three steps:
  - **IPL(Initial Program Load):** it is responsible for detecting and setting up the RAM memory.
  - **SPL(Second Program Load):** when the RAM is detected, the SPL will be copied to the RAM and will be executed. The SPL is deals with loading the Android OS, providing access to diferent modes such as fastboot and recovery mode, initiating several hardware parts like keyboard and the console.
  - **Step 3:** the next step is to load Linux kernel from boot media and to copy it to the RAM memory to be executed.
- **The Linux Kernel:** is responsible for process and memory management and Android security. Once the kernel is loaded, the root filesystem will be mounted and access to system and user data will be provided.
- **The init process:** It is the root process that starts first, we can find it under *Android-source/system/core/init*. It searches the script init.rc that describes the system services, filesystem, and other parameters that need setting up. After launching the system service processes, the ANdroid logo will be displayed on the screen.
- **Zygote and Dalvik:** Once the phone is booted, and after the init process, Zygote will be lauched and will initiate the DVM and create a set of instances to support each Android process. After this, applications run by requesting new DVM.
- **The system server:** is the part that initiates the essential features of the phone like telephony and network. It sends a broadcast action that is called *ACTION-BOOT-COMPLETED* to inform all the dependent processes that the boot process has finished.Then, the phone start displaying the home screen and user interface.

2.3.3 Storage structure

File Hierarchy

As Android systems are based on Linux kernel, the file hierarchy of Android will be the same as Linux systems. So, files are organized under a single tree, and the top of the tree is symbolized by "/". Files can be accessed based on privileges that a user has. The following are some of the most famous and significante folders:

- ***/boot:*** contains all the needed files for booting, and the kernel and RAM disk.
- ***/system:*** contains system-related files such as the Android framework, libraries, system binaries, and pre-installed applications.

- ***/recovery:*** contains needed resources for recovery mode where updates and maintenace operations are performed.
- ***/data:*** contains applications data.
- ***/cache:*** the frequently accessed data are stored in this partition to make access more quickly.
- ***/proc:*** this is the filesytem mount point, that gives access to the kernel data structures.

Storage on Android systems

The mobile phone is dealing with an amount large of user data such as contacts, videos, emails, pictures, and so on. These data can be stored differently by the device, because there are multiple types of storage available in ANdroid systems, they are presented as follow:

- **Shared preferences:** this type is used to store primitive data types in XML format such as Boolean, float, int, and long.
- **Internal storage:** used to store applications data under the directory */data/data/*. This directory is privileged, which means that is accessed only by the root.
- **External storage:** can be removable like an SD card or non-removable that comes with the device.
- **SQLite database:** stores data in database, this type of storage is supported by ANdroid sytems. To see this kind, we move under */data/data/AppPackageName/databases*.
- **Network:** refers to all data type that are stored in a third party using network.

Android filesystem :

By definition, filesystem is the way in which files are named, placed, and retrieved. Each filesystem specifies its own conventions for storing files and allows applications to access different filesystems in a uniform way. Android supports various filesystem files as Linux does. The following figure presents different filsystems supported by ANdroid devices:

2.3.4 Android applications

APK files(Android Application Package)

Executable files of applications in windows have ***.exe*** extension, the same thing goes for Android applications, they have as extension ***.apk***, which means ***Android Application Package***. As we have already mentionned in previous sections, Applications in Android are written in Java programming language, and all the resources needed by the applications to work are compressed in one single file that we call apk file. The apk file contains the following components:

- ***assets:***
- ***META-INF:*** it is a folder that contains the manifest file and certificates that are used to sign the application.
- ***res:*** contains different resources needed by the application.

Figure 2.8: Supported filesystems by Android

- ***AndroidManifest.xml:*** it is a file that contains details about the application such as permissions, main activity, etc. It is a .jar file that needs decoding to read it. It is th firt file that will be read by Android to start the application.
- ***classes.dex:*** there are the compiled dalvik executable files.
- ***resources.arsc:*** contains the precompiled resources needed by the application such as XML files that support the UI component of an app.
- ***lib:*** here we find the compiled code that is specific to a processor like x86 or arm64.

Application components

Android applications use a set of components to communicate with the system and the other applications, they are described as follows:

- ***Intent:*** is an essential component, used to ensure communication between an application and the system or between applications with each-other. Each operation (for example requesting a resource from another application)that needs to be performed is expressed by a detailed message that is contained in an intent. Intent can be expressed explicitly when it knows the exact component needed, or implicitly when it has no knowledge about the required component, so the decision is left to Android system based on elements introduced by the intent.
- ***Activity:*** every single screen with user interface with which users can interact is called an **activity**.

- ***Services:*** are components that can be started and stopped without the user interface, and we can generally find them in long running tasks in the background. We distinguish two types of services:
  - ***Unbound services:*** is a component that starts a service and continues to run in the background even if we destoy the original component that initiated it.
  - ***Bound service:*** is a service that runs as long as the original component is not destroyed.
- ***Broadcast receivers:*** Android systems generally launch broadcast events to inform other applications of any changes that have been happened, so broadcast receivers come into play to capture these broadcast events to make required changes in the application.
- ***Content provider:*** is a component that is used by applications to share data with each-other.

APK files build process

Understanding how Android applications are built is quite interesting, it will help during reverese engineering that we will talk about in next sections.

We get the apk file through a long process which consists of a set of steps, described as follows: ***

- ***Source code generation:*** This first step aims to collect all the sources that will make the application works correctly, such as AndroidManifest.xml file, libraries, source files and so on. Those resources are compiled using **aapt** tool to get as output .jar file.
- ***Interface code generation(AIDL Files):*** AIDL files are Android Interface Definition Language files that define the programming interface for client and service to communicate using interprocess communication.
- ***Compilation:*** in this phase, a virtual machine is created, in which the .jar and AIDL files are converted into .class files that we call bytecode.
- ***Conversion:*** The bytecode (.class files) will be converted into .dex file which we call dalvik bytecode. This convertion is done using the **dextool** tool.
- ***Pre-building:*** The .dex files and other ressources such as assets, lib, and ressources.arsc folders are compressed using the **ApkBuilder** tool to get one single file with the .apk extension.
- ***Signing:*** The final step, is to sign the .apk file by the developer key using tools such as **JarSigner**. This phase is important since Android does not accept installing non-signed applications.

2.3.5 Security and malware in Android

Android security

Google has implemented certain features to ensure the security of users, data, and applications. In our context, before delving deeper into Android forensics, the analyst must first understand the security implementaion to know what can be or cannot be accessed under normal conditions. The following elements summerize Android security implementation:

- **Permissions:** For Android, to access any resource(access to internet or to contacts list for example) in the system, an application must get the permission of the user. Therefore, applications developers are required to explicitly declare all the permissions needed in the Manifest file. So during installation or using, the system will prompt the user to confirm the permission required by the application. This feature is important, because it gives to the user an idea about what resource the applications is looking to access to. For example, a chat application may need to send pictures, for that it should declare in the Manifest file the CAMERA permission.

  ***

  Android permissions are categorized by Google into four types as follows:

  - **Normal:** are permissions that do not pose a risk to the system or user, they are automatically allowed without the user confirmation during installation or usage.
  - **Dangerous:** are the ones that may cause harmful actions to the system and other applications. They are granted by asking the user approval.
  - **Signature:** are granted if the requesting application is signed by the same certificate as the requester application.
  - **System:** are permissions that only the system can grant them.

- **Sandboxing:** Android applications are running separately, each one is attached with its ID and runs in a sandbox isolatedly from the other applications. This feature is a big advantage for Android because it prevents applications from getting access to other applications resources since it has not enough permissions.

- **SELinux:** For 4.3 versions and upper, Android systems have started supporting SELinux, which means that the Mandatory Access Control(MAC) is imposed in Android device. So even if the device was attacked by the malware, the system part cannot be easily accessed especially files running with root privileges.

- **Application signing:** By design, Android refuses installing applications that are not digitally signed with a certificate. This signature is not necessary related to a certificate authority, which means that the developer can use self-signed certification. The main goal of this condition is to know the author of the application and to keep updates communication between the developer and the applicayion.

Android Malware

Malware is the abreviation of malicious software, that means every software that can cause harmeful damage to a given device. The damage depends on the type of the malware. For Android systems, malware refers to applications, specifically to apk files.

Types

We can generally categorize Android malware into the following types:

- **Adware:** is a very popular android malware, which means advertisement malware that displays continuous, annoying ,and unwanted advertisements on the user screen. These advertisements contains attractive offers to draw the user attention to click on them, and once the user clicks revenue is generated for the developer. Unlike revenue generation, there are some adware that result in the downloading of another malware more dangerous.

- **Spyware:** is a monitoring and loging software that sends important and sensitive data from the victim device to the attacker server that we call CandC server (Command and Control server). These data is later used for malicious activities. This transmission is done when the user gives permissions to get access to the device information such as gallery, sms, etc.
- **Ransomeware:** is a malware that encrypts files or locks the device of the user to make them inaccessible. In this case, the attacker asks for a ransom to provide the decryption key, but there is no garantee that the user will get the key after paying.
- **Cryptomining malware:** is a malicious software that targets the computing resources of the vectim device to mine cryptocurrencies such as bitcoin and etherum.
- **Trojan:** is a malware that holds malicious activities, but it looks legitimate so the user can not be aware of it.

Malware installation

Android malware generally use three ways of social engineering techniques to install onto users phones:

- **Repacking:** attackers tend frequently to use this techniques, that consists of downloading a legitimate and popular application, disassembling it, enclosing malicious payloads, and then re-assempling and submitting the generated application to the official store or an alternative market. Therefore, users can be triqued by the name and the logo of the application, so they can download and install the infected one.
- **Update attack:** This technique is similar to the first one, but instead of adding the payload and re-pack the application, it includes an update part that will look for or download the malicious payload at runtime.
- **Drive by download:** is a social engineering trick that invites and entices users to downloadand install an application by drawing their attention with interesting and attractive features.

Malware activation

To activate the malicious action of Android malware, attackers use various tricky techniques and period of time to lauch the malware, so the user will not be aware of the abnormal behaviour of the malware. In general, malware focus on the built-support of automated event notification on Android systems to trigger the payload. One of the most used system event is BOOT-COMPLETED, that triggers the malware once the system finishes its booting process.

Malware identification

There are a lot of automated systems that can be used to identify known malware, they are categorized in the following items.

- **Antivirus Scanners:** as a first step, it is recommneded to filter known malware.
- **VirusTotal :**a free online tool that analyses URL and diferent files to check whether they are malicious. It contains a very large database of known malware hashes. ***

- **Yara rules:** a language aims to give hands to engineers to set rules based on textual or binary patterns found in a sample of malware, so the malware will be detected if some or all of the rules are matched. ***

However, these systems can not be effective especially for new malware, so the only effecient way to identify malware is malware analysis that we are going to cover in the next section.

2.4 Android Malware Analysis

2.4.1 Definition

Malware analysis is a process that aims to deeply understand the behaviour of a malware and its impact on a device. Its main goal is to get a detailed description of the malware function and implementation so blue team experts can enhance the system security by reinforcing malware detection systems and threats mitigation.

The following are some of the key advantages of malware analysis:

- Identifying the attack source
- Identifying security damages caused by the analyzed malware
- Determining vulnerability exploitation level of the malware and future patching
- Enhancing IOC(Indicators Of Compromise), alerts, and notifications

To analyse malware, there are three approaches to choose; static, dynamic, or hybrid analysis.

2.4.2 Static analysis and Obfuscation

Android Malware Static Analysis

Static analysis consists of extracting static characteristics of a malware without executing it. It looks for every abnormal indicator and string such as file names, signatures and hashes, and ip addresses.

To do static analysis, we only need an apk file and a set of tools, the following items summarize the whole process of static analysis:

- With a given apk file, the first thing to do is to calculate its hash and search, using malware identification tools, if it was a known malware or not.
- After checking with known malware, we pass to reverse engineering. It is a process that aims to go back to understand how the application was implemented.It is done through the following steps:
  - **Unpacking:** an apk file is generally compressed, so to see its content we need to decompress it using any archiver such 7-Zip or others.
  - **Decoding:**To view the apk components we need to decode it using decoding tools such as **axmldec** or **apktool**. After decoding, we can see all the resources in a readable format, so we can check the **META-INF** folder to see the certificate used to sign the application. As well as, we can open the manifest file to see the permissions that the application asks for, main activities to see which activity starts once the malware is run, broadcast receivers to see how the application receives

system events. By analyzing those elements we get a a preliminary idea about what the application tries to acheive.

- **Decompilation:**After decoding we get .dex files that we convert to .class files in a .jar container using decompilation tools such as **dex2jar**.
- **Viewing and analyzing:**Once decompilation is done, tools such **JD-GUI** can be used to view the source code of the application that we can review to seek some key words such as ip addresses, http connections, and so on. Moreover, with the source code we can extract the imported libraries to know the required materials that will be used by the malware. For ransomeware for example, they will need to import encryption libraries.

The following figure modelises the static analysis process:
*** Static analysis is the safest way to understand how a malware works, as we do not have enough information about its real behaviour, and executing it can cause serious damages to the phone or the whole network in which the phone is connected.

Malware Obfuscation Techniques

Sophisticated malware are actually aware of static analysis, therefore malware authors are working hard to enhance their techniques and tricks to hide the malicious action found statically. It is what we call obfuscation techniques which refer to every strategy that changes the content of malware resources(such as .dex file or the manifest file) while preserving the original functionalities of the application and without modifying its semantic. The legitimate use of obfuscation is for code confidentiality that developers tend to use for closed-source applications. However, it recently became a trend for malicious application to evade detection.

We distinguish two types of obfuscation:

- **Trivial techniques:** are obfuscation strategies that are simple and do not require much time and skills. They consist in:
  - **Repacking** that aims to decompress the apk file, change the order of its components, then repack it, and resign it with a new custom signature which changes the hash of the application, and so it will not be detected by the based signature scanners.
  - **Disassembling and re-assembling** the bytecode which modifies the placement of .dex files and then the signature will be changed, which help to evade based classes.dex signature systems.
  - **Changing package name**
  - **Alignment** that realigns the apk file data, which generates a new hash signature.
- **Non Trivial techniques:** are complicated and require much time and resources to obfuscate the bytecode, which justifies their effectiveness against detection tools. The following items summarize some of the methods used in this technique:
  - **Identifier renaming :** consists in renaming methods, classes, and variables with random names to confuse the analyst. Code before renaming: *** Code after renaming: ***
  - **Call indirections :** consists of modifying the graph of the calls, for example a method call A will be substututed with a new method call B that when invoked makes the original method call A.
  - **Code reordering :** modifies the order of the instructions in the source code.

- **Junk code insertion :** adds useless instructions in the code that does not impact the semantic of the application after execution, which helps in evading detection systems that inspect the sequence of the instructions of the malware.
- **Encrypting payloads and native exploits :** encrypts payloads that will stay encrypted until runtime.
- **Bytecode, strings, and class encryption :** encrypts some instruction and strings in the code, and decrypts them at runtime.

  example of code after encryption:
  ***

Some obfuscation tools

There are many tools used by developers and attackers to obfuscate their code, some of them are:

- **Proguard:** a tool included in Android SDK, that renames classes, methods and variables.
- **DexGuard:** a commercial version of **Proguard** that encrypts and renmaes classes with non ASCII symbols.

As we just explained, static analysis can not be effective in case the malware was obfuscated. Therefore dynamic type is the next step that comes after static analysis, to view the real actions of the malware.

2.4.3 Dynamic analysis and its limits

Dynamic analysis

Dynamic technique aims to execute the malware in a controlled and isolated environment called ***Sandbox*** which allows analysts to observe the malware in action without the risk of infecting the device or spreading through the network in which the system is connected. During execution, the analyst should capture a series of artifacts to determine its impact on the system. Those artifacts are described as follows:

- **Network communication:** a network capture with **wireshark** or **tcpdump** is quite interesting, because the majority of sophisticated malware tend to communicate with an external entity , called Command and Control server, to send sensitive information. Once we detect the network communication, we should search the ip address reputation to see if it was a blacklisted address.
- **Running processes:** once the malware is run, many processes will start running, so by listing those processes, the analyst will be able to understand how the malware intracts with the system resources. To do that, some linux commands such as **ps** and **fridaps** will do the job perfectly.
- **System calls:** are the system functions called by every running process. To list them, we can attach the process id of the malware to the **strace** command.
- **RAM usage:** is a powerful indicator that proves the abnormal behaviour of an application. Therefore, it is important to display the usage of RAM memory by the malware during dynamic analysis.

- **Battery usage:** is another significante indicator that can be useful to determine how the executed software uses the device resources.
- **Memory dump:** contains everything happens in the system and interesting artifacts about the malware. As we already have mentionned in a previous section, many malware use encryption to hide malicious payloads which will be decrypted at runtime. Those decrypted payload are found in plain text the memory.

The figure below medelises the dynamic analysis process:
***

Dynamic analysis limits

- **Environment evasion**

  Even if the dynamic technique is considered as a powerful and effective method to completely understand the malware behaviour, new generation malware are aware of the usage of sandboxes to run malware, therefore malware developers use some techniques to first detect the execution environment and once the sandbox is detected, the malware postpones its malicious action or launch a benign activy so the analyst classifies the malware as a legitimate application. This technique of evasion is based on parameters and artifacts that differentiate a real phone from a sandbox. Those artifacts are described below:

  - ***Static heuristic:*** refers to every parameter with a fixed and unique value that identifies phones. For example, the device IDs, IMEI (Internatinal Mobile Station Equipment Identity) which is a unique number that identifies the device in the GSM network aand IMSI (International Mobile Subscriber Identity) which is asociated with the SIM card. These IDs have unique and unified values for each device, that are equal to ***null*** for emulators , sandboxes, and virtual machines which facilitates the detection of the execution environment by malware. Moreover, there are other unique parameters related to the model and the device manufacturer that have default values on emulators such as Google-sdk or goldfish.
  - ***Dynamic heuristic:*** refers to different sensors that we find in real mobile devices such as accelerometer, gyroscope, GPS, and so on. The values of those sensors depend on the environment such as weather and location. While sandboxes have not this kind of interaction with the external environment and simulating such parameters is a challenging mission, which makes a clear difference between real phones and virtual machines. Morever, in real devices we notice that there is certain activities specific to users like photos, contacts, calendars, and modifying default configuration, that do not exist in emulators where we find that default settings are the same. So, based on all these dynamic heuristic, sophisticated malware can detect wheither the environment is real or not.

- **Unsatisfied conditions**

  Another limit of dynamic analysis is related to conditions that the malware looks to meet. For example, a spyware that collects users data to send them to a control server, so as not to attract the user attention, the attacker will choose monday at 02:00 am as a perfect time to transmit data, as monday is a working day and the user will go to sleep early, the user will not pay attention to this secret communication. So in this case, this condition of time should met to trigger the malicious action, However, during dynamic analysis this kind of conditions may be not be realized which means that the analyst will not be able to understand and analyse the malware behaviour.

2.4.4 Tools

2.4.5 Lab

2.5 Conclusion

In this chapter,

Chapter 3

Android Forensics Process

3.1 Introduction . . . 35
3.2 Android forensics guide and configuration . . . 35
3.2.1 General steps . . . 36
3.2.2 Seizure . . . 36
3.2.3 Extraction . . . 36
3.2.4 Analysis . . . 36
3.2.5 Configuration . . . 36
3.2.6 Logical acquisition . . . 36
3.2.7 Physical acquisition . . . 36
3.3 Conclusion . . . 36

3.1 Introduction

Nous allons entamer ce chapitre par une présentation des outils utilisés pour la réalisation de ce projet et leur utilité puis nous allons passer pour décrire les différentes étapes du développement des différentes interfaces de notre solution.

3.2 Android forensics guide and configuration

3.2.1 General steps

3.2.2 Seizure

3.2.3 Extraction

3.2.4 Analysis

3.2.5 Configuration

3.2.6 Logical acquisition

3.2.7 Physical acquisition

3.3 Conclusion

Ce chapitre a été consacré pour la description des étapes parcourues dans la mise en œuvre et la réalisation de notre solution....

Chapter 4

Design and Implementation

Nous allons entamer ce chapitre par une présentation des outils utilisés pour la réalisation de ce projet et leur utilité puis nous allons passer pour décrire les différentes étapes du développement des différentes interfaces de notre solution.

Table of contents

4.1 Introduction 38
4.2 Design 38
4.2.1 Définition et concepts 38
4.3 Implementation 38
4.3.1 Réalisation 39
4.4 Conclusion 39

4.1 Introduction

4.2 Design

4.2.1 Définition et concepts

4.3 Implementation

4.3.1 Réalisation

4.4 Conclusion

Ce chapitre a été consacré pour la description des étapes parcourues dans la mise en œuvre et la réalisation de notre solution....

CONCLUSION