



University of Piraeus
Department of Digital Systems

Postgraduate Programme
«Security of Digital Systems»

Master's Thesis

Forensics on Android applications

Kitsaki Theodoula-Ioanna

Under the supervision of **Dr. Christoforos Dadoyan**

Piraeus 2017-2018

Table of Contents

Table of Figures.....	iii
Table of Tables	iv
Acknowledgements.....	v
Abstract.....	vi
1. Introduction.....	1
2. Related Work.....	3
3. Background - Android Operating System	5
3.1 What is Android?	5
3.2 Android features.....	6
3.3 Environment Setup.....	7
3.4 Android boot process	8
3.5 Android SDK and ADB.....	10
3.6 Architecture.....	11
3.6.1 The need for a common architecture	11
3.6.2 Architecture components.....	11
4. Android Forensics	22
4.1 Android storage possibilities.....	22
4.2 Filesystem	25
5. Methodology	27
6. Android application Scenarios.....	30
6.1 My COSMOTE Android application	30

6.2 What's Up Android application.....	40
6.3 My CU Android application.....	49
6.4 OASA Telematics Mobile Android application.....	58
7. Future Work.....	62
8. Conclusions.....	63
References	65

Table of Figures

Figure 1: Android OS Booting Flow Chart	8
Figure 2: Android operation system architecture components	12
Figure 3: The final Android architecture	21
Figure 4: Flow chart of methodology.....	29
Figure 5: Structure of <code>\databases</code> folder	31
Figure 6: Structure of application.db database via DB Browser for SQLite.....	32
Figure 7: "Browse Data" tab selection/preview	32
Figure 8: Table selection on "Browse Data" tab	32
Figure 9: Open default.realm via Realm Studio.....	33
Figure 10: Preview of 5A68895B039D-0001-3B82-8ACFCEF5F130SessionApp.json file.....	34
Figure 11: Preview of log files	35
Figure 12: Preview of com.crashlytics.settings.json file	35
Figure 13: Preview of session_analytics.tap	36
Figure 14: Preview of cookies table	36
Figure 15: Preview of <i>USERATTRIBUTES</i> table.....	38
Figure 16: Preview of 5A9A604D01C1-0001-40C2-19BB625A33E9user.meta file.....	43
Figure 17: Tables of WhatsUpDatabase.db database.....	44
Figure 18: My CU user's data/credentials.....	50
Figure 19: Preview of cookies table	51
Figure 20: Preview of autofill table.....	52
Figure 21: OASA Telematics app's home screen.....	58
Figure 22: Cache preview.....	60

Table of Tables

Table 1: Versions of Android..... 5

Acknowledgements

This thesis is dedicated to my family, Mike, Glykeria, Vaggelis, Padelis and my beloved grandparents Leonidas and Artemis for their endless support and patience.

I would like to thank my thesis advisor Dr. Christoforos Dadoyan for the guidance and advice of my study and research.

Abstract

The purpose of this study is forensic analysis of specific Android applications. These were investigated in order to be examined whether they store information on device storage or not. The selected Android applications were My COSMOTE, What's Up, My CU and OASA Telematics Android applications. The directory in which usually Android application store their data is `\data\data\application_name`, thus the analysis focuses primarily there. After months of investigation the results were really interesting since these Android applications used from many Android users. Findings indicate that My COSMOTE mobile Android application did not store data on user's device and needs internet connection in order to run. What's Up mobile Android application saves encrypted data on device so it can run without internet connection using these non-updated data. My CU mobile Android application saves user's credentials in cleartext on their device. For this reason, without internet connection, user stay logged in but they are not able to be informed about their account details. For this activity, internet connection is required. Finally, OASA Telematics mobile Android application locates all the necessary on user's device. However, for some operations, such as real-time bus location, internet connection is a necessary precondition.

1. Introduction

The increased use of Android devices, mainly smartphones, and the variety of their capabilities both facilitate and cause problems to users. They are able to download applications on their devices either for their entertainment or not. In most cases, mobile applications require permissions such as access device storage, access contacts list, permissions to read and write on mobile devices if need be, and muchmore in order to operate. For this reason, it is important for users to know how applications treat their devices and their data. They should be informed about which of their data applications use and the location in which these store them either into devices or in servers. If they save data on devices it is important for users to know if these are encrypted or not and how easy could be compromised from a malicious attacker.

For these reasons, this thesis adduces everything a user should know about Android and forensic science, as well as presents experiments in four popular Android applications. At the beginning, related investigations on other popular Android applications are referred. These are Facebook, Skype, WhatsApp, Viber, ChatSecure and WeChat Android applications.

Chapter one mentions all the necessary details for Android operating system. The Android term, its features and the environment setup of Android applications are defined. Subsequently, details about Android SDK and ADB tools are mentioned and at the last section is referred to Android architecture, the reason which is necessary a common architecture and then the components consists of it .

Chapter two combines Android operating system with forensics science. It explains what forensics is and mainly mobile forensics. Consequently, the partitions in which Android application can store their data and the Android filesystem are referred. This information is useful for the investigator in order to know the partitions that should be analyzed.

In chapter three, are explicitly described the methodology steps which are followed in the investigation of each one from the selected applications. All the necessary tools which have been used are mentioned, as well as errors which showed up and their possible solution.

In chapter four, the Android applications which are chosen for forensic analysis are presented. These are My COSMOTE, What's Up, My CU and OASA Telematics Android applications, which widely used by Android users. Every step of this inspection has been extensively presented, followed with related screenshots/figures, in order for a reader new to this area to fully understand the process.

Last chapters mention proposals for future inspection. Both re-examine the selected Android applications presented in this thesis at their new versions' and analysis on other popular Android applications. Ultimately, conclusions of this investigation are presented.

2. Related Work

In this thesis Android applications were selected, which have not been previously examined. However, making a quick search found that in many widespread Android applications became a criminal inspection.

Already in 2011 the Valkyrie-X Security Research Group (VXRL) analyzed the Facebook application [1]. Using ready-made tools and platforms were investigated signatures and traces of Facebook from RAM or cache file browser for several activities, such as comments and discussions. Same results were found in the virtual machine image files. Also found traces of data and files related to an Android application account which stored in various mobile user devices.

In 2014, McQuaid dealt with the analysis of Skype on Windows, iOS and Android [2]. In a database there is information about a user's account, calls, messages and group conversations, contacts, files' transfers and more. In the same folder as this database there are files containing user's conversations and information about them, as well as the IP addresses associated with any Skype activity.

Similar findings were found in research done on WhatsApp and Viber applications. In 2013, Mahajan, Dahiya and Sanghvi were the first that dealt with these two applications [3]. In the first application were found databases of messages, conversations and contact lists. In the second application was stored history of calls made on Viber or not, contacts and messages and more relevant information. A more detailed analysis was made in 2015 by Lone, Badroo, Chudhary and Khaliq [4], where apart from the databases research extended to the *\shared_pref* folder, the external memory and the network level information. In both cases the investigations were performed both on rooted and non-rooted mobile devices.

More recently, ChatSecure (2016) and WeChat (2017) mobile applications were analyzed [5, 6]. Exploring databases and folders, as previously, in the first application found some encrypted data which decrypted and combined. As a result, user accounts contact lists, messaging and

metadata were analyzed. In the second case were found encrypted messages and other media data in raw form. It is worth to point out that in all these cases used commercial tools and platforms mainly for the analysis of recovered data. The most prevalent platforms were Cellebrite UFED, XRY and Oxygen.

3. Background - Android Operating System

3.1 What is Android?

With the Android term meant an open source operating system based on Linux, which is used in mobile devices like smartphones and tablets. The development of this operating system was a collaboration project among different companies, with Google to be the leader.

It offers a unified approach to application development for Android devices. This means that developers should develop applications only for Android, but they should be able to run on different Android devices.

The first beta version of Android Software Development Kit (SDK) was released in 2007 by Google and September 2008 as a commercial version with the name Android 1.0. Then several updates to the original version were released:

Table 1: Versions of Android

Version	Code name	API level
1.0	No code name	1
1.1	No code name	2
1.5	Cupcake	3
1.6	Donut	4
2.0, 2.0.1, 2.1	Eclair	5, 6, 7
2.2.x	Froyo	8
2.3 - 2.3.2, 2.3.3 - 2.3.7	Gingerbread	9, 10

3.0, 3.1, 3.2.x	Honeycomb	11, 12, 13
4.0.1 - 4.0.2, 4.0.3 - 4.0.4	Ice Cream Sandwich	14, 15
4.1.x, 4.2.x, 4.3.x	Jelly Bean	16, 17, 18
4.4 - 4.4.4	KitKat	19
5.0, 5.1	Lollipop	21, 22
6.0	Marshmallow	23
7.0, 7.1	Nougat	24, 25
8.0.0, 8.1.0	Oreo	26, 27

<https://source.android.com/setup/start/build-numbers#platform-code-names-versions-api-levels-and-ndk-releases>

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

3.2 Android features

Android is a powerful operating system with a variety of features. Some of these are the following [7]:

- Attractive user interface (UI)
- Connectivity: Supports GSM / EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX
- Storage: For data storage it uses the SQLite relational database
- Media support: Supports H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP

- Messaging: SMS and MMS
- Web browser: Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3
- Multi-touch: Android has native support for multi-touch which was initially made available in handsets
- Multi-tasking: User can jump from one task to another and same time various Android applications can run simultaneously
- Resizable widgets: Widgets are resizable, so users can expand them to show more content or shrink them to save space
- Multi-Language: Supports single direction and bi-directional text.
- GCM: Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
- Wi-Fi Direct: A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
- Android Beam: A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

3.3 Environment Setup

An important advantage is that an Android application can be developed on the following operating systems:

- Microsoft Windows XP or later version

- Mac OS X 10.5.8 or later version with Intel chip
- Linux including GNU C Library 2.7 or later

Another positive aspect is that every tool needed to build Android applications is free and can be obtained easily from the internet. Some useful software is Java Development Kit (JDK), Android Studio and Eclipse IDE.

3.4 Android boot process

As mobile platforms have some differences compared with Desktop systems with regard to their starting up and booting is useful referring to the initial starting stage of a Android mobile [8, 9]:

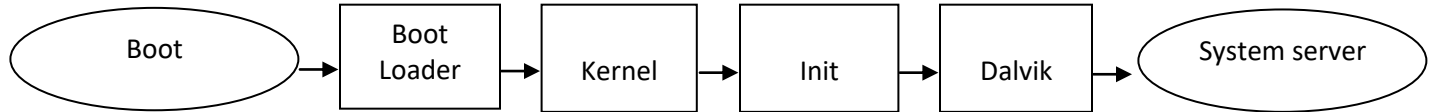


Figure 1: Android OS Booting Flow Chart

1. Boot: When CPU is powered on, essentially there has been no initialization. The internal clocks are not being set up and the only available memory is the internal memory RAM. When power supplies are constant voltage then the execution will start with the Boot ROM code.
2. Boot loader: This is a special program separate from the Linux kernel that is used to make initializations and load the kernel to RAM. On desktop systems the boot loader are defined as programs, while on embedded Linux as the chosen U-Boot. Device manufacturers usually use their own boot loaders. For Linux running

on ARM systems, the requirements of a boot loader can be found in the Booting document under /Documentation/arm in the kernel source tree.

3. Linux kernel: The kernel of Linux starts up in a manner similar to Android as in other systems. It will configure everything needed for the system to run. Initializes interrupt controllers, sets up memory protections, caches and scheduling.
4. Init process: The initialization process is the grandma of all system processes. Every other process on the system will start with this process or one of its descendants.
5. Dalvik: Dalvik starts from the initialization process. Essentially, it starts by simply running and initializing the Dalvik VM.
6. System server: This is the first java part that runs in the system. It will launch all Android services such as telephony manager and Bluetooth. Start up of each service is currently written into the execution method of system server. The source code of system server can be found in the frameworks\base\services\java\com\android\server\SystemServer.java file in the open source project.
7. Boot completed: Once the system server runs and the system boot is completed, there is a standard broadcast action called ACTION_BOOT_COMPLETED. A registration needs to be made of a successful boot, in order for a user to run an Android application after startup.

3.5 Android SDK and ADB

The investigation of an Android device has been made a lot easier now by using some freeware online tools available. Android Software Development Kit (SDK) is a set of software development tools. It contains all the necessary resources for Android applications' creation. It is also a powerful forensic tool. SDK conveniently assists a developer in order to create and debug an Android application or an analyst in order to investigate the characteristics of an Android device.

An important tool is Android Debugging Bridge (ADB). This tool allows communication with an emulator or connection with an Android device. Basically, it is a server-client program, which is consisted of a client, a server and a daemon. Client runs on the analyst's machine. The researcher can create a client via a shell using the adb command. For this purpose, the analyst can use alternative tools such as DDMS which is used in the present thesis. The server runs as a background process on analyst's machine and it is responsible for managing the communication between client and the adb daemon. This daemon runs as a background process either on an emulator or on a (mobile) device.

However, in order for someone to connect to any Android device from a computer and transfer data, an option from "Developer Options" tab must be activated beforehand on the device. This option is called "Usb Debbing". In our experiment using an Android device version 6.0.1, every time a user activates this option a warning has been prompted.

3.6 Architecture

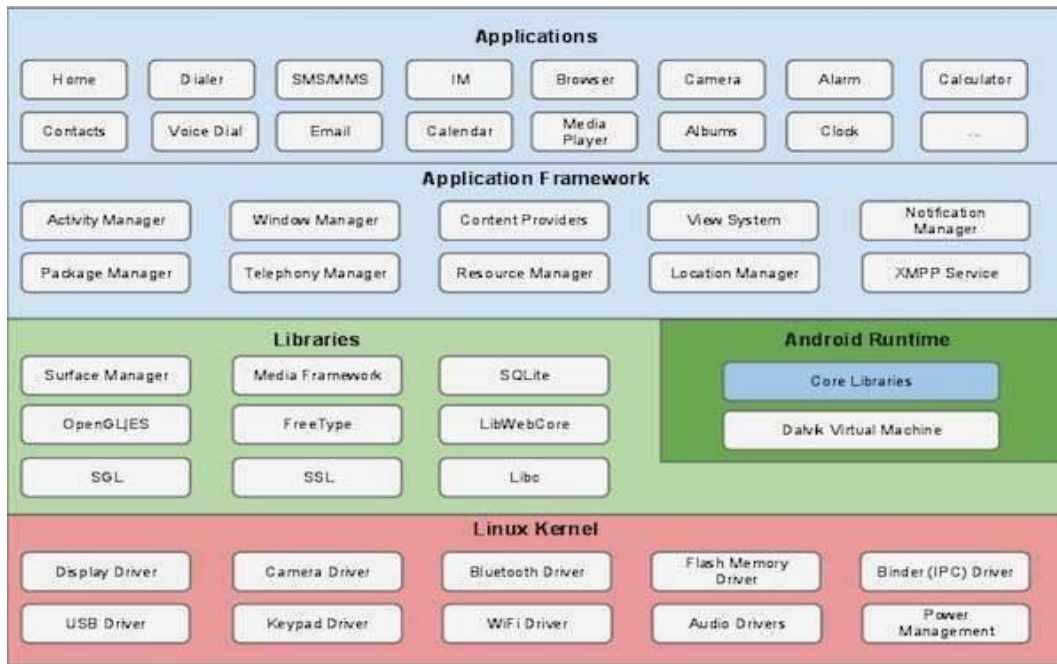
3.6.1 The need for a common architecture

A fairly important point which deserves to be clarified is the reason that architecture is needed. Android framework enables developers to write all of their code in the Activity class. An activity is a unique and targeted action that a user can perform. Almost all activities interact with the user, so the Activity class creates a window in order programmer to write interface's (UI) code.

However, this ability creates some problems, which had been resolved by creating various architectural standards such as MVP (Model-view-presenter) and MVVM (Model-view-viewmodel). These standards enable code writing to be robust and easy code correction and testing process. Once Android team realized the need for a common direction, officially announced Android components and gave relevant guidelines.

3.6.2 Architecture components

Android operating system is a stack of software components which is roughly divided into five sections and four main layers [10, 11].



<https://www.quora.com/What-is-the-Android-application-Architecture>

Figure 2: Android operation system architecture components

These sections are the “Android applications”, “Android application framework”, “libraries”, “Android runtime” and “Linux kernel” sections. These sections are grouped into four layers. In the bottom layer is the Linux kernel layer, in the above layer are contained libraries and Android runtime sections, then the Android framework layer and in the higher layer is Android applications section. A more detailed description of these sections is presented as follows:

- Linux kernel

At the bottom layer, Linux kernel is located which is based on version 2.6 of Linux. It provides an abstraction layer between device’s hardware and higher levels of the Android software stack. Actually it provides a preemptive multitasking. Here are performed basic low-level system services such as management of memory, processes and energy. It also contains all the necessary drivers for the correct functioning of the device, such as camera and

keyboard. Finally, this layer manages all operations in which Linux is well known for, such as networking and a wide variety of devices' drivers which are responsible for interconnection with hardware.

The first Linux kernel was developed in 1991 by Linus Torvalds and is combined with a set of tools, utilities and compilers developed by Richard Stallman at Free Software Foundation in order to create a complete operating system called GNU / Linux. Various Linux distributions build upon these foundations, like Ubuntu and Red Hat Enterprise Linux.

It is worth noting that Linux was originally designed for desktop computers and servers environments and not for mobile use. Android is the only operating system which uses Linux. In fact, Linux is most widely developed in critical corporate server environments. That proves the power, effectiveness and efficiency of current Android smartphones.

- Android Runtime – Dalvik Virtual Machine

This is the third section of Android architecture and the second layer from the bottom part. Here is provided a key component called Dalvik Virtual Machine (VM), a kind of Java virtual machine specially designed and optimized for Android.

As previously mentioned, Linux kernel provides a multi-execution environment that allows multiple processes to run concurrently. It can be assumed that every Android application runs just as a process directly in the Linux kernel. In fact, each Android application executes its own process using its own Dalvik VM instance of the device.

Android applications running on virtual machines offer a number of advantages. Initially, Android applications are mostly sandboxed, because neither they adversely affect (intentionally or otherwise) the operating system or other Android applications nor can have direct access to the device's hardware. Secondly, this forced abstraction layer makes Android applications neutral regarding platform, because they have never been associated with any particular hardware.

This VM was developed by Google and based on Linux kernel for low-level functionality. It is more efficient than standard Java VM regarding memory usage and it has been designed to allow multiple instances run efficiently with the resource constraint that a mobile device comes with.

In order Android application code to be executed within a Dalvik VM should be transformed from standard file type (Java classes) into executable Dalvik (.dex) format, which has a 50% smaller memory footprint than standard Java bytecode. Files of standard Java class can usually (if not always) be transformed into Dex format by using dx tool included in Android SDK.

As mentioned in details as follows, another feature provided in this section is a set of core libraries that allow developers to create Android applications using the Java programming language.

- Android Runtime - Dalvik Libraries,

Main libraries of Android, known as Dalvik libraries, are divided into three categories, which are:

- Java Interoperability Libraries

For development of Android applications, Java programming language is used. Standard Java development environment includes a wide variety of classes contained in core Java runtime libraries. These libraries provide support for tasks such as string management, networking and file handling.

Moreover, these libraries are used widely by Java developers, regardless of the platform of use.

These libraries are an open source implementation of a subset of standard Java core libraries which have been adapted and transformed for use by Android applications running on a Dalvik VM.

- Android Libraries

This category includes libraries based on Java and specifically used for Android development.

Examples of such libraries are libraries of Android application framework and those that facilitate the creation of user interface, the graphics' design and the access of the database.

Some basic libraries for Android development are:

- ✓ android.app: Provides access to the Android application model and comprises the basic library on all Android applications.
- ✓ android.content: Facilitates access to content, publication and exchange of messages between Android applications and their data.
- ✓ android.database: It is used to access data published by content providers and includes classes for managing SQLite database.
- ✓ android.graphics: A two-dimensional (2D) API (Android application programming interface) for the low-level graphic design, including colors, signs, filters, rectangular and canvases.
- ✓ android.hardware: Presents an API that provides access to the hardware such as acceleration measure and light sensor.
- ✓ android.media: Provides classes which help the reproduction of audio and video.
- ✓ android.net: A set of APIs that provide access to the network stack. It includes android.net.wifi, which provides access to the wireless device stack.
- ✓ android.opengl: A Java interface to the OpenGL ES 3D graphics rendering API.

- ✓ android.os: Provides access to Android applications on standard operating system services, including messaging, system services and communication between processes.
- ✓ android.provider: A set of classes that provide access to main Android databases of content provider such as those maintained by calendar and contacts Android applications.
- ✓ android.text: It is used to perform and manage text on a device's screen.
- ✓ android.util: This is a set of useful classes which are responsible for tasks' execution such as converting strings and numbers, management of XML language and date and time configuration.
- ✓ android.view: The basic units for building interfaces for the users of Android applications.
- ✓ android.widget: A rich collection of pre-built user interface's components, such as buttons, labels, list views, layout managers, radio buttons etc.
- ✓ android.webkit: A set of classes designed to allow Android applications to integrate browsing capabilities on the web.

- C/C++ Libraries

The basic libraries which mentioned above are based on Java and provide basic APIs for programmers in order to develop Android applications. However, it is important to be noted that these libraries do not perform much of the actual work and are substantially Java "wrappers" which build upon libraries based on C/C ++. For example, when a call to the android.opengl library is made in order to perform a design task (i.e. design three-dimensional (3D) graphics on the display), under the hood the library perform another internal call to OpenGL ES C ++ library which cooperates with the Linux kernel in order to complete the initial request.

C / C ++ libraries are included to perform a wide and diverse range of functions, such as design of 2D and 3D graphics, SSL communication (Secure Sockets Layer), management of databases SQLite data, audio and video play, management of graphics and an implementation of the standard C system library (libc).

In practice, typically an Android application developer will have access to C/C++ libraries exclusively through Java APIs “wrappers” libraries. In case that immediate access to these libraries is required, this can be achieved using Android Native Development Kit (NDK), whose purpose is calling the intrinsic methods of programming languages except Java (such as C and C ++ languages) through the Java code using Java Native Interface (JNI).

- Application Framework

This layer provides a high-level service on Android applications with the format of Java classes. This framework implements the idea that Android applications are made of reusable, interchangeable and replaceable components. This idea is one step closer to the notion that Android applications are also able to publish their capabilities, along with any corresponding data, so they can be found and reused by other Android applications. Therefore, these services can be used by programmers to develop Android applications.

Main services included in the Android application Framework are:

- Activity Manager: It controls every aspect of Android application’s lifecycle and stack operations.
- Content Providers: Enables Android applications to publish and share data with other Android applications.
- Resource Manager: Provides access to non-code embedded resources such as color settings and provisions of user interface.

- Notifications Manager: Allows Android applications to display warnings and alerts on the user.
 - View System: It is a scalable set of facets used to create user interfaces of Android applications.
 - Package Manager: This is the system whereby Android applications have the opportunity to learn about other Android applications installed on the device.
 - Telephony Manager: Provides information for telephony services available on the device, such as status and information of subscribers, on the Android application
 - Location Manager: Provides access to location services allows an Android application to receive updates when the location changes.
- Applications

At the highest level are located all the Android applications. It includes both the pre-installed Android applications with the specific implementation of Android (for example internet browsers and email Android applications) and all those that were installed by the user after device's acquisition.

The file format of Android applications is the APK format. Basically it is a zip file containing all the necessary files that consists an Android application. Normally contained files such as *classes.dex*, *res/*, *resources.arsc*, *AndroidManifest.xml*, *libs/*, *assets/*, *META-INF/* [12].

Specifically, *classes.dex* file contains the compiled code of Android application in Dex bytecode format. Starting from the Android 5.0, introduced the ART runtime replacing Dalvik, these files are transformed into OAT files using the ahead-of-time (AOT) compiler at the Android application installation stage and placed in the data partition device.

In *res/* folder is contained the majority of xml resources and drawables in folders with many qualifiers, like *-en* and *-de* for multilingual support or *-large* and *-medium* for different screen sizes. It is worth noting that each XML file located in this folder has been transformed into a

more compatible, binary representation during compile process. For this reason, these files cannot be opened with a text editor through APK file.

In *resources.arsc* file, resources and identifiers are compiled and flattened. Normally these stored uncompressed into an APK file in order to be easily accessible during execution. Compressing this file with a manual way could be considered a positive practice, but actually this is not a good idea. This occurs mainly for two reasons. First, Play Store compresses any data for transferring, and second, having the compressed file in an APK will be wasting system resources (RAM) and performance (especially the starting time of the Android application).

Another file contained into the APK is the manifest file. Basically it is an xml file that contains basic information about the application on the Android build tools, on the Android operating system and on Google Play. Like any xml file, it is transformed in binary format during writing step. Play Store uses information from this file to determine whether an APK file can be installed on a device, checking the permissible densities or screen sizes and the available hardware and features such as a touch screen. After writing phase, for checking these entries in the manifest file can be used aapt tool from Android SDK using the command:

```
$ aapt dump badging your_app.apk
```

Each native library (*.so file) will be placed in subfolders named after ABI-Android application Binary Interface (CPU architecture, e.g. x86, x86_64, armeabi-v7a) aimed below *libs/* folder. Normally, they are copied from the APK file in *\data* partition during the installation. However, once an APK located on a device and does not change by its own, this essentially doubles the space required for each native library.

Assets/ folder is used for storing assets that will not be used as Android-type resources. Usually in this folder are contained font-style files or game data such as levels and plots, as well as any other Android application file that the user wishes to open directly as a file stream.

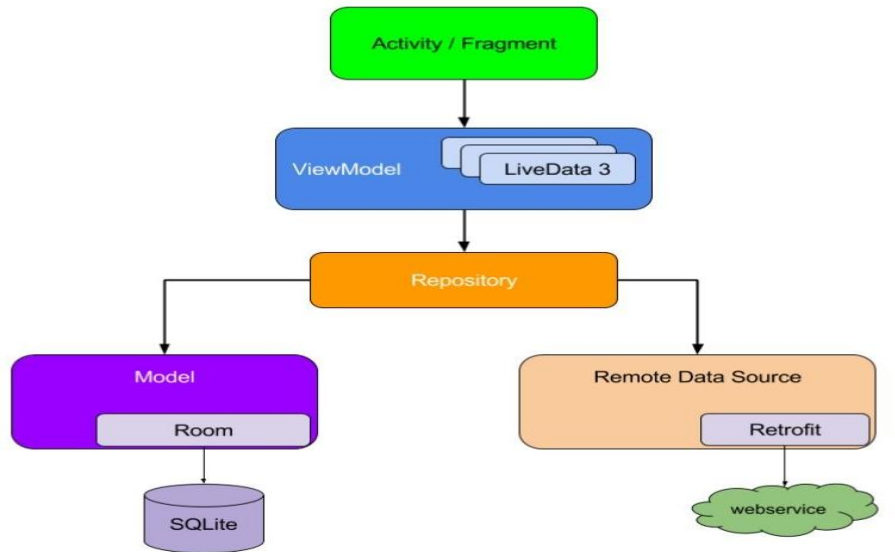
META-INF file is located into signed APK files and it contains a list of all files in the APK with their signatures. At this moment, signature method on Android verifies one by one signatures against the uncompressed file contents of the archive.

This fact has some interesting consequences. Since each entry in a compressed (.zip) file is stored separately, it is concluded that the compression level of individual files can be changed without having to sign it again. Nevertheless, if any file from the archive removed after its signature, then the signature's verification will fail.

Something else worth mentioning about the way a signed APK file is generated is that zipalign tool used in the final stage of the build. If contents of the archive are manually changed, they should be signed again and then should be used zipalign tool before the APK will be uploaded on Play Store.

Architecture components are a new collection of libraries which assists design robust, testable, and maintainable Android applications. In other words, these help overcoming common problems related to configuration changes, memory leaks and developing Android applications that can easily be tested, while at the same time the architecture is maintained.

After several proposals the final version of Android architecture was agreed. This includes the lifecycle, ViewModel, LiveData, Lifecycle - ViewModel - LiveData and Room components (Figure 3). These can be used either together or separately [13, 14].



<https://developer.android.com/jetpack/docs/guide>

Figure 3: The final Android architecture

Concisely, ViewModel component is designed to store and manage UI-related data so that the data survive configuration changes such as screen rotations. LiveData is a component which holds the value (data holder) and allows observing for changes on this value. It is designed to hold the data of ViewHolder but it can also be used for data sharing in the Android application. Lifecycle — ViewModel — LiveData component is referred in the fact that UI components, such as TextView or RecyclerView, observe LiveData, which, in turn, observes the lifecycle of an Activity or Fragment using a LifecycleObserver. Finally, Room provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

4. Android Forensics

A known branch is forensics or otherwise forensic science. This is the science applied to criminal and civil law during criminal investigation, emphasizing criminal side. This research is based on legal norms of acceptable evidence and criminal procedure.

Digital forensics is sector of forensics. This branch includes recovery and inspection of data found in digital devices, often in relation to cybercrime. The increased utilization of mobile devices and the multitude of offered possibilities have led to the creation and continuous development of a new sub-sector of digital forensics. It is called Mobile Device Forensics concerning the recovery of digital data or data from a mobile device under forensic conditions. Usually the term mobile device refers to mobile phones. However, it can be associated with any digital device that has both internal memory and the ability to communicate, including PDA devices, GPS devices and tablets.

This thesis focuses on forensics on Android applications; therefore a more extensive reference to this sector is necessary. This is a relatively new sector which dates from the early 2000s and late 1990s. The rapid growth of mobile devices, especially smartphones, in combination with their capabilities, consist a subject of research. The inability of testing these devices with existing techniques referring to computers led to the creation of a new branch.

4.1 Android storage possibilities

The goal of forensic analysis is to extract data from a device. Such investigation in order to be effective, a well-known background of device's filesystem (i.e. where data is stored, partitions) as long as knowledge on different file extensions, is necessary. Having this knowledge, the analyst has the ability to make a substantiated decision where to search for the necessary data and what techniques can be used for their extraction.

For this reason, a basic knowledge of the file hierarchy of the investigated device's operating system is judged useful. It concerns the way that the operating system, Android in our case, organizes data in files and folders. Researcher must have root access on the device in order to access all system's files. A list of directories which exist in Android described briefly as follows. [15, 16]:

- Acct: This is the starting point for the control group (cgroup) which is responsible for managing user accounting.
- Cache: In other words, temporary memory. It is the directory (*/cache*) where Android stores data and Android application's components that are often accessible. Deleting data from current directory is not affecting user's personal data. Also, in this directory, the */lost+found* folder is located, which contains recovered files (if any) in case of corruption of the filesystem, such as incorrect removal of the SD card. Therefore, cache may contain important information for criminal investigation, such as pictures, browsing history and other Android application's data.
- d: A symbolism of *\sys\kernel\debug* folder. It is used to boot debugfs filesystem and to detect errors in the kernel.
- data: This is the starting point of */dev/block/mtdblock1* partition. Some important files contained in this folder are:
 - dalvik-cache: This folder contains log files which may be useful during examination, according to the existed requirements.
 - data: The *\data\data* partition contains private data of all Android applications. The majority of users' data are stores in this folder, thus the significance of it in forensic terms during an inspection.
 - dev: This directory contains dedicated device's files for all devices. This is the starting point of tempfs filesystem. This filesystem defines available devices to Android applications.

- **init:** During Android kernel booting, init program is running. This program appears under the specified folder.
- **mnt:** The specified directory serves as the starting point of all filesystems, internal and external SD cards and so on.
- **proc:** This is the starting point of procfs filesystem, which provides access to kernel's data structures. Several programs use this directory as the source of the information they need. It contains files with useful information for procedures.
- **root:** This is the home directory for root access. This folder can only be accessed if the device is rooted.
- **sbin:** Contains binary files for several important daemons. From a forensic point of view and during an inspection, this folder is not considered a key folder.
- **misc:** As the name indicates, this folder contains information about various settings. From this folder can be accessed information related to hardware settings, USB settings, and so on.
- **sdcard:** This partition contains data from either removable or not SD card that exists on the device. Every Android application which is installed on the device and has permission to write to the external storage (`WRITE_EXTERNAL_STORAGE`) can create files and folders in this location. However, some default folders, such as `android_secure`, `Android`, `DCIM` are found in almost all mobile devices.
- **Digital Camera Images (DCIM):** The default folder of Android operating system in which user's images, video and thumbnails (cache) are stored. User's photos are saved under `\DCIM \Camera` folder.
- **system:** This folder contains libraries and archives associated with the system. These could be either on binary or other format. Preinstalled Android applications which come with the device are installed in this directory.

Furthermore, in this folder there is also the app/ subfolder which contains system Android applications and preloaded Android applications.

4.2 Filesystem

As already mentioned, Android operating system uses Linux kernel. At the top of the filesystem there is an abstraction layer called Virtual File System (VFS). This layer defines an interface between kernel and a specific filesystem. Supported filesystem types vary on Android devices. Thus, in order for our research to be successful and effective, it is important for analyst to possess adequate background knowledge on the involved device. Such knowledge comprises elements of device's filesystem, unix commands and so on, in order to used later on during this investigation with the most appropriate way. Common filesystems are the following:

- EXT (Extended File System): This is the standard type used by most Linux systems. From December 2010 it began to be used in Android devices.
- YAFFS2 (Yet Another Flash File System version 2): This is the default AOSP (Android Open Source Project) flash filesystem for the kernel's version 2.6.32. YAFFS2 is not supported in new kernel's versions, as, in our case, the kernel's version was 3.10.49. For this reason, it does not appear in the source tree. However, specific suppliers of mobile devices might still support YAFFS2.
- exFAT (extended File Allocation Table): Microsoft's proprietary filesystem, which requires license purchase in order to be used. For this reason this is not part of standard Linux kernel. However, some manufacturers provide support to Android operation system for this filesystem.
- F2FS (Flash-Friendly File System): This filesystem was introduced by Samsung in 2012 as an open source Linux filesystem.

- JFFS2 (Journalling Flash File System version 2): The successor of JFFS filesystem. This is the default flash filesystem for AOSP kernels beginning from Ice Cream Sandwich version.

5. Methodology

For the implementation of the technical part of this thesis specific actions carried out. For better understanding of the process followed, a more detailed explanation is described. Initially, a decision has to be made on the investigation's approach. The first approach involved use of special platforms for mobile forensics, as UFED, XRY and Oxygen Forensics Detective. However, the difficulty in obtaining these tools led to the second method which is described in detail as follows.

First and foremost a crucial step of preparation concerns device's rooting. By the term "root" is mentioned the process by which a superuser's account is added in a mobile device, proportional "Administrator" account in Windows. Essentially user has privileged access' rights to the system.

Rooting is often performed in order to overcome some limitations which exist in devices. In this way, rooting enables modification or replacement of installed Android applications and system settings. Moreover, it allows the execution of specific Android applications that require administrative privileges or other functions which would otherwise be inaccessible to a regular user. In addition, rooting devices might allow complete removal and replacement of operating system of the device, usually with a recent version of the operating system [17].

In this thesis, a Samsung Galaxy J5 mobile device was used for applications' inspection. A prerequisite for rooting is to enable "Developer options" and then activate "OEM unlock" and "USB Debugging" options. Firstly, recovery.tar.md5 [18] and SuperSU-v2.82.zip [19] files have to be downloaded and stored in device's memory. Consequently, in order to activate the download mode on the device, on/off, menu and sound reduction buttons were pressed at the same time while the device was powered off. Moreover, the mobile device was connected to a laptop in order for us to use Odin3 v3.12.10 tool [20]. At AP field of this tool, we filled we filled the path to the recovery.tar.md5 file and pressed the Start button. Once this process has been completed and after the necessary device restart, "Swipe to Allow Modifications" and "Install"

options are selected (in screen's menu). Furthermore, SuperSU-v2.82.zip file and "Reboot System" option should be selected. Finally, in order to verify the successful rooting, "Root Checker" Android application was downloaded from Google Play Store.

It is worth noting an error came up after device's rooting, during the third time in which the device was powered on. Specifically, an error screen was prompted labeled "Custom binary blocked by FRP" during a power on attempt. Consequently, an online research was conducted in order to identify the cause. It turned out, that it was a common issue of rooted mobile devices. However, the solution was not so simple. The original firmware of the device was needed to be installed again, and repeat the rooting procedure.

After a successful rooting, "Root Browser" Android application was used which is available on the Play Store. This Android application allows us access to rooted device files. An important folder is data/data folder which is inaccessible from ordinary users.

A copy to a computer has to be made in order for the abovementioned files (rooted or not) to be analyzed. Tools such as Explorer Droid or Android Device Monitor DDMS using Android Studio are capable of previewing and copying these files via computer [21]. However, after using several times these tools creates problems in recognizing mobile device and showing up the files. When the mobile device was connected to a computer, were not appeared neither its normal name nor the included files. To solve this problem some configurations on the mobile device have to be made. The user had to select "Options for Developers" tab through "Settings" option, then to choose "Revoke USB debugging authorization" option and press "OK" in pop-up windows. The result of these actions was the recognition of the mobile device of each tool by displaying its name and its files. Then, user has the ability to select folders or files and extracting them on the computer was given.

It is worth noting that there is another simple way to export data, which does not require the use of a tool. Essentially, the necessary files can be copied to mobile device's memory card and then to a computer for analysis. Although the simplicity of such approach is greater than the first one, lacks on effectiveness. It was observed that when a folder was selected, the included

system files cannot be copied directly. Each one of those should be separately selected for transferring to the memory card. Ultimately, this method has the same result as the first one.

In some cases, we had to export data several times, as in My COSMOTE Android application, in order to check again some files. This step was implemented in cases, where the results were not as expected. The Android application was used for long period of times and the data of it were exported regularly. Such an approach had to be made in order for the research results to be reliable.

It is really important to point out that during our research; the test device did not turn off. As a result, the data from RAM (Random Access Memory) will not be deleted and the inspection could be carried on.

Briefly, a summary of methodology's steps are presented as follows (Figure 4):

1. Mobile device rooting
2. Use of each Android application for long period of time
3. Android application data extraction
4. Android application data analysis
5. (If needed) Repetition of steps 2 to 4 as many times as required
6. Conclusions export

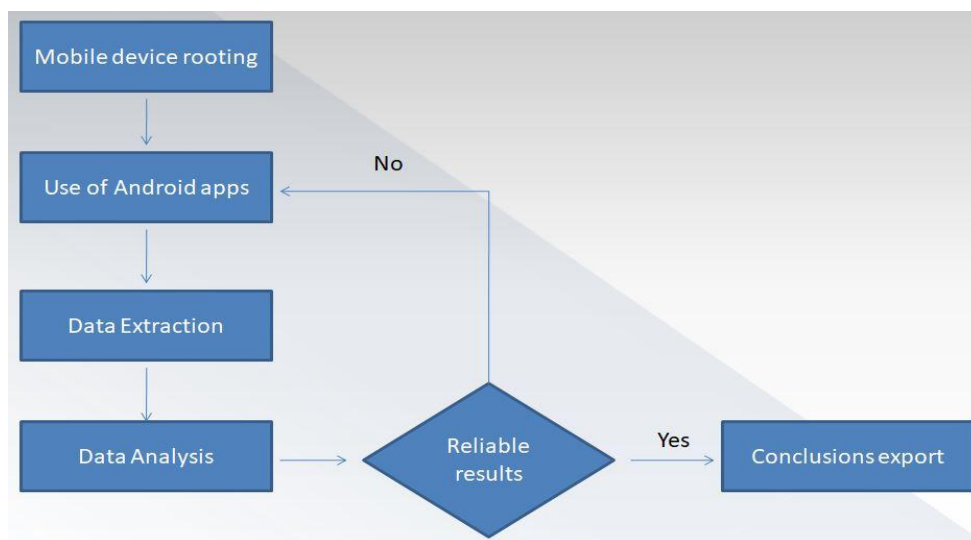


Figure 4: Flow chart of methodology

6. Android application Scenarios

6.1 My COSMOTE Android application

The first experiment concerns My COSMOTE Android application of COSMOTE MOBILE TELECOMMUNICATIONS S.A., aimed at users either landline or mobile telephony of this company and it is available free on Play Store. The application offers the users the opportunity to manage their accounts via their mobile device. They can be informed for issues relating to their accounts and for available services that the company offers. Moreover, users are able to activate any service upon their will and change their billing program.

In regard with mobile telephony, users can manage their mobile connection getting information about available airtime, messages and mobile data, without any password necessary. Moreover, users can pay their bill and be informed about its amount and its payment date and briefly about other important information about their calls. They can share airtime, messages and mobile data with other Cosmote numbers, to activate and deactivate services and to request information about billing programs or select a new one.

In landline telephony, users can be informed about their monthly bill and pay it. Also they can be informed for the enabled services.

Each user has the ability to contact Customer Service Center either through an online form or making a phone call by clicking on the relevant shortcut and search for the nearest physical store.

Signing in to My Cosmote Android application can be achieved either by typing a username/email address and a password after prior registration via email or using the SIM card. In our case, the second way was chosen. The mobile phone number which will be connected to the Android application was filled in. Consequently, a SMS with verification code was sent to the above phone number. When it was received, the code automatically placed on the appropriate field inside the Android application. Upon completion of the aforementioned

actions, the Android application was fully synchronized with information related to the declared phone number. However, these actions are only required on the first signing in to the Android application. For each subsequent connection this synchronization is done automatically.

Data of this application are saved in `\data\data\gr.cosmote.myaccount.widget` folder, in which contained the subfolders appeared in Figure 5.

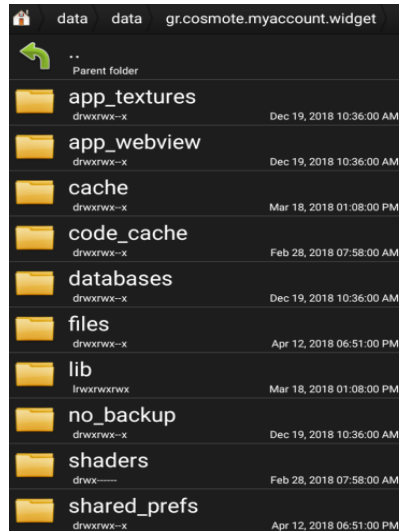


Figure 5: Structure of `\databases` folder

Into `\databases` folder are located the `application.db`, `google_app_measurement.db` and `google_app_measurement_local.db` databases. In order to be gained access to every data stored on those databases, DB Browser for SQLite tool [22] was used. Selecting tab “Database Structure” shows the tables and columns of the selected database (Figures 6). The following figures illustrate the `application.db` database.

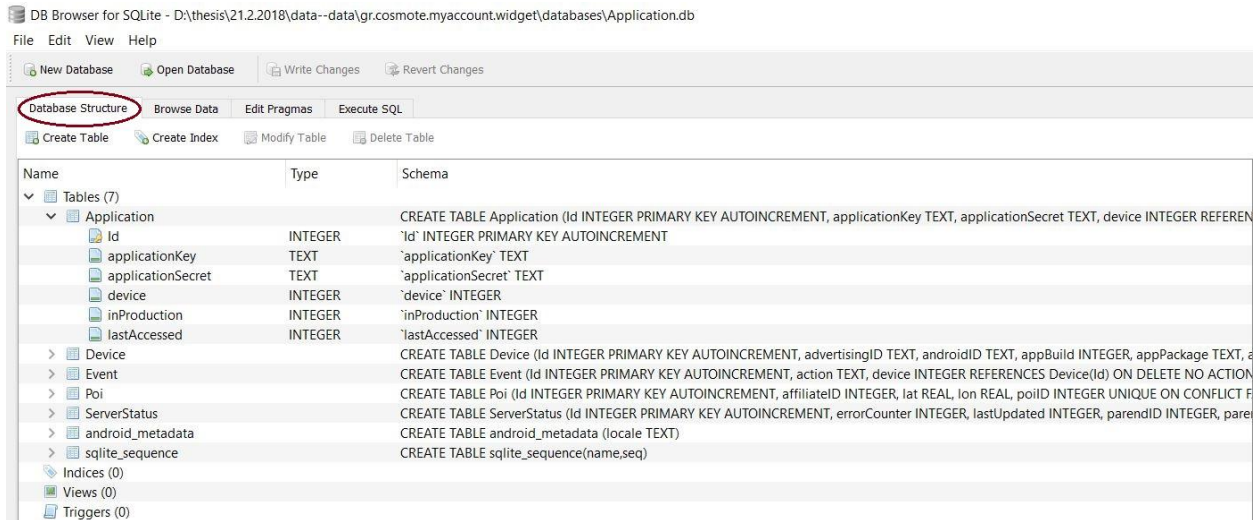


Figure 6: Structure of application.db database via DB Browser for SQLite

In the *Browse Data* the user can preview the columns and any data related with the selected database table (Figures 7 and 8).

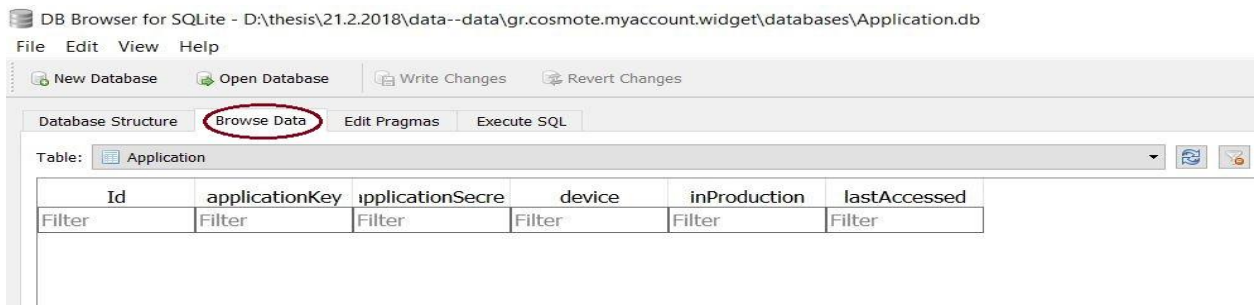


Figure 7: "Browse Data" tab selection/preview

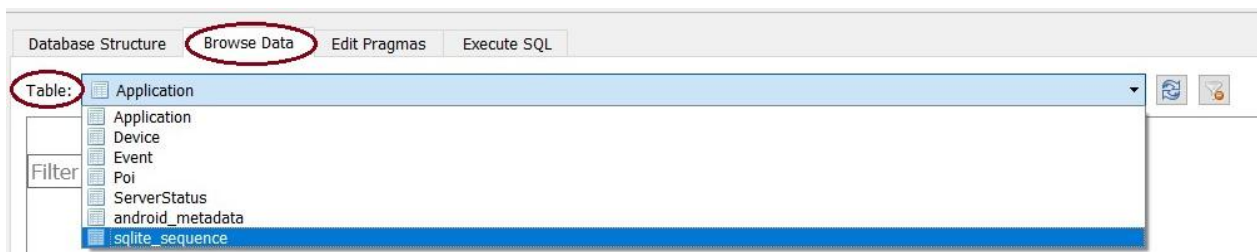


Figure 8: Table selection on "Browse Data" tab

In our first attempt to connect to application's database and after one day of user, it was discovered that none relevant information was stored. Our first assumption was that probably the Android application needs more time in order to store any data. Keeping this in mind, we continue the use of the application for several days while checking if any data has been logging. Our findings, indicate, that nothing was stored in these databases after a month of continues use.

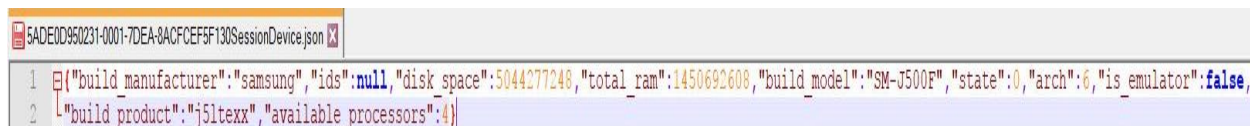
On the other hand, plethora of findings was found on files into `\files` folder, concerning network level information. The default.realm file was opened with Realm Studio program [23]. In this file was found that the mobile phone number which is assigned to the Android application, was stored unencrypted. Application's classes with relevant information have been found in this file. However, no additional user's data were stored (Figure 9).



Figure 9: Open default.realm via Realm Studio

In the same folder is located *DATA_ServerControlledParametersManager.data.gr.cosmote.myaccount.widget* file, which stores information about servers and sites with respective satellites used for location search such as *www.gstatic.com*. Also, user's country and language were stored. In this case was stored "el" for language and "GR" for country.

Furthermore, in this particular folder some hidden system files were stored. Specifically, there is *\.Fabric* folder, wherein are contained *\com.crashlytics.sdk.android.crashlytics-core*, *\com.crashlytics.sdk.android.answers* and *\io.fabric.sdk.android.fabric* subfolders. From the first folder is already known the Session ID which contained in the name of each file. In this case, Session ID is *5A68895B039D-0001-3B82-8ACFCEF5F130*. Therefore, files' names were consisted of the session ID, "Session" and one more word depending on the file's content. In *5A68895B039D-0001-3B82-8ACFCEF5F130BeginSession.json* file was contained unencrypted the identifier. In *5A68895B039D-0001-3B82-8ACFCEF5F130SessionApp.json* file was contained information about the Android application, such as its version (7.0.2), the UUID of the installation and the api key which was unencrypted. In *5ADE0D950231-0001-7DEA-8ACFCEF5F130SessionDevice.json* is contained information about mobile device, such as manufacturer, product number, device model and information about memory and processors (Figure 10).



```
5ADE0D950231-0001-7DEA-8ACFCEF5F130SessionDevice.json
1 [{"build_manufacturer": "samsung", "ids": null, "disk_space": 504427248, "total_ram": 1450692608, "build_model": "SM-J500F", "state": 0, "arch": 6, "is_emulator": false,
2  "build_product": "j5ltexx", "available_processors": 4}]
```

Figure 10: Preview of 5A68895B039D-0001-3B82-8ACFCEF5F130SessionApp.json file

In *5ADE0D950231-0001-7DEA-8ACFCEF5F130SessionOS.json* file there was information about operating system, its version and whether the mobile device was rooted or not.

In the same folder *\log_files* folder was included. There were temporarily (.temp) files which record user's actions. Great interest indicates the fact, when the user registers and subsequently connects on the Android application for the first time; a record in the corresponding log file has been made which stores the mobile phone's number (Figure 11).

```

1  [REDACTED],214 ?/LoginWithCredencialsActivity onCreate
2  [REDACTED],757 ?/LoginWithCredencialsActivity onResume
3  [REDACTED],2751 ?/LoginWithCredencialsActivity onPause
4  [REDACTED]GS2782 ?/HomeActivity onCreate
5  [REDACTED]DC23226 ?/d onCreate
6  [REDACTED]GS3275 ?/HomeActivity onResume
7  [REDACTED]DC23275 ?/d onResume
8  [REDACTED].4139 ?/LoginWithCredencialsActivity onDestroy
9  [REDACTED]"4454 ?/SanitizePhone 30694 [REDACTED]
10 [REDACTED]"#4455 ?/SanitizedPhone 30694 [REDACTED]

```

Figure 11: Preview of log files

The third folder is *com.crashlytics.settings.json* folder which holds information about user's configuration on the Android application. For example, if user agrees on data collection and report sending in case of an error. These settings are presented in more details in Figure 12. Collected data which are about to be sent are stored encrypted in a .tap file into the second folder. Both data concerning the mobile device and the particular session are stored (Figure 13).



```

com.crashlytics.settings.json
1  {"settings_version":2,"cache_duration":7200,"features":{"collect_logged_exceptions":true,"collect_reports":true,"collect_analytics":true,"prompt_enabled":false,
2  "push_enabled":true,"firebase_crashlytics_enabled":true},"analytics":{"url":"https://e.crashlytics.com/spi/v2/events","flush_interval_secs":120,"max_file_count_per_send":1,
3  "track_custom_events":true,"track_predefined_events":true,"track_view_controllers":false,"flush_on_background":true,"max_byte_size_per_file":40000,"max_pending_send_file_count":20,
4  "sampling_rate":1,"forward_to_google_analytics":true,"include_purchase_events_in_forwarded_events":true},"beta":{"update_suspend_duration":1800,
5  "update_endpoint":"https://api.crashlytics.com/spi/v2/platforms/android/apps/gr.cosmote.myaccount.widget/beta_update_check"},
6  "app":{"identifier":"gr.cosmote.myaccount.widget","status":"activated","url":"https://api.crashlytics.com/spi/v1/platforms/android/apps/gr.cosmote.myaccount.widget",
7  "reports_url":"https://reports.crashlytics.com/spi/v1/platforms/android/apps/gr.cosmote.myaccount.widget/reports",
8  "ndk_reports_url":"https://reports.crashlytics.com/sdk-api/v1/platforms/android/apps/gr.cosmote.myaccount.widget/minidumps","update_required":false},
9  "session":{"log_buffer_size":64000,"max_chained_exception_depth":16,"max_complete_sessions_count":4,"max_custom_exception_events":3,"max_custom_key_value_pairs":64,
10 "identifier_mask":255,"prompt":{"title":"Send Crash Report?","message":"Looks like we crashed! Please help us fix the problem by sending a crash report.",
11 "send_button_title":"Send","show_cancel_button":true,"cancel_button_title":"Don't Send","show_always_send_button":true,"always_send_button_title":"Always Send"},
12 "expires_at":1524504662526}

```

Figure 12: Preview of com.crashlytics.settings.json file

```

session_analytics.tap
aa23c8f","osVersion":"6.0.1\J500FXXU1BQE2","deviceModel":"samsung\SM-
J500F","appVersionCode":"1712190","appVersionName":"7.0.19","timestamp":
1515771932826,"type":"STOP","details":
{"activity":"gr.cosmote.myaccount.widget.Activities.LoginWithCredentialsActivity"}},
{"appBundleId":"gr.cosmote.myaccount.widget","executionId":"f51b5e7c-a819-4627-b4d8-
c87811bfdb57","installationId":"02e4566ef88d476a9824323245f1eb69","androidId":"6f37
25bbe98c7942","advertisingId":"71f2eb5b-6594-4203-9b62-
c3f81e5d60ac","limitAdTrackingEnabled":false,"buildId":"75fc35fc-8d2c-4b9a-9b5d-6323b
aa23c8f","osVersion":"6.0.1\J500FXXU1BQE2","deviceModel":"samsung\SM-
J500F","appVersionCode":"1712190","appVersionName":"7.0.19","timestamp":
1515771954939,"type":"PAUSE","details":
{"activity":"gr.cosmote.myaccount.widget.Activities.HomeActivity"}+
{"appBundleId":"gr.cosmote.myaccount.widget","executionId":"f51b5e7c-a819-4627-b4d8-
c87811bfdb57","installationId":"02e4566ef88d476a9824323245f1eb69","androidId":"6f37
25bbe98c7942","advertisingId":"71f2eb5b-6594-4203-9b62-
c3f81e5d60ac","limitAdTrackingEnabled":false,"buildId":"75fc35fc-8d2c-4b9a-9b5d-6323b

```

Figure 13: Preview of session_analytics.tap

The following analyzed folder was `\app_webview` folder. In `Cookies` file was observed storing information about Android application's cookies (Figure 14). Among several important elements is the value of cookies, the date and time of their creation and expiration in utc format, and the field that informs whether the cookie has expired or not.

creation_utc	host_key	name	value	path	expires_utc	secure	httponly	last_access_utc	has_expires	persistent	priority	encrypted_value	firstpartyonly
13158153522560938	xmas.cosmote.gr	xmasBrowserId	PYQKSF...	/	0	0	1	13158153595273168	0	0	1		0
13158153522744574	.cosmote.gr	visit_incap_241939	pe5KYXv...	/	13189616413744574	0	0	13158153595273168	1	1	1		0
13158153522744844	.cosmote.gr	incap_ses_374_241939	6zg+RU...	/	0	0	0	13158153595273168	0	0	1		0
13158153616883414	xmas.cosmote.gr	xmasId	Rx92Hs...	/	13158240016883414	0	1	13158153616883414	1	1	1		0
13158153617022784	.cosmote.gr	_ga	GA1.2.21...	/	13221225617000000	0	0	13158153617022784	1	1	1		0
13158153617027636	.cosmote.gr	_gid	GA1.2.17...	/	13158240017000000	0	0	13158153617027636	1	1	1		0
13158153617054340	.cosmote.gr	_gat	1	/	13158153677000000	0	0	13158153617054340	1	1	1		0

Figure 14: Preview of cookies table

The `metrics_guid` file stores the GUID (global unique identifier) in the 8-4-4-4-12 format. Basically it is a number of 128 bits which is used to identify information on computer systems.

In `\code_cache` directory was included the `com.android.opengl.shaders_cache` file. This file contains information about the used OpenGL ES API. Specifically, EGL library was used, as mentioned at section number 1.6.2.


As previously mentioned, another forensically important folder is `\shared_prefs` folder. In this particular folder, xml files for user's actions were found. These files were consisted of two elements, which are name and value elements. Parameter "name" contains user's action, while the parameter "value" takes different values as a response to each action. More specifically, some user's actions that are stored are presented as follows:

- *dsq.cosmoteprepaid.userHaveSeenTutorialGuide.xml*: If user has previewed Android application's guide with the basic steps that appears when they first connected to the Android application (value = true if they have previewed it, otherwise false)
- *dsq.userAppLogIns.xml*: The number of times that users have been logged in to the Android application (either automatically or not)
- *dsq.userAppRatingPostponeDate.xml*: The last timestamp that the user postponed the evaluation of the Android application
- *dsq.userAppRatingPostponeDateLimit.xml*: The maximum number of times/dates user can postpone rating the Android application
- *externalProfilingConsent.xml*: If user has given consent for profiling by third parties (external) or an external user
- *hasSeenTerms.xml*: If user has read and accept the terms of the Android application's use
- *internalProfilingConsent.xml*: If user has given consent for profiling by the Android application itself (internal) or an internal user
- *dsq.cosmoteoneapp.userD4UEnableInform*: If user is able to preview "Cosmote Deals For You" offers
- *lastDealsForYouUpdate.xml*: The date of last offers' update. These offers addressed to the user depending on the category to which they belong

- *payBillFixedTerms.xml*: If user has read the conditions for payment of landline telephony bills
- *payBillMobileTerms.xml*: If user has read the conditions for payment of mobile phone bills
- *UserHaveSeenTutorial.xml*: If the user has previewed detailed (step by step) guide of the Android application

Device's information was found in a folder with no particular interest from a forensic perspective. This is `\shaders` folder, which is responsible for the colors that designed objects will have. It was found that each file's content begins with information for mobile device (j5lte), informing about its model (j5) and the standard wireless connection it can achieve (lte).

However, two months later making an update to 7.2.2 version and checking the Android application again was found that there were some changes. In `\databases` folder was added a new file named "*MOEInteractions*". Seven tables were included in this file, one of which had data stored. This was *USERATTRIBUTE* table, in which was stored the UUID of the Android application (Figure 15).

Table:  USERATTRIBUTES

	<code>_id</code>	<code>attribute_name</code>	<code>attribute_value</code>
	Filter	Filter	Filter
1	1	APP_UUID	483bc024-8f35-44a3-bbd0-2a20ba2e7b92
2	2	legal_consent	false

Figure 15: Preview of *USERATTRIBUTES* table

Moreover, another change was observed in `\files` folder. Although, the *default.realm* file is again located at the same place as before the version update, this time the file was encrypted and had different suffix (`.enc`).

Last folder in which new files were added was `\shared_prefs` folder. Firstly, there was *consentStatus.xml* file which had the format of name and value as other xml files. This file indicates whether or not user gave consent in order to process their personal data. Based on

this action, “*value*” parameter takes true or false values, respectively. Also, *lastContactRenew.xml* file was added which stores the last timestamp that users updated their contact lists. Another file which indicates enough interest was *pref_moe.xml* file. In this file was saved useful information such as the Google Advertising ID (GAID), which was created by Google to identify unique users, the UUID of the Android application that has been referred previously and the last timestamp that Android application’s data have been synchronized. Finally, *userHasSeenForceUpdate.xml* file had been added, which indicates whether user had updated the Android application or not.

Furthermore, in both applications’ versions tested, no data were stored in *application.db* database. Finally, an internet connection is mandatory in order for the application to operate. In any different case, a display with the company logo and a corresponding message will appear on screen in order to inform the user about.

6.2 What's Up Android application

The second experiment concerns the What's Up Android application of COSMOTE MOBILE TELECOMMUNICATIONS S.A., which aims at mobile phone users of the company who are registered at "What's Up" program. The Android application is available free on Play Store. Users can preview their balances, renew their airtime, activate packages, be informed about offers and earn gifts.

When a user is connected for a first time on the Android application, an optional short guide wizard tour can be selected. Consequently, users have been asked to read and agree on the terms of use. Then, users will be prompted to fill the mobile number which they wish to connect the application. A message with confirmation code will be sent to the device and filled on the appropriate field automatically. At the end of this procedure, the account will have been synchronized with the Android application.

Now users are able to select their preferred services, check their balances, preview the available airtime, SMS and mobile data, top up and buy packages and bundles. Moreover, users can be informed on which other member of their contact lists owns a What's Up subscription in order to share directly deals with them and get gifts.

Android application's data are located in `\data\data\gr.cosmote.whatsup` folder, whose structure is shown in Figure 5. As observed, the structure of this Android application shares similarity with My COSMOTE Android application. However, they have a significant difference. As mentioned earlier My COSMOTE Android application requires the internet by displaying the data saved from the last online activity. Instead, What's Up Android application can run without using Internet connection displaying saved account's data. These data were referred to the last time user was signed in using Internet connection. Therefore from our investigation, we can conclude that My COSMOTE application can operate solely on server-side, in stark contrast with What's Up which can be utilized in both cases. The above difference and

similarities between these two Android applications are being proven by the analysis of the second one.

Firstly, the `\app_webview` folder was analyzed. Aforementioned, *Cookies* file stores information about Android application's cookies. Among several other, important elements are the value of cookies, the date and time of creation and expiration in utc format and a field which informs whether this cookie has expired or not. The structure of this file was the same as in previous Android application that was analyzed (Figure 13).

Another noticeable similarity was, is the existence of *metrics_guid* file in the previous folder. This file stores GUID (global unique identifier) with the same format, as at the previous Android application described. Moreover, `\code_cache` folder stores information about OpenGL ES API.

Users' activities are stored in the `\files` directory. The file name format of each included file is *rList-gr.cosmote.whatsup.Activities.ActivityName*. Keeping the same notion and by replacing the file's suffix ("*ActivityName*"), different files could be created. Specifically, there are files having the following values instead of "*ActivityName*":

- *.AskMobileNumberActivity*: It has been asked from user to fill in the mobile phone number which is associated with the Android application
- *.AskVerificationCodeActivity*: It has been asked from user to fill in the authentication code which had been sent in a text message to previously declared phone number
- *.BundleInfoActivity*: Purchased bundles' information has been updated and displayed on the screen
- *.ConsentTermsActivity*: Describes user consent on terms of Android application's usage
- *.DealsForYouActivity*: User has previewed concerning offers
- *.MainUserScreenActivity*: User has viewed the Android application's home screen

- *.MenuTermsAndConditionsActivity*: User has read the terms and conditions of the Android application
- *.MyCommunityActivity*: The Android application has access to user's contacts in order to inform them which contacts are linked to What's Up subscribers
- *.NewPackagesFragmentActivity*: New packages have been added
- *.StoreActivity*: User has visited "Store" of Android application's display
- *.StorePackageInfoActivity*: Information about packages which exist in the "Store"
- *.WebViewActivity*: User is able to display web pages as a part of their activity layout. It does *not* include any features of a fully developed web browser, such as navigation controls or an address bar. Using WebView class a web page is derived as a part of a client application.
- *.WhatsNewFragmentActivity*: It contains the new features added to the Android application

As in the previous Android application, in the above folder is also located *.Fabric* system folder. Within this temporary logs users' activity history are stored. There were also some *.json* (and the respective *.cls*) files, the title of which contains also the Session ID. In the given case, this ID is *5A9A604D01C1-0001-40C2-19BB625A33E9*. Therefore the existing files are *5A9A604D01C1-0001-40C2-19BB625A33E9BeginSession.json*, *5A9A604D01C1-0001-40C2-19BB625A33E9SessionApp.json*, *5A9A604D01C1-0001-40C2-19BB625A33E9SessionDevice.json*, *5A9A604D01C1-0001-40C2-19BB625A33E9SessionOS.json* and *5A9A604D01C1-0001-40C2-19BB625A33E9user.meta*. The first four files are respectively contained information for the current session, the Android application itself, device's operating system and device's characteristics. These files have been previously analyzed at the My COSMOTE Android application. However, in stark contrast with the previous app, another file was added to the specific folder which was *5A9A604D01C1-0001-40C2-19BB625A33E9user.meta* file. It contains the username in encoded format (Figure 16).

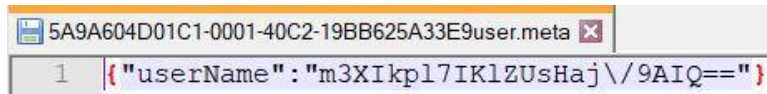


Figure 16: Preview of 5A9A604D01C1-0001-40C2-19BB625A33E9user.meta file

In this folder are also located *com.crashlytics.settings.json* file, which contains configurations that have been presented in detail at the previous Android application.

Finally, there is *WhatsUpDatabase.db* database which is located in *\databases* folder. Throughout our research was found out that the database is encrypted using SQLCipher; an open source extension to SQLite that provides transparent 256-bit AES encryption of database files [24]. Attempting to open it with DB Browser it was asked to insert the used passphrase. This passphrase is a user's input. Different approaches been made in order to crack it. First attempt was to type the verification code which was sent to user's mobile number. As it turned out, this effort was unsuccessful. Another approach, was to discover the location in which passphrase was stored. Possible key's storage positions were investigated [25], but none of them had a positive impact. Using a binary viewer program was confirmed that this database was not empty.

Next thought was the analysis of android application's code in order to find the requested key. Viewing the code was observed that some java classes were obfuscated [26] using ProGuard tool [27], having names that do not help in easy comprehension of the code, such as C2843a.java. The corresponding code snippet is presented as follows:

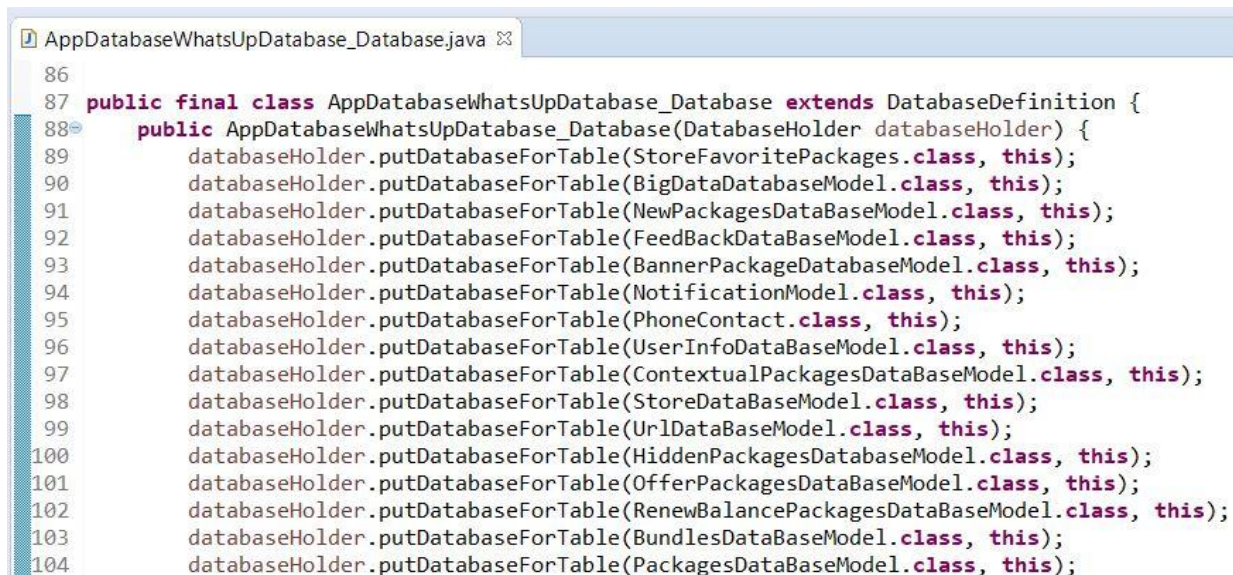
```
throw new RuntimeException("Could not read ZoneInfoMap. You are probably using Proguard wrong.", e);
```

To place the reader in context, obfuscation is the purposive act of creating source or machine code that is difficult for humans to understand. Programmers usually choose this method in order to prevent tampering, deter reverse engineering, or even as a puzzle or recreational challenge for someone reading the source code. For this reason it was difficult to understand

the whole code and discover the key or its storage position. The different ways with which the code was accessed and extracted are referred as follows in more details.

However, there is another way in order to be concluded which information was stored on user's device. The investigator should run the application without using internet connection and associate previewed information with the tables' fields in the database. These fields could be found viewing the application's source code, which is located in the `\data\app\gr.cosmote.whatsup-2` on apk format. Although extracting this file is an easy process, it should be taken into consideration that this file format is not accessible through Windows OS. For this reason, it was used an online apk decompiler [28] to java file (.jadx) in order to view and analyze (be viewed and analyzed) the android application's code on investigator's machine. Accessing .jadx file requires corresponding software, such as Android Studio or Eclipse Java Oxygen [29].

Implementing code analysis, details were found for the structure of application's database. This information is contained in the `\com\raizlabs\android\dbflow\config\AppDatabase\WhatsUpDatabase_Database.java` file. At the beginning of this file, reference is made to the sixteen tables which constitute this database (Figure 17). Analyzing further this java file, were found the columns which consist each one of these tables.



```
AppDatabaseWhatsUpDatabase_Database.java
86
87 public final class AppDatabaseWhatsUpDatabase_Database extends DatabaseDefinition {
88     public AppDatabaseWhatsUpDatabase_Database(DatabaseHolder databaseHolder) {
89         databaseHolder.putDatabaseForTable(StoreFavoritePackages.class, this);
90         databaseHolder.putDatabaseForTable(BigDataDatabaseModel.class, this);
91         databaseHolder.putDatabaseForTable(NewPackagesDataBaseModel.class, this);
92         databaseHolder.putDatabaseForTable(FeedBackDataBaseModel.class, this);
93         databaseHolder.putDatabaseForTable(BannerPackageDatabaseModel.class, this);
94         databaseHolder.putDatabaseForTable(NotificationModel.class, this);
95         databaseHolder.putDatabaseForTable(PhoneContact.class, this);
96         databaseHolder.putDatabaseForTable(UserInfoDataBaseModel.class, this);
97         databaseHolder.putDatabaseForTable(ContextualPackagesDataBaseModel.class, this);
98         databaseHolder.putDatabaseForTable(StoreDataBaseModel.class, this);
99         databaseHolder.putDatabaseForTable(UrlDataBaseModel.class, this);
100        databaseHolder.putDatabaseForTable(HiddenPackagesDatabaseModel.class, this);
101        databaseHolder.putDatabaseForTable(OfferPackagesDataBaseModel.class, this);
102        databaseHolder.putDatabaseForTable(RenewBalancePackagesDataBaseModel.class, this);
103        databaseHolder.putDatabaseForTable(BundlesDataBaseModel.class, this);
104        databaseHolder.putDatabaseForTable(PackagesDataBaseModel.class, this);
```

Figure 17: Tables of WhatsUpDatabase.db database

Next step was the use of android application without using internet connection. The investigator could match the information which was able to see on the android application with the columns of the above table. In this way, users would extract conclusions for saved information at a great percentage. To illustrate, the most important fields and the data which application saves, are referred.

In the *StoreFavoritePackages* table are located details about user's favorite packages. Package identifier ("*favoriteIdentifier*" column), the number of times ("*timesPurchased*" column) that user purchase the specific package and the last purchase ("*lastPurchase*" column) can be found in this table.

BigDataDatabaseModel table stores information about SIM cards and internet connection. It saves SIM card's provider name ("*carrierName1*" and "*carrierName2*" columns), mobile numbers which have been matched with these SIM cards ("*simNumber1*" and "*simNumber2*" columns), internet connection type and user's IP address. If connection type is wireless, then it is stored the service set identifier ("*ssid*" column) too.

Consequently, into *NewPackagesDataBaseModel* table is stowed information about new packages which have been added. Some of these are whether this package is presented to the user or not ("*presented*" column), if offer title exist ("*offerTitle*" column) and if it is an advantageous offer ("*hotOffer*" column), if this package is available only for specific users' categories ("*availableOnlyForUserLists*" and "*notAvailableOnlyForUserLists*" columns), the message which inform user for the existence of this offer ("*offerMessage*" column), it's price ("*price*" column), a short ("*shortDescription*" column) and a long description ("*longDescription*" column). Furthermore, here exists information about offer's recipient and whether use is already registered to this offer or they have chosen to unregister.

In the *FeedBackDataBaseModel* table are stored application's version ("*appVersion*" column), the label of the message when application asks from user to rate the application ("*message*" column), user's rating ("*rating*" column), the rating timestamp ("*timestamp*" column) and the universally unique identifier ("*uuid*" column).

BannerPackageDatabaseModel table saves information about the package which is presented on the application's banner. This table's columns are like *NewPackagesDataBaseModel* table's information columns.

A table which contains important information is *PhoneContact* table. This table is filled with data when user has given application their permission to access their contact list. It contains details for user's contact list. In particular, for each contact the above table holds its name ("*displayName*" column) and its phone number ("*honeNumber*" column), its URI for the thumbnail photo ("*photoThumbnailUriString*" column), its first name and its last name ("*FirstName*" and "*LastName*" columns), whether is a favorite contact or not ("*favoriteContact*" column) and how many time they have sent mobile data as a gift to their friends in order to receive them mobile data too ("*pareMoiraseTimes*" column).

In *UserInfoDataBaseModel* table is stored information about user's account and their device. Here can be found user's balance (*Balance* column), account's and balance's expiration date (*BalanceExpiration* and *AccountExpiration* columns), last time they update their data and their contact list (*lastDataRenew* and *lastContactsRenew* columns) or make a recharge (*LastRecharge* column). Other important details are the initial letters of user's first name and surname (*initials* column), their photo (*photo* column), device's identifier (*deviceId* column), MSISDN and IMSI numbers (*msisdn* and *imsi* columns).

ContextualPackagesDataBaseModel table contains details about contextual packages. Some of them are offer's identifier (*offerId* column) and type (*contextualOfferType* column), its date of expiration (*expirationTime* column), short (*shortDescription* column) and long description (*longDescription* column) and its price (*price* column).

Next table which is contained in the above database is *StoreDataBaseModel* table. It holds information about packages, bundles, offers and their activation method, user's search history and details about services in which they have already registered or willing to unregister from.

In *HiddenPackagesDatabaseModel* table are contained details about packages which were not showed up at the user at the specific moment in which investigation took place. Its structure is the same as *NewPackagesDataBaseModel* table.

OfferPackagesDataBaseModel table includes details about offer packages. Its structure is similar to the above table.

Another table is *RenewBalancePackagesDataBaseModel* table. Its content refers to the renewal of packages' balance. It includes packages' title, price, short and long description, activation method and so on. Most of its columns are the same with *NewPackagesDataBaseModel* table.

BundlesDataBaseModel table includes information about bundles. Important details are bundle's description (*descriptionFirst* column) and information about it (*informationTitle* column), the number of days which this bundle stays visible to user (*daysToStayVisible* column) and the date of its last update (*lastUpdate* column).

Finally, in *PackagesDataBaseModel* table can be found information about packages. Specially, inside this table columns such as *type* and *category* are defined. Moreover, user's available data for relevant packages (*availableData* column) as long as expiration data indicated by *dataExpiration* column. In addition, *NotificationModel* table is responsible for storing users' information which they will be accessible during application's operation. *UrlDataBaseModel* table stores the URL (*url* column) which the application utilize in order to retrieve individual users' information (using *key* column).

In *\shared_prefs* directory, files are located regarding to user's activities having the name-value format as in previous application. More specifically, the user's actions which are stored, are presented as follows:

- *dsq.userAppLogIns.xml*: The number of times that users have been logged in to the Android application (either automatically or not)
- *dsq.userAppRatingPostponeDate.xml*: The last timestamp that the user postponed the evaluation of the Android application

- *dsq.userAppRatingPostponeDateLimit.xml*: The last timestamp that the user postponed the evaluation of the Android application
- *gr.cosmote.whatsup.firstInstallationTime.xml*: The first time the application installed on user's device
- *gr.cosmote.whatsup.firstTimeAnimateBundles.xml*: Whether or not the user activates bundles for first time
- *gr.cosmote.whatsup.firstTimeWhatsNew.xml*: Whether or not the user views the added features on the application
- *gr.cosmote.whatsup.userConsentTermsChoice.xml*: Whether the user gave consent for full access on the required data
- *gr.cosmote.whatsup.userHasSeenConsentTerms.xml*: Describes user consent on terms of Android application's usage
- *gr.cosmote.whatsup.userHasSeenContextualPackage.xml*: The timestamp which user previewed contextual package
- *gr.cosmote.whatsup.userHasSeenTutorial.xml*: If the user has previewed detailed (step by step) guide of the Android application
- *gr.cosmote.whatsup.userHaveSeenTermsAndConditions.xml*: The user has read the terms and conditions of application's usage

6.3 My CU Android application

This is an Android application created and owned by the "Vodafone-Panafon Hellenic Telecommunications Company S.A". It is available free on Play Store for Android version 5 and above. The application aims to facilitate company's prepaid subscribers (CU users) in their daily needs of using the service (top up balance, monitor expenses and so on) and at the same time offer several exclusively offers.

A registration is required in order to use the service. Such action can be accomplished with two ways; either automatically by activating the mobile data on the mobile device or by filling out a registration form. In our case the second method was chosen. This procedure might appear typical, but it is important to be described for later on utilization in our research.

At the registration form, users asked to fill up all the necessary information in order to activate the service. Name, surname, Vodafone CU mobile number assigned, username chosen, email address comprise all the mandatory fields. Subsequently and by accepting the terms and conditions of this Android application, the verification stage takes place. The application connects to the company's main server in order to evaluate the user's details. If the verification is successful, a final activation code sent to the stated mobile number via text message.

To put theory into practice, "Ioanna" was filled as value at the name and surname fields of the registration form. Moreover, mobile number "694*****7", "Joanna" as username and the f*****f@gmail.com email address was typed. However, username was modified several times until entered a valid value due to username was already existed. Attempts such as "Joanna7", "Joanna71" and "71Joanna" were filled in as username's values. Finally, the last one was valid for username (Figure 18).

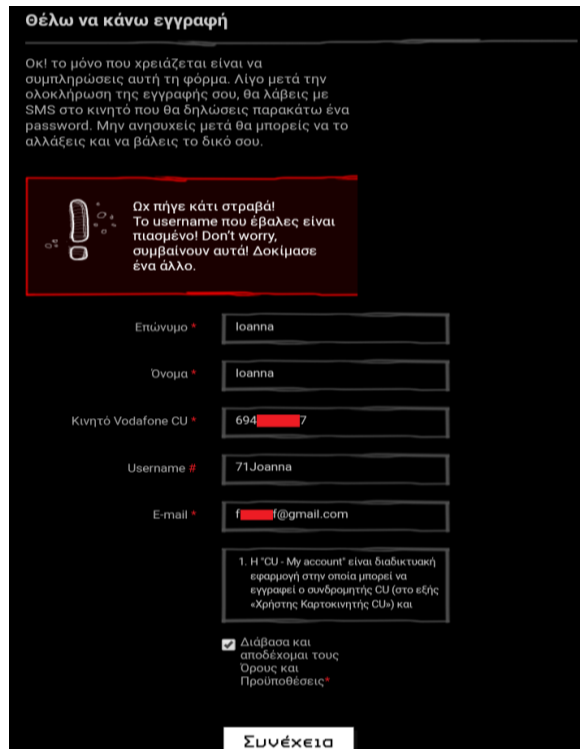


Figure 18: My CU user's data/credentials

The verification code was sent to the mobile was 7V297ZR. Consequently, user had to change this code by typing someone of their choice. It was entered a code with 13 digits, which had the form #@ ***** +. Instead of asterisks there are other characters that are hidden for security reasons.

After using the application for a couple of days, data was retrieved from `\data\data\gr.mobile.vodafone` folder. Although, the application is from a different company from the previous ones, the structure of the file shares similarities as presented at the figure as follows (Figure 5).

The `\app_webview` folder includes files which store important information for forensic investigation. Relevant information was stored at the Cookies table. Name and value of the cookie, creation-last access-expiration timestamp in utc format and the path in which cookie will be applied are some cookie attributes which was persisted. It is worth to point out that,

“encrypted_value” was null which indicated that all the information above was stored unencrypted (Figure 19).

Table: cookies New Record Delete Record


	creation_utc	host_key	name	value	path	expires_utc	secure	httponly	last_access_utc	has_expires	persistent	priority	encrypted_value	firstpartyonly
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	13166705659249350	www.vodafone.gr	VOP_USER_IDENTIFICATION	o5b68A...	/	13245545658249350	0	0	13166911909469958	1	1	1		0
2	13166911923932388	www.vodafone.gr	JSESSIONID	8653ca...	/portal	0	1	0	13166911923932388	0	0	1		0

Figure 19: Preview of cookies table

Similarly, with the previous examined applications, *metrics_guid* file saves the GUID (global unique identifier) in 8-4-4-4-12 format for information’s identification.

Web Data file is probably the most important file that was found during investigation of the specific Android application. It is a file acting as database, which holds many tables. From all the contents inside, *autofill* table is subject of significant interest as plenty of findings were discovered inside. The table comprises from data that user inputs to the application. Name, surname, mobile Vodafone CU number, username and password, which indicates all the information of registration form, was found unencrypted. Also, there are columns which were storing dates both for their creation and the last time they were used (Figure 20).

Interesting is the fact that all the entered username’s values have been stored in this table. As mentioned previously entered usernames were “Joanna”, “Joanna7”, “Joanna71”, with “71Joanna” to be the valid one. However, it was observed that among the stored values was also the username “Tzo”. This was the first username typed by the user, which was immediately replaced by “Joanna” value before sending as a registration element. Hence, it can be deduced that these possible values have been stored unencrypted, even if they were not sent as registration data.

Table:  autofill

	name	value	value_lower	date_created	date_last_used	count
	Filter	Filter	Filter	Filter	Filter	Filter
1	user.lastname	Ioanna	ioanna	1522437739	1522437925	6
2	user.firstname	Ioanna	ioanna	1522437739	1522437925	6
3	user.msisdn	694 [REDACTED] 7	694 [REDACTED] 7	1522437739	1522437925	6
4	user.username	Tzo	tzo	1522437739	1522437739	1
5	user.username	Joanna	joanna	1522437834	1522437834	1
6	user.email	f [REDACTED] f@gmail.com	f [REDACTED] f@gmail.com	1522437834	1522437925	5
7	user.username	Joanna7	joanna7	1522437859	1522437865	2
8	user.username	Joanna71	joanna71	1522437881	1522437881	1
9	user.username	71Joanna	71joanna	1522437925	1522437925	1
10	username	71Joanna	71joanna	1522438126	1522438149	2
11	passwordtext	#@[REDACTED] +	#@[REDACTED] +	1522438126	1522438126	1
12	passwordtext	7V297ZR	7v297zr	1522438149	1522438149	1

Figure 20: Preview of autofill table

In `\cache\org.chromium.android_webview\0370d314acea3e0d_0` file was stored information regarding to used certificates. These SSL certificates are digicert certificates.

As with the previous Android applications, the `\code_cache` folder stored information about the OpenGL ES API.

The ensuing investigated folder is `\databases` folder. It contains three databases, which are `google_analytics_v4.db`, `google_app_measurement.db` and `google_app_measurement_local.db` databases. The database of interest is the first database, because it is mentioned at Measurement Protocol [30]. This database stores unencrypted the Client ID (`cid`) and the tracking ID/web property ID (`tid`). The Client ID is required when the User ID (`uid`) is not determined on the user's request, where in this case `app_guid` acquires the value 0. Anonymously identifies a particular user, a device or a browsing. With regard to the internet, this identifier is stored as a cookie with a lifetime of two years. However, in case of mobile Android applications this identifier is randomly generated for each specific Android application's installation. The Tracking ID (`tid`) has the UA-XXXXXXXX-Y format. All collected data are associated with this identifier. Also, there is a column named `params` which contains the `"an= My% 20CU & aaid = com.android.vending & aid = gr.mobile.vodafone & av = 2.2.2"`

value. Specifically, *an* parameter is the abbreviation of Android application's name and *aiid*, *aid* and *av* are optional and respectively indicate the identifier for the installation of the Android application, the Android application's identifier and its version.

Subsequently, *\files* folder was analyzed. In *gaClientId* file was contained unencrypted the Client ID, which had been found previously in *google_analytics_v4.db* database. Also, there is *\.Fabric* system folder, whose structure has been analyzed in the first Android application. However, in this case, information was stored only in the *\com.crashlytics.sdk.android.crashlytics-core* folder and specifically in files related with session. Their structure remains the same as the corresponding files of the first Android application with the diversification of some parameter's values, such as session's value, to be the only difference.

Into *\no_backup* folder we came across with several databases with great interest. In particular, there was *\com.urbanairship.databases* subdirectory which four databases were found. As it is obvious, the urban airship platform was used. In *FcOQaQDGTl6gNhZ6G586_A_ua_analytics.db* database were locally stored events and derived periodically at the background. In *events* table, the event's identifier, size of the event, execution time and whether or not it runs on background of the Android application were saved. Also here were stored additional data such as the type of Internet connection (if any), the time zone, the type of user's notification (sound and vibration), the identifier of the session and the version of the operating system. In *FcOQaQDGTl6gNhZ6G586_A_ua_preferences.db* database was stored information about urban airship platform. Registration token presents a great interest, which is reported in-depth as follows. Additional significant files are the named user id key, the last registration's payload in which was contained among others the time zone (Europe \ / Athens), the token and the identifier of the user, the channel's identifier through which the communication was taken place, the size of the first database, and useful dates and times, such as the time of last registration.

As in previous cases, in the `\shaders` folder each file's content starts with information about mobile device (j5lte), informing about its model (j5) and standard of wireless connection it can achieve (lte).

Next folder which was analyzed is `\shared_prefs` folder. As the name implies, the `com.google.android.gms.appid.xml` file informs users about the use of Google Cloud Messaging (GCM) client application. This file contains the registration token, which was found in a previous file. Essentially, an Android application needs to connect to servers through GCM connection before starts receiving messages. During registration, Android application receives a token and sends it to Android application's server. Relevant information is contained in `com.google.android.gms.measurement.prefs.xml` file, such as the `app_instance_id`, which is used for creation and renewal of registration tokens and the version of device's operating system. A great interest indicates the `gr.mobile.vodafone_preferences.xml` file. In this file is stored the following information:

- `<string name="preferences_msisdn_id">694*****7</string>`: The number of mobile phone with which the Android application has been matched
- `<string name="preferences_ivr_permission">true</string>`: Activation of interactive voice response
- `<string name="preferences_email_permission">true</string>`: User has given permission to the Android application to use their email address or send an email to them
- `<string name="preferences_temporary_password_url_id">https://www.vodafone.gr/portal/client/idm/cuLoginForm.action</string>`: The page to which the Android application is connected when the user signs in
- `<string name="preferences_named_user_id"> MSHSH_14_CXD_00000785861_68A70534BC7AC1FAE</string>`: User's identifier
- `<string name="preferences_sms_permission">true</string>`: User has given permission to the Android application in order to access device's messages or sending messages to the device

- `<string name="com.facebook.appevents.SessionInfo.sessionId">70301408-33b6-4faa-9ffe-e27c75bd763d</string>`: Facebook session identifier
- `<string name="refreshToken">ENmnmwgtTtXtOJKgi6SsX6c0.....mqAPWKWUA9PTelck</string>`: The new token after renewal
- `<string name="authenticationLevel">user</string>`: Either an application user, like in this case, or an application administrator is authenticated
- `<string name="clientSecret">F*****9</string>`: The *clientSecret* is a secret which should be known only to the application and the authorization server. This secret value used from OAuth 2.0, which is the industry-standard protocol for users' authorization to the server [31]
- `<boolean name="preferences_user_logged_in_id" value="true"/>`: User is logged in the Android application
- `<string name="preferences_online_top_up_bank_url_id"> https://www.vodafone.gr/apppages/pay/</string>`: The page with which the Android application is connected to when the user wishes to implement an electronic payment through bank
- `<boolean name="preferences_terms_launch_id" value="true" />`: User has read the terms of use when starts using the Android application
- `<string name="preferences_online_top_up_pay_pal_url_id"> https://www.vodafonecu.gr/apppages/pay/</string>`: The online page to which the Android application is connected to when the user wishes to implement an electronic payment from Android application through PayPal
- `<string name="preferences_basic_permission">true</string>`: User has given the basic permission to the Android application in order to access their device
- `<string name="accessToken">ijhSXH91YB-KvN27T....FrmlajhbH28xs</string>`: The Android application's identifier for accessing user's account
- `<string name="preferences_registration_url_id"> https://www.vodafone.gr/portal/client/idm/cuShowRegisterPrePay.action?request_locale=el</string>`: The

online page to which the Android application is connected to when a user is registered for the first time

- `<string name="Bundle_Checker_Active_Bundle_Label"> Ενεργό</string>`: Active packages' examination
- `<string name="preferences_forgot_password_url_id"> https://www.vodafone.gr/portal/client/idm/cuViewForgotPasswordRetail.action?request_locale=el
</string>`: The online page to which the Android application is connected to when a user selects to reset their forgotten password
- `<long name="com.facebook.appevents.SessionInfo.sessionEndTime" value="1522438558939" />`: Facebook session's end time
- `<string name="clientId">3654</string>`: Client's identifier
- `<string name="preferences_advanced_permission">true</string>`: User has given additional permission to the Android application in order to access their device
- `<string name="preferences_password_id">#@*****!</string>`: User's entered login password
- `<string name="preferences_username_id">71Joanna</string>`: User's entered username
- `<string name="preferences_forgot_username_url_id"> https://www.vodafone.gr/portal/client/idm/cuForgotUsername.action?request_locale=el
</string>`: The page to which the Android application is connected to when user selects to reset their forgotten username
- `<string name="preferences_moip_permission">true</string>`: User has given permission for moip technology use which is a solution for transactions and payments through mobile.
- `<long name="expiration" value="1591558454440" />`: The expiry date of the above permission

Finally, it was observed that there were labels associated with General Data Protection Regulation (GDPR), such as "*GDPR_Read_More_Label*" label. Therefore, it is concluded that the specific Android application is compliant with the above regulation.

6.4 OASA Telematics Mobile Android application

This is an Android application of Athens Urban Transport Organization (OASA). Users have the opportunity to be informed about scheduled bus routes, routes and locations of buses in real time, as well as buses' stops. Still, they can plan a route. Specifically, they can search for nearest bus stop from the place they are or choose any stop, preview real-time information for arrivals and buses' locations, choose their destination, departure or arrival time and a proposal for the best route via Public Transport (Figure 21).

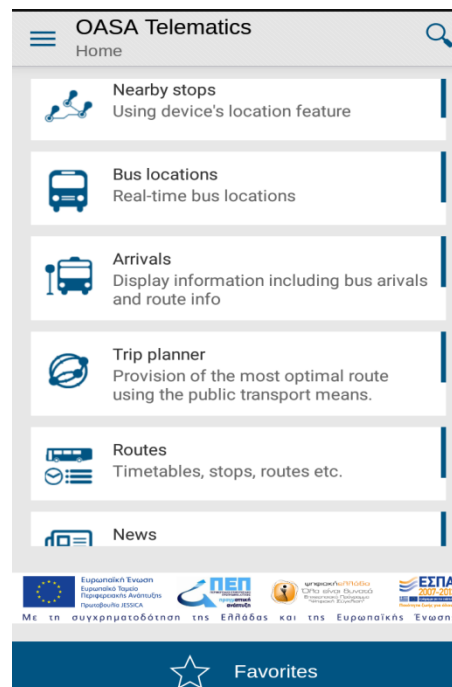


Figure 21: OASA Telematics app's home screen

Android application's data were stored in `\data\data\com.att.android.tfa` folder, which includes the `\cache`, `\code_cache`, `\databases`, `\files`, `\no_backup`, `\shaders` and `\shared_prefs` folders.

In `\databases` folder, `tfa.db` and `google_analytics_v4.db` crash_reports databases were located. In first database, information about bus routes and timetables was contained. Specifically, `Lines` table saves the existed bus lines, namely the line identifier and the code that user views, the

departure and arrival area and the route identifier each day . Main routes (from departure area to arrival area) were stored in *Masterlines* table -without the opposite route-, masterline's identifier and their code. In *MasterlinesDetails* table was matched main route's identifier with line's identifier. In *SchedCats* table was stored information about routes' separately based on season (winter and summer) and assigns different identifier (*sd_c_id*) for each one of them. In *SchedEntries* table was saved relative information about scheduled routes, such as *sd_c_id*, the departure time and estimated arrival time. In *SchedLineCats* table were matched the main line with the route category (winter or summer) as long as the month of which is valid. In *Routes* table were saved the route identifier and the bus line, the beginning and the end area, the distance and type of route (1 if the route starts from the start area and 2 if the route starts from the end area). In *RouteStops* table information about buses' stops, which are the route's identifier, the stop's identifier and the serial number of them were stored. In *SearchHistory* table was contained user's search history for buses. In *Stops* table was stored information about each stop, such as the name of the stop, the street where it is located, as long as the destinations to which the passenger can travel from every stop. In *UVersions* table, the version of each table and the last time the application was updated were saved. Finally, in *Favorites* table were contained the lines that user has saved as favorites. In *ITEM_ID* column are respectively stored the line identification (*Line_ID*) or the stop identification (*Stop_ID*). The fact that most of the above mentioned databases' fields could be modified and then saved with the new data is important.

As in My CU Android application, in *google_analytics_v4.db* database the *cid* and *tid* identifiers were stored, as well as *an*, *aiid*, *aid* and *av* parameters (*an* = *OASA% 20Telematics* & *aiid* = *com.android.vending* & *aid* = *com.att.android.tfa* & *av* = *2.0.35*) which were previously analyzed.

The last database pertain to bug reports and stores the size of bits which are recorded and transmitted each time as well as the moment when this transmission of errors began.

In *\cache* folder was contained *\volley* subfolder, which was created because the Volley library was used. Volley is an HTTP library that makes networking for Android applications easier and

most importantly, faster. This folder was contained files that store data regarding user's requests to the server. For example, each time user was selected to know in real time the position of a bus, a corresponding request to the server with all the necessary information was created. It was contained the IP address to which the Android application communicates and parameters related to the information the user was looking for. For example, Figure 22 illustrates a user request made for the location of bus (*act = getBusLocation*) with 2005 Route ID (p1).

```

HTTP/1.1 200 OK
Content-Length: 733
Content-Type: text/html; charset=UTF-8
Date: Tue, 08 May 2018 14:33:54 GMT
Keep-Alive: timeout=1, max=500
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/5.4.16
X-Android-Received-Millis: 1525790041174
X-Android-Response-Source: NETWORK 200
X-Android-Selected-Protocol: http/1.1
X-Android-Sent-Millis: 1525790041092
X-Powered-By: PHP/5.4.16
[{"VEH_NO": "20451", "CS_DATE": "May 8 2018 05:33:47:000PM", "CS_LAT": "37.9440910", "CS_LNG": "23.6658480", "ROUTE_CODE": "2005"}, {"VEH_NO": "20466", "CS_DATE": "May 8 2018 05:27:25:000PM", "CS_LAT": "37.9385170", "CS_LNG": "23.6309270", "ROUTE_CODE": "2005"}, {"VEH_NO": "20485", "CS_DATE": "May 8 2018 05:33:50:000PM", "CS_LAT": "37.9674880", "CS_LNG": "23.7306440", "ROUTE_CODE": "2005"}, {"VEH_NO": "20552", "CS_DATE": "May 8 2018 05:33:52:000PM", "CS_LAT": "37.9595000", "CS_LNG": "23.7204060", "ROUTE_CODE": "2005"}, {"VEH_NO": "20554", "CS_DATE": "May 8 2018 05:33:39:000PM", "CS_LAT": "37.9360960", "CS_LNG": "23.6384730", "ROUTE_CODE": "2005"}, {"VEH_NO": "20707", "CS_DATE": "May 8 2018 05:33:51:000PM", "CS_LAT": "37.9502490", "CS_LNG": "23.6924910", "ROUTE_CODE": "2005"}]

```

Figure 22: Cache preview

Furthermore, information for other user's activities was stored. For each activity a corresponding file is created, which structure shares similarities with the previous file. User has chosen to view bus' lines (*act = getLines*) and the bus' arrivals (*act = getStopArrivals*) at a specific stop (*p1 = 400052 and p1 = 400058*, where p1 parameter was the stop's code) and display the position (*act = getBusLocation*) of a specific bus (*p1 = 1803 and p1 = 2005*, where the p1 parameter was the path identifier). Moreover, user was searched position information for all buses of that path (*act = getRouteDetailPerRoute & p1 = 1803*).

As previously mentioned, `\code_cache` folder was contained information about the OpenGL ES API.

In `\files` folder was contained `\Fabric` system's folder and other files. In `gaClientId` file, the Client ID was contained unencrypted, which was found previously in `google_analytics_v4.db` database. Also, there were some files in relation to user's actions and have the `rList`

threepi.transport.app.ui.activity.ActivityName format where *ActivityName* was been replaced with the name of the corresponding activity. The *.ActivityHome* file was created when user was logged in the Android application and viewed its initial screen. The *.ActivityRouteDetail*, *.ActivityStop* and *.ActStopsMap* files are respectively related to user's actions about viewing details of a route or a stop or viewing the map with route's stops. The *.FavoritesActivity* file refers to user's favorite routes. All the necessary information is stored here in order for a user to add and preview the selected routes which previously has made. Finally *.SettingsActivity* file revealed that user has seen and has adjusted the Android application's settings.

As stated in an aforementioned Android application, in the same folder also exists *DATA_ServerControlledParametersManager.data.com.att.android.tfa* file, which stores information about servers and sites with the used satellites for finding a locating, such as www.gstatic.com. It was also been stored the website with the maps that the Android application uses. This website was www.google.gr/maps. In addition, the country (GR) and language (en) of the user have been saved. Finally, there are *firebase_inter_process_mutex-lock_send_report_to_server.lock* and *firebase_inter_process_mutex-lock_write_report_to_sqlite.lock* files, which indicate that Google's platform, Firebase, has been used.

As reported in detail in previous Android applications, *\.Fabric* folder saves reports about Android application's errors that will be sent for analysis, also unencrypted the session identifier (*Session ID*) and information about the Android application, mobile device and operating system. Moreover, it was been contained information about configurations that user has made in the Android application.

In *\shaders* folder there were files to which store mobile device's model (j5) and the wireless connection it can be achieved (lte).

Finally, in *\shared_prefs* folder, xml files existed, which contained plenty of information. In *com.att.android.tfa_preferences.xml* file were contained user's choice to activate the automatic update of the Android application, the date and time of the last Android application's update, its language and its version and database's version.

7. Future Work

Upon completion of our inspection, findings with a great interest have been discovered. As observed, a version's update could cause changes on the structure of `\data\data\application_name` folder and its files. For this reason, in the future it would be useful to be investigated and whether or not there will be any additions in a way that will affect users' data. Users tend to become aware of their privacy, thus they begin to care more for their data and how it process. It would be interesting to know if new Android applications' versions store additional to our findings personal information on their device and whether or not the data could be leaked easily.

On the other hand, even if an Android application's operation requires storing user's data on their device, It is important for additional analysis on new versions' of certain files (based on our findings) in order to ascertain whether or not previous inadequate implementations have been corrected. It should be checked if their data are stored in an encrypted format or into an encrypted database. If such thing happens, then should be defined the degree of difficulty for a third-party to decrypt them.

Finally, it would be helpful to re-examine the above mentioned Android applications and to proceed with the inspection to others. The analysis of popular Android applications would be useful to users in order to decide whether or not they will download an Android application. They will be able to know where their data are located and whether or not the existing security measures are capable to prevent a data breach.

8. Conclusions

Each Android application stores its data on `\data\data\ application_name` directory. In order to access this directory; the Android device should be rooted. Forensic investigation of Android applications could be accomplished either using commercial platforms for data extraction and analysis or using open-source tools. These platforms are Cellebrite UFED, Oxygen and XRY. However, in this thesis has been used Android Device Monitor (DDMS) through Android Studio In order to preview and copy files back and forth from the tested device. In some cases the access on rooted files has been succeeded using “Root Browser” Android application. These files can be copied from the internal to the external memory and then to a computer in order to be analyzed using DB Browser for SQLite, Realm Studio and other tools.

In each Android application, the `\data\data\ application_name` directory has a standard structure of ten folders. Making an analysis of these found that all of them save information about the Android application, user’s device, cookies, sessions and user’s activities. My COSMOTE Android application at its initial connection stores, among others, user’s mobile number. However, internet connection is prerequisite in order to run. The second Android application examined was What’s Up Android application. This application could operate without any internet connection to be necessary. In this case, it uses stored data informing users that these data are not updated. Those important data are located into an encrypted database using SQLCipher encryption algorithm. Furthermore, into another file is stored the username in encoded format. The third Android application which was investigated was My CU Android application. User’s personal data, credentials, history of their changes and the secret value for user’s authorization to the server are stored unencrypted into a database’s table. Without internet connection user could be logged in but the Android Application could not display any information about their account balance. The last analysis that we performed was about OASA Telematics Android application. It has been found that this Android application saves all the necessary information in user’s device. Moreover, it is stores user’s location,

search history and their favorite routes. The only case in which internet connection is necessary is for real-time bus' location search.

References

1. "Facebook Forensics" Valkyrie-X Security Research Group, 2011
2. Jamie McQuaid, ""Skype Forensics: Analyzing Call and Chat Data From Computers and Mobile", Magnet Forensics Inc 2014
3. Mahajan, Dahiya, Sanghvi, "Forensic Analysis of Instant Messenger applications on Android Devices", April 2013
4. Lone, Khalique, "Implementation of Forensic Analysis Procedures for WhatsApp and Viber Android applications", October 2015
5. Anglano, Canonico, Guazzone, "Forensic analysis of the ChatSecure instant messaging application on android smartphones", 2016
6. Wu, Zhang, Wang, Xiong, Du, "Forensic analysis of WeChat on Android smartphones", 2017
7. "Android Overview". Available at:
https://www.tutorialspoint.com/android/android_overview.htm
8. Chen, Yang, Liu, "Design and Implementation of Live SD Acquisition Tool in Android Smart Phone", 2011
9. Z.W. Chang, "The Study and Implementation of Operating System Porting for Android," July 2009
10. CircleCI, "What is the Android application Architecture?" Available at:
<https://www.quora.com/What-is-the-Android-application-Architecture>
11. "An Overview of the Android Architecture". Available at:
https://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture

12. Wojtek Kaliciński (2016), “#SmallerAPK, Part1: Anatomy of an APK” Available at: <https://medium.com/google-developers/smallerapk-part-1-anatomy-of-an-apk-da83c25e7003>
13. Akshay Chordiya (2017): “Introduction to Android Architecture Components”. Available at: <https://android.jelise.eu/introduction-to-android-architecture-components-22b8c84f0b9d>
14. Google Developers (2018): “Guide to App Architecture”. Available at: <https://developer.android.com/jetpack/docs/guide>
15. Hashim Shaikh (2017): “Practical Android Phone Forensics”. Available at: <https://resources.infosecinstitute.com/practical-android-phone-forensics/#gref>
16. Google Developers (2018): “Data and file storage overview”. Available at: <https://developer.android.com/guide/topics/data/data-storage>
17. Wikipedia (2018): “Rooting (Android)”. Available at: [https://en.wikipedia.org/wiki/Rooting_\(Android\)](https://en.wikipedia.org/wiki/Rooting_(Android))
18. recovery.tar.md5 file. Available at <http://androidhost.org/d0pWJ>
19. SuperSU-v2.82.zip file. Available at <http://www.supersu.com/download>
20. Odin3 v3.12.10 tool. Available at <http://www.droidviews.com/download-odin-tool-for-samsung-galaxy-devices-all-versions/>
21. Android Studio. Available at <https://developer.android.com/studio/>
22. DB Browser for SQLite tool. Available at <https://sqlitebrowser.org/>
23. Realm studio program. Available at <https://realm.io/products/realm-studio/>
24. Zetetic Company (2018), SQLCipher. Available at <https://www.zetetic.net/>

25. Godfrey Nolan (2015): Hiding the key | Practical Advice for Building Secure Android Databases in SQLite. Available at:
<http://www.informit.com/articles/article.aspx?p=2268753&seqNum=4>
26. CERT-UK (2014), “Code Obfuscation”
27. ProGuard obfuscator tool. Available at
<https://www.guardsquare.com/en/products/proguard>
28. Apk decompiler. Available at <http://www.javadecompilers.com/apk>
29. Eclipse Java Oxygen. Available at <https://www.eclipse.org/>
30. Google Developers (2017): “Measurement Protocol Parameter Reference”. Available at
<https://developers.google.com/analytics/devguides/collection/protocol/v1/parameters>
31. Okta, “The Client ID and Secret”. Available at <https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/>