



VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING AND
CYBER SECURITY LABORATORY

Bachelor Thesis

Investigation of Digital Forensic Methods for Mobile Devices

Done by:

Dovydas Patapas

signature

Supervisor:

assoc. prof. dr. Linas Bukauskas

Vilnius
2021

Contents

Abstract	3
Santrauka	4
Introduction	5
1 Related Works	6
2 Methodology	7
2.1 Mobile Device Forensics	7
2.1.1 Forensic analysis process	7
2.1.2 Mobile data acquisition types	8
2.2 Android Mobile Device Features	10
2.2.1 Android architecture	10
2.2.2 Android apps and data storage	11
2.2.3 Android security features and Root access	12
2.3 Android application development	13
3 Forensic analysis approaches	14
3.1 Specialized forensic software tools	14
3.2 Analysis using Android Debug Bridge	16
3.3 Analysis using a native application	19
4 Prototype forensic application	20
4.1 Functionality	20
4.2 Implementation	20
4.2.1 Finding package folders	21
4.2.2 Finding databases	22
4.2.3 Accessing data	22
Conclusions and Recommendations	24
Further Research	25
References	26

Abstract

Mobile devices are widely used in various areas of our lives. In some cases, mobile devices can be involved in criminal activities. Mobile device forensics is a scientific field about extracting and analyzing mobile device data to be used as evidence in a criminal case. Forensic investigators use various techniques and tools to collect evidence from mobile devices. Because of the constant advance in technology and a variety of mobile device designs, mobile device data extraction can become difficult. This work investigates several approaches to forensically analyzing Android mobile devices. Some background theory related to mobile device technology is also provided. One of the data analysis approaches is demonstrated by developing a prototype data analysis application for Android devices. Functionality and implementation of this applications are described. Suggestions are given on how this approach could be used and where it could be improved.

Santrauka

Skaitmeninių įrodymų mobiliuosiuose įrenginiuose rinkimo bei patikrinimo metodų tyrimai

Mobilieji įrenginiai plačiai naudojami įvairiose mūsų gyvenimo srityse. Kai kuriais atvejais mobilieji įrenginiai gali būti susiję su nusikaltimais. Mobiliųjų įrenginių kriminalistika yra mokslo šaka, susijusi su duomenų, esančių mobiliuosiuose įrenginiuose, išgavimu ir analize, siekiant juos panaudoti kaip įrodymus nusikalstamoje byloje. Ikiteisminiai tyrėjai naudoja įvairias technikas ir įrankius rinkti įrodymus iš mobiliųjų įrenginių. Dėl nuolat tobulėjančių technologijų ir mobiliųjų įrenginių dizainų gausos mobiliųjų įrenginių duomenų išgavimas gali tapti sudėtingu procesu. Šis darbas aprašo kelis Android telefonų duomenų analizavimo būdus. Pateikiama teorinės medžiagos susijusios su mobiliųjų įrenginių technologija. Vienas iš duomenų analizės būdų demonstruojamas kuriant prototipinę aplikaciją Android įrenginiams. Aprašomas šios aplikacijos funkcionalumas ir įgyvendinimas. Pateikiamos rekomendacijos kur šis analizės būdas gali būti naudojamas ir kaip jis galėtų būti patobulintas.

Introduction

Technology has become a major part of our lives. Many of our activities such as communication, leisure and research would be hard to imagine without electronic devices. Some of them are mobile devices like smartphones that most people carry with them every day. Mobile devices have made a huge technological progress in a relatively short amount of time. Smartphones today can perform the majority of tasks of a modern personal computer. People use them for many purposes ranging from voice and text communication to various mobile applications and data storage. Since mobile devices have become so important and widely used, it's natural that they can be involved in criminal activities. Mobile devices store various types of digital data that could contain important information related to these criminal activities. To analyze it, the mobile device must be forensically investigated. Digital Forensic investigators are responsible for taking in mobile devices, acquiring the data stored in them and analyzing it to find evidence [4]. Over the years these methods have been researched and became a scientific field - mobile device forensics.

There are many challenges that arise when analyzing mobile devices. A smartphone that is being investigated may run one of the widely used mobile operating systems - Android, iOS, Windows. Each of these operating systems have many different versions. Moreover, there is a huge variety of smartphone models from different manufacturers that differ in hardware characteristics and other features. Because of these differences forensic investigators may need to choose between different mobile device investigation approaches [14]. Manufacturers often release new operating system versions, device models and software updates. Sometimes these changes may impact mobile device file structures and the way data is stored [12]. Forensic investigators must keep up with this constant change and choose from many different software tools that can be used to analyze mobile devices. Another big challenge is dealing with smartphone security features. Phone screen locks, passwords, disk encryption and other mechanisms which are created to protect user data make it difficult to access information stored in mobile devices [12]. As new devices are released, these security features are constantly updated, requiring more sophisticated forensic tools and techniques. Forensic investigators must know how to choose the right tools and methods for each investigation.

This work focuses on ways to analyze Android smartphones. Android is currently the most widely used mobile device operating system, with a 71% market share in 2020 worldwide [6]. It uses software applications called apps that provide various functionalities. Apps can be either installed by the user or provided by default with the system. Data related to these apps is usually stored in database files and contain information that can be used in an investigation. The main goal of this work is to investigate methods of forensic data extraction from Android smartphones. A simple Android application is developed to demonstrate how a native smartphone application can be used in a forensic investigation.

The first section describes articles, books and other literature sources that are related to this work. Section 2 gives an overview of the methodology used. Subsection 2.1 is a short introduction to the mobile device forensics field, subsection 2.2 describes Android technical features relevant to the work and subsection 2.3 shortly describes Android application development. Section 3 provides investigation of a few widely used forensic analysis approaches and describes several software tools. In Section 4, the goal functionality and implementation of a prototype forensic application is described. The conclusions and recommendations section gives final remarks about the results of this work and provides insights about what could be done differently. Finally, some possibilities for further work and research topics are described in the last section.

1 Related Works

Although mobile device forensics is a relatively new field of study, there is a variety of books and scientific articles published on this subject.

The 2018 edition of *Practical Mobile Forensics - Third Edition: A hands-on guide to mastering mobile forensics for the iOS, Android, and the Windows Phone* by Rohit Tamma, Oleg Skulkin and Heather Mahalik [14] is a great starting point when researching this field. It describes the overall mobile device forensic process and provides several data extraction approaches. Some technical details of the most popular mobile operating systems, Android, iOS and Windows Phone are provided. The book describes tools and techniques used for analyzing each of these devices and explains common technical challenges that forensic investigators encounter.

Another book, *Learning Android Forensics* by the co-authors of previously mentioned work, Oleg Skulkin, Rohit Tamma and Donnie Tindall [12] is an in-depth guide focused on the most popular mobile device platform - Android. In the latest 2018 edition, this work describes the tools and technical knowledge needed to analyze Android devices. This book provides an overview of the Android file hierarchy, storage types and examples of forensic analysis for various popular applications such as Messenger, Facebook, Google Chrome, Whatsapp and many more.

There are also many in-depth scientific articles on various subjects related to mobile device forensics. *Developing Process for Mobile Device Forensics* [4], an article published in 2011 by detective Cynthia A. Murphy, describes the digital forensic process from the law enforcement point of view. Although this article is written some time ago, the forensic analysis steps of identification, preparation, isolation and processing are applied for investigating mobile devices to this day.

The Forensic Analysis of Smartphones: The Android Data Extractor Lite, ADEL [5] is a project started by German scientists in 2011 to develop an application which extracts selected SQLite database files from an Android device and analyzes them in an automated way. A system for dumping Android database files with Android Debug Bridge, saving them on a forensic workstation and creating reports is provided. This project was being developed for 2 years and has an online blog describing the results. Although this work was done using older versions of Android, the steps taken to develop such an application can be useful when working with newer device versions today.

A more recent article, *Hack an Android App: Finding Forensic Artifacts* by Kolin Stürt [16], is a practical guide for extracting Android application data using Android Debug Bridge. It uses an example application and describes several ways to access its data. It also provides useful practical information on various Android security features, analyzes several command line tools and gives suggestions on how to proceed with further research on this topic.

A Technical Look at Phone Extraction is an online publication by the Privacy International organization written in 2019 [9]. It describes various approaches that can be used to analyze the newest mobile devices. Various popular commercial forensic tools are analyzed and evaluated in relation to mobile device security features.

2 Methodology

2.1 Mobile Device Forensics

2.1.1 Forensic analysis process

Growing popularity and use of smartphones led to an increased demand for forensic investigation. Mobile device forensics is the process of extracting and analyzing data stored in mobile devices to find evidence that could be used in a criminal investigation [14]. There are some common challenges that many forensic investigators face when trying to extract useful information from a smartphone:

- **Smartphone hardware differences:** There are many different smartphones from different manufacturers. Different models may need different methods to extract data. In some cases a forensic investigation software tool may work only on certain device models. New smartphone models are released quite frequently and newer models are generally harder to examine because fewer software tools can analyze them. Mobile device forensic investigators and forensic software developers need to keep track of new device releases and remain updated.
- **Smartphone operating system differences:** Forensic investigators need to be aware of the most widely used mobile device operating systems. Although there are not as many different operating systems in the market as there used to be several years ago - most devices run either Android or iOS - these operating systems receive regular updates that can change various features. Because of this, forensic tools and methods used to examine a smartphone may also need to be changed.
- **Smartphone security features:** Smartphones are sold with many built-in security features that are created to protect user data and privacy. User data and system files in a smartphone may be protected using encryption. For example Android uses a file-based encryption mechanism that encrypts different files with different encryption keys [1]. Investigators and forensic software developers may need to break through such security features to extract data. As new security features and patches are released, new security loopholes need to be discovered.
- **Preventing data modification:** It's important that mobile device data is not altered during a forensic investigation for it to be used as evidence [14]. Techniques such as data hashing can be used to verify if there were any data changes during extraction. Common file types such as media files and documents are not likely to be altered, but there are many background processes that constantly run on mobile devices and result in minor changes to the system. Forensic investigators need to choose data extraction and analysis methods that minimize the risk of data alteration.
- **Lack of resources:** The growing number of different mobile device models and systems requires forensic investigators to have tools suitable for many different situations. Growing amount of data in mobile devices often makes it necessary to have workstations with relatively high performance that can support fast data extraction and analysis.

To avoid as many difficulties as possible and simplify forensic investigation, several standard processes have been developed. There are certain steps that have to be taken during most investigations (see Figure 1).

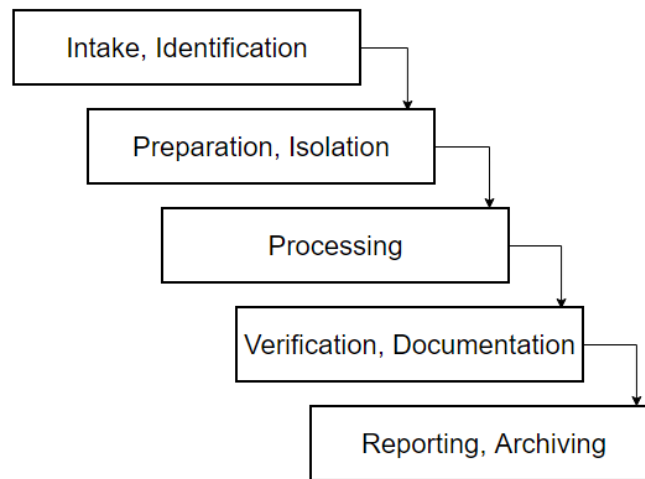


Figure 1. Generalized digital forensics process [14, 4]

The investigation begins with the intake of a mobile device. The criminal activity that this device is related to lets investigators formulate the goal of the investigation. Choosing a specific goal for a forensic investigation is important because it often helps to understand what types should be prioritized. This in turn may narrow down the list of possible data analysis approaches. It is also very important to correctly identify mobile device features such as model, operating system version, identification information and whether the device contains any removable data storage such as SD cards. This information lets forensic investigators begin the preparation phase, during which they research the mobile device and choose the tools and methods that will be used for data acquisition and analysis [4]. During some investigations mobile phones are isolated from cellular, Bluetooth or wireless networks, so the data on it could not be changed or erased remotely.

After the initial phases and all preparations are complete, processing can begin. This is when the mobile device data is extracted and prepared for analysis. Forensic investigators try to use data extraction methods that are already tested and can be repeated [14]. These methods should also be as forensically sound as possible - meaning that data changes on the device should be minimized. The previously clarified goal of the investigation helps to decide how much data should be analyzed - whether only specific information is required or a full device data copy should be made. After the data is extracted, it can be verified to ensure that it wasn't altered, often using hashing techniques.

It is a good practice to document what steps are taken during the investigation. Documentation may include information about the device such as device condition and state, time of the investigation, device state, tools used for analysis and various other notes. This way forensic investigators can be sure that nothing is overlooked and the investigation was done according to all requirements and regulations that may be stated by law. During final phases investigation results are prepared for reporting. Data that is most relevant to the investigation is collected and presented in a format that is suitable for use in court of law.

2.1.2 Mobile data acquisition types

Forensic investigators must gather mobile device data to look for evidence. Acquisition is the process of extracting information from a mobile device. There are various acquisition types a forensic

investigator may choose, often depending on the investigation type, goals and resources. The most basic approach is manual extraction. This is done by simply searching for relevant data through the device's touchscreen or keypad [14]. This method can be used in cases where the required information is easy to find and there is no need for large amounts of evidence. An investigator can look through instant messaging applications, user files, photo's and similar evidence. The information on the mobile device can be photographed and documented manually or with the use of simple tools. There is a high probability for errors during manual extraction, also some data may be change during the process, for example text messages marked as read. Manual extraction can be used to access only the data that is accessible through the user interface, excluding deleted information, system and other hidden files.

In many cases investigators require large amounts of data for further analysis. There are ways to acquire an image of the whole mobile device data. This is usually done by connecting the device to a dedicated computer used for investigation, often called the forensic workstation. Data is then copied from the mobile device to the computer using software tools. Some data stored in mobile devices may be easily accessible, while other types of data may require more effort. Smartphone information can be extracted by making images, which are usually categorized into two main types - logical and physical [15].

Logical images are a copy of all files and folders from mobile device storage. This is done by extracting all the contents of a mobile device filesystem. The copied files can be easily accessed on a workstation computer because their format remains intact - files have their filesystem headers. Because of this, logical images are relatively easy to work with because they have a defined structure. This is not a complete copy of the filesystem though, only it's logical contents. This means that a logical image does not include most system files, hidden logs and deleted files.

A Physical image is used to extract all the data stored in the physical mobile device memory. This allows to access deleted data and hidden files that could be important for a forensic investigation. Physical acquisition is generally harder to perform than logical acquisition, and requires knowledge about the mobile device and more sophisticated software tools to access the information. However, it is preferred by forensic investigators that want to extract as much evidence as possible.

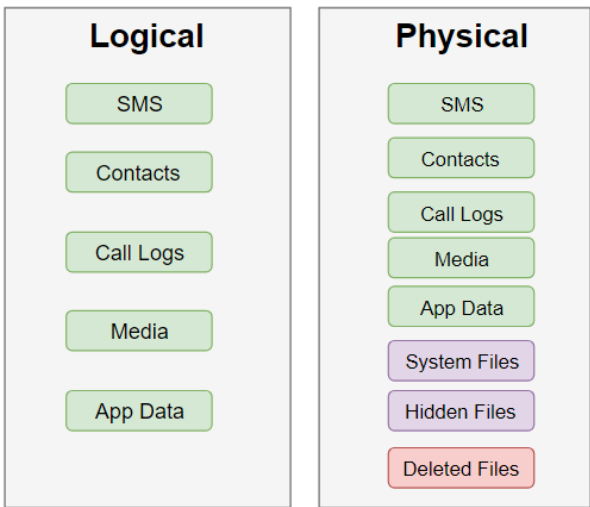


Figure 2. Types of data that can usually be extracted using logical and physical file acquisition

2.2 Android Mobile Device Features

2.2.1 Android architecture

Having a good understanding of mobile device operating system features can help forensic investigators the correct methods for data extraction and analysis. The Android operating system can be logically divided into several layers of software that exist on top of one another [3].

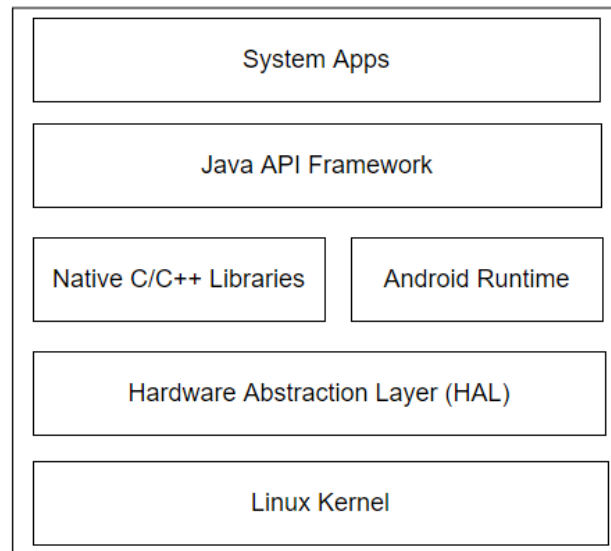


Figure 3. Android operating system architectural structure

The lowest level of the Android operating system is the Linux kernel. The kernel is the core component of an operating system that's responsible for interaction between software and hardware. It acts as a level of abstraction between hardware and software through drivers - programs that control the various devices of the smartphone. The kernel is also responsible for core functions of the operating system such as process management, memory management, networking and security [12].

The Hardware Abstraction Layer (HAL) is another abstraction level that allows the Java API framework to control the mobile device's hardware. This is done through standard interfaces that allow Java code to access hardware components such as camera, various sensors or Bluetooth. This layer simplifies smartphone software development making it relatively independent from most the lower level specifics.

Android Runtime is a set of virtual machines, in which every Android application runs its processes. This is done by executing a special bytecode format that compiled from Java source code.

Native C/C++ Libraries are used to implement core system components, including Android Runtime and the Hardware Abstraction Layer.

The Java API Framework contains the main building blocks from which Android applications are developed. These building blocks are components and services such as the View System that allows to build application's user interface using elements such as lists, grids, text boxes, buttons, Resource Manager that provides access to files and other resources of an application, such as strings, graphics and layout, Notification Manager that allows applications to display alerts on the device screen, Activity Manager that manages the flow of the as application user interface and Content Providers that allow applications to access data from other applications and share their own data to them.

The Application Layer is the top layer in the Android architecture stack. It holds Android applications (called apps), programs that users can interact with.

2.2.2 Android apps and data storage

The main goal of digital forensic analysis is to extract all the necessary data from the device. To achieve this, forensic investigator must know what kind of data is stored on the devices, how and where it is stored. Knowing these details helps the investigator to decide which methods will be used to extract the data.

The Android main data storage is divided into logical units called partitions. Partitions allow to logically divide disk space into sections that can be accessed independently [12]. Android partitions may be different between mobile device manufacturers and operating system versions, but there are a few main partitions that are always used.

- The */boot* partition contains information and files required for the mobile device to start its processes when it boots after it is turned on
- The */system* partition contains all the main components required for the operating system to work and pre-installed applications
- The */recovery* partition allows the phone to boot into a menu that can be used to install updates and perform other maintenance tasks. This partition is often used to install custom third party system software or gain privileged access to the device by exploiting a security vulnerability
- The */data*, usually called userdata partition is where the devices stores the application and user data. Most of mobile device forensic analysis is performed on this partition.
- The */cache* partition is used to store data that is frequently accessed.
- The */misc* partition contains configuration for various system settings, such as USB configuration some hardware settings.
- The */sdcard* is not an actual partition on the device internal memory but the place where contents of an external SD card storage are found.

The file organization in the Android file system is similar to Linux. All files and directories are in a hierarchy that is a single tree which has a starting point called root, marked as */* in the terminal interface. Data belonging to different applications can be stored either in */sdcard* directory or in */data*. If it is stored in */data*, the location is predefined, with each application having a separate folder called after its package name. Each application directory may contain several subdirectories that store different types of data (figure 4).

Android applications can be divided into two main categories: system apps and user installed apps [12]. System applications are installed by default when a new device is shipped by the manufacturer. These are apps that are responsible for basic smartphone functions such as phone calls, SMS messages or default web browser. Most system apps are stored in the */system* partition. They are meant to be read-only - they cannot be changed or uninstalled by the user. However, in some cases they can be disabled - they are then hidden from the user, but their data remains. System apps are generally stored in the */system* partition. User applications are apps that can be installed or removed by the user. They are stored in the */data* partition and can usually be edited.

Sub directory	Description
shared prefs	stores the shared preferences XML file, which is a configuration file that can contain various configuration parameters, version numbers
lib	custom library files required by the application to run
files	files saved by the developer
cache	files that are used frequently, cached by the application
databases	the application SQLite database and related journal files

Figure 4. Android application directory structure

2.2.3 Android security features and Root access

By default, Android has many application security features. These features can cause some problems for forensic investigators that try to extract and analyze mobile device data. Because of this, it is worth what some these features do and how can they be bypassed.

Android is a modified version of the Linux operating system. It uses many of the security features Linux provides by default. Linux is a multi-user operating system with each user having its unique identification number and file access permissions. In Android, each application is like a Linux user, having certain read, write and execute permissions to various files. Each app only has the access to files that are required for it to work [1]. An application can request permission to access various device data such as location information, Bluetooth services and camera, but under normal conditions it can not access the internal storage data of another app. Two apps can share the same data, but only if they are created using the same certificate used for security authentication. Because of this, forensic investigators need to find ways to get permissions for full application data access.

There are ways to gain full file system permissions on Android, also called the Root access. This allows a user to browse all the files stored on the mobile device and install apps that have special permissions. Root access was not intended by Android operating system developers, so it is usually gained by exploiting some kind of a security flaw. The Android operating system is open source, however the hardware and ROM - read only memory is not. In earlier Android versions, up to around 5.0, Root access could be gained by simply downloading and installing an application. This meant that mobile device data could be easily extracted from a phone. In newer Android versions, special security features such as disk and file data encryption were introduced, and greatly increased the difficulty of gaining Root access on the device. These features usually mean that gaining Root permissions deletes all data on the device /data partition [11].

Some commercial forensic data extraction tools provide ways of gaining root access without deleting any user data. These methods usually involve installing a custom recovery partition or custom ROM - read only memory code to bypass the security and gain a temporary Root access for data extraction [9]. However, the exact methods used by these big companies are usually not revealed to the public.

This work is done under the circumstances that Root access is already gained on the test device. For this Samsung device, it was done by installing a custom recovery partition, which allows to install a special application for gaining Root. This custom recovery provides a simple menu that allows users to install various third-party software that otherwise would not be allowed by the

phone's security. The recovery partition is installed by entering a special download mode on the device. This mode is used by system developers and mobile device repairmen to install custom system software and firmware. On the Samsung device used, the download mode is entered by rebooting and holding power, home and volume down buttons all at once.

2.3 Android application development

Android applications are usually small programs that can be run on many different Android versions and devices. These apps can be written using either Java, Kotlin or C++ programming languages. This work will be using Java, because it has many learning resources on the web and especially Youtube. Android applications are compiled into a special format, the APK - Android Package. This file format is an archive that contains all the data of the app, including various file contents [1]. It's used by Android devices to install an app, and can be easily moved between them.

Android applications are usually developed using the Android Studio IDE - Integrated Development Environment. Although other IDE's can be used, Android Studio provides many built-in tools for application GUI development, and many pre-installed packages for various common application features. Android Studio also provides AVD Manager - a virtual device manager plugin that can simulate a real Android device - a phone, tablet and many others - in the computer operating system environment. This creates a virtual device that resembles all the characteristics of a real one. It can be used for testing an app on many different device models, with different screen resolution and performance, to check portability. There are some device models that are available by default, mostly including devices from Google, but the user can import his own virtual device configurations. The practical part of this work is tested on one real and two virtual devices. The real device is a 2017 Android Galaxy A5 with a relatively new Android 8.0 operating system.

Before Android applications are compiled for distribution, they have a certain file structure (figure 5) . Understanding this file structure reveals a lot about how the applications work and helps creating applications that can be used for forensic investigation.

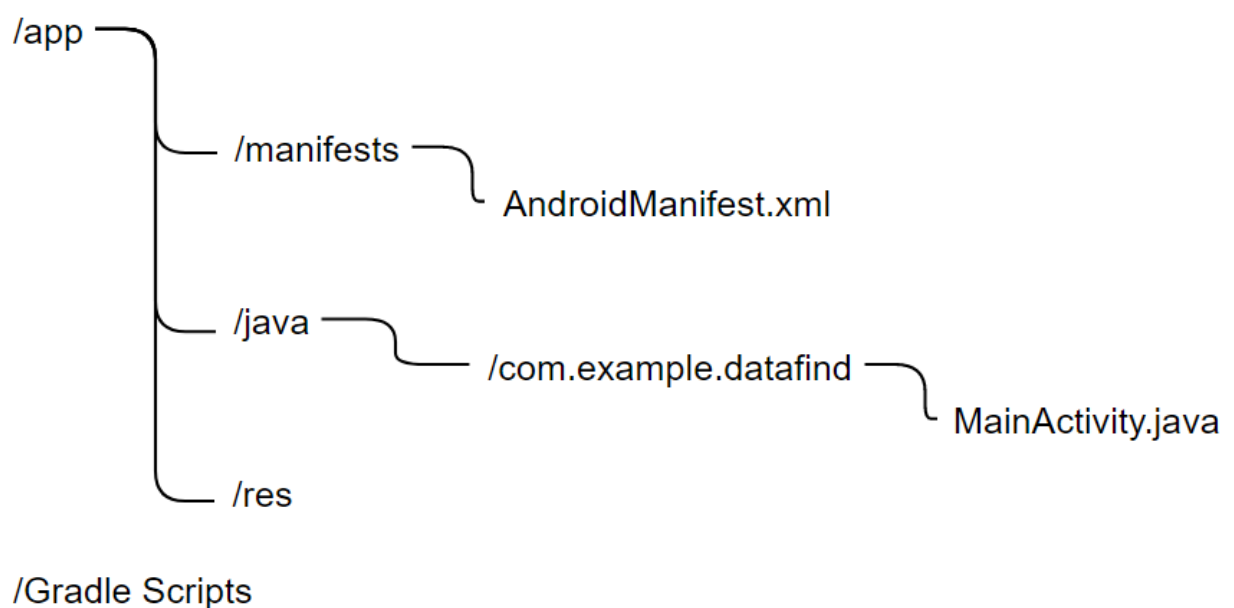


Figure 5. Android application file structure in the Android Studio IDE

The Java files that describe the main application logic are contained in the `/java/com.example<app name>` folder. There are 4 main types of java classes that are used in Android development - Activity, Service, Broadcast Receiver and Content Provider. The Activity class is the main one that is instantiated when a program starts. Each activity represents the logic behind a user interface window that is displayed to the user and is the main entry point to the application. An activity can start another activity, for example when a user open a new window.

The user interface configuration is stored in the `/res` folder. It's written in XML format, but Android Studio provides a drag-and-drop GUI creation tool that automatically adds configuration to the file. Each user interface file, also referred to as layout file, usually corresponds to a single Activity. The `/res` folder also contains various resources used in the application, such as pictures, string texts, color values. It is a good practice to store these resources in this directory, so they wouldn't have to be hard-coded into the application logic.

The Android manifest file is an important configuration file that lists each application component and includes information about minimal operating system and hardware requirements for the app (figure 6). It is also used to identify any user permissions that are required by the application. These may include access to external data stored in the `/sdcard` directory of the device, camera, location access and many more. One of the settings very relevant to this investigation is `android:allowBackup` parameter, that defines if a full application data backup can be done using ADB. This setting is set to `false` in almost all popular applications.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.datafind">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <application
        android:allowBackup="true"
```

Figure 6. Some application user permissions from the Android Manifest file

The last section of project files are the Gradle Scripts. Gradle is an additional tool supported by Android Studio. It has scripts that are responsible for building the application - taking all the files in the project and making an APK file from them [7]. It provides the ability to organize the file structure before a project is finished. Gradle has many automated features, such as testing and dependency checks, that improve the performance of the IDE and overall development process.

3 Forensic analysis approaches

3.1 Specialized forensic software tools

While mobile device forensics is a relatively new field, there are many specialized tools that can be used for investigation. Both commercial and open-source tools exist in the market. Commercial forensic investigation tools are sold to private companies or investigators. These applications are used as complete solutions for the whole mobile device forensic investigation process. Users receive tools that have simple GUI and some companies have training courses to assist them. Commercial tools receive regular updates so they can be used to investigate the newest mobile devices. With competition between forensic software companies, technologies involved in these applications are proprietary and usually kept secret [9]. Here are some of the most popular commercial forensic applications that can be used to investigate mobile devices:

- Cellebrite is one of the leading forensic software companies. Their tool Cellebrite UFED Physical Analyzer can be used for mobile device extraction, data examination, analysis and reporting [13]. Physical Analyzer can be used for both logical and physical acquisition, and bypass smartphone lock screens. Cellebrite claims that their tools can also overcome most modern smartphone data encryption and security features. This company offers many different solutions for various customer needs.
- Magnet forensics is another company that provides forensic tools for various types of investigations. Magnet Axiom, their complete forensic solution, is advertised for its ability to bypass some of the newest iOS and Android security features. They also offer a separate package for data extraction, Magnet Acquire, that can be ordered by private companies with a 30 day free trial. This tool can be used for physical acquisition from devices with locked screens and no root permissions.

There is also a wide variety of open-source tools available for forensic investigations. This can be anything from small applications that perform a single task, to big bundles of applications that allow users to choose which tools they want to use. Some of these tools have evolved from digital forensics tools that have been used to investigate all digital storage devices in general. Unlike commercial tools, free forensic software tools allow anyone to modify and change how their tool works. Here are some of the widely used open source forensic solutions:

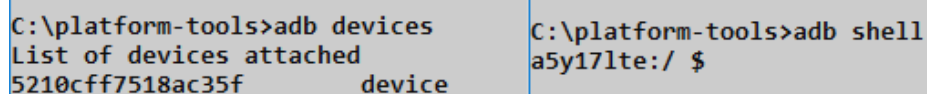
- SANS Investigative Forensics Toolkit (SIFT) is a forensic operating system that has a huge variety of tools for many types of digital forensic investigations [13]. SIFT can be used directly on a workstation, from a virtual machine or as a live operating system from a bootable storage device, allowing forensic investigators to use it in various difficult situations. SIFT is a multi-purpose system that can be used all kinds of digital forensic tasks, including mobile device forensics. It contains hundreds of tools and packages for examining different disk image and file system types, data analysis, reporting and other forensic activities. However, SIFT requires installing a separate operating system and performing many configurations, which may be difficult for new users.
- Autopsy is a popular forensic toolkit that allows investigators to examine extracted data and create reports. Users can create forensic cases with many evidence data sources and save them for later use. Autopsy is a GUI application that is relatively easy to use. It performs data examination and reporting using various modules. Autopsy can be installed on Windows, Linux or iOS operating systems, and it can also be used as part of the SIFT toolkit mentioned earlier. In addition to default modules, users can add third party modules or create their own to improve Autopsy. Autopsy is regularly updated and has a big developer community. However, it can't be used as a standalone forensic tool for mobile devices, because it doesn't support mobile device data extraction, data must be imported into Autopsy after it is extracted using a different tool.

While commercial forensic software solutions offer many advanced features such password unlocking and encryption bypassing, their services are usually expensive. Forensic investigators have to evaluate the costs, and for many small companies this might not be worth the price. Although there are communities that participate in the development of open source forensic tools, it's very hard to keep up with mobile device security updates and maintain support for every smartphone model.

3.2 Analysis using Android Debug Bridge

One of the tools that is widely used in mobile device forensic analysis and data extraction is Android Debug Bridge. Android Debug Bridge, or ADB in short is a tool used by Android developers to test and install applications on Android devices. It allows the user to establish a command line interface from the workstation to the device and use the Linux terminal commands on the device. The workstation can then send commands to the device. ADB can be used to search and extract mobile device data in several ways, coupled with some additional command line tools if necessary. Android Debug Bridge is a free tool and can be downloaded from the Android developers website along with other development tools. The following section provides an example of how this tool can be used to extract mobile device data.

The Android device is usually connected to a forensic workstation using a USB or similar cable. It is important that both the USB file transfer and USB debug mode is enabled on the mobile device through the settings. The workstation must also have the correct device drivers, these may be different for each manufacturer. Once the connection is authorized, the adb devices command is used to see what devices are connected to the workstation. Then the adb shell command can be issued to establish a command line interface and allow the investigator to use various tools on the device (figure 7).



```
C:\platform-tools>adb devices
List of devices attached
5210cff7518ac35f      device

C:\platform-tools>adb shell
a5y17lte:/ $
```

Figure 7. Connecting to an Android Device using ADB

Because the Android operating system uses a Linux kernel, some of the Linux terminal commands can be used on the mobile device. The ADB shell allows users to navigate through the mobile device file system and search for data. There are several ways that ADB can be used to extract data from a mobile device. The simplest method is using the ADB backup command. This command can be used to make a backup of certain application data or make a full backup of all applications (figure 8). It creates a backup.ab file that can be secured with a password and later recovered on a different device. However, without privileged permissions this method only extracts applications that store their data in the shared application data directory and allow their data to be backed up.



```
C:\platform-tools>adb backup -apk -shared -all

C:\platform-tools>adb backup -apk -shared com.example.android.SqliteApp
```

Figure 8. Using the ADB backup command to extract shared application data of a single app or all available apps

In many cases forensic investigators may want to extract a full mobile device data copy - perform a physical data acquisition. The ADB can be used together with a few other command line tools to perform a physical extraction. This extraction method requires root access to the whole mobile device file system. To perform a physical extraction using ADB, an application called BusyBox is installed on the mobile device for gaining access to additional Linux commands.

This application can be installed directly from the workstation using ADB. When the device is connected to the workstation, the su command is issued, switching to privileged root access. The Linux dd command is used to make the physical device data extraction (figure 9). This is done by copying the Mmcblk0 partition of the Android storage system. Then, Netcat application is used on both devices to redirect the output of the dd command from the mobile device to the workstation. This creates a mobile device disk image file. The image file is usually large and generally takes some time to extract. After the extraction, forensic investigators analyze the physical image to find useful evidence. A copy of the image or a hash function value may also be saved to later confirm data authenticity if the investigation requires that.

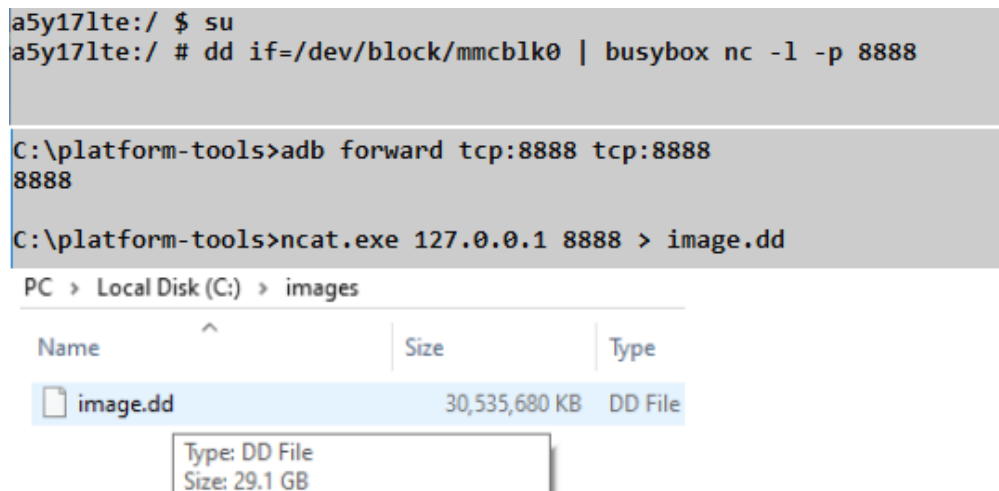


Figure 9. Ar tikrai čia eina localhost IP adresas ADB, Netcat and dd commands used on both the mobile device and the workstation to perform physical data acquisition

It is also possible to extract certain application databases using ADB with root permissions. When the ADB shell is opened, the package manager command pm list packages can be used to list the applications installed on the mobile device. This gives a long list that contains both the user installed and system apps. If required, the output of this command can be modified using the grep command line tool to see only the user installed applications (figure 10). The naming convention for Android application folders is similar to *com.<manufacturer name>.<application name>*. If an application has a database, this directory contains a */databases* subdirectory, which stores the databases and related files. To extract these databases, they can be copied to the */sdcard* directory, which is a public storage area of the mobile device that can be accessed by external devices. From there, the database files can be extracted using ADB or simply manually copied using the file explorer on the workstation [16]. The ADB pull command is useful because it can be included in various scripts. If a list of paths to needed files is known, it could be relatively easy to write a script that pulls these files from the device [12].

```

a5y17lte:/ # pm list packages -f | grep /data
package:/data/app/com.google.android.youtube-1tZTfo609XWrkKan8XNFsg==/base.apk=com.google.android.youtube
package:/data/app/stericson.busybox-AnL4RqrIf10B0TJY2164Eg==/base.apk=stericson.busybox
package:/data/app/org.telegram.messenger-1TmXnD8faUnb1Rb9fZ5sbQ==/base.apk=org.telegram.messenger
package:/data/app/com.whatsapp-Q2GQLeAeQjNZ-AvdwuG7QA==/base.apk=com.whatsapp
package:/data/app/com.example.sqliteapp-UZq7eLZUD-xaZ4gupsgUqg==/base.apk=com.example.sqliteapp
package:/data/app/com.sec.android.app.samsungapps-nT7N7iiRFTd7wf7ToFRwRg==/base.apk=com.sec.android.app.s
a5y17lte:/ # cp /data/data/com.example.sqliteapp/databases/* /sdcard/extracted_databases
a5y17lte:/ # cd /sdcard/extracted_databases
a5y17lte:/sdcard/extracted_databases # ls
student.db          student.db-journal

C:\platform-tools>adb pull /sdcard/Extracted_databases
/sdcard/Extracted_databases/: 2 files pulled, 0 skipped. 0.4 MB/s (33304 bytes in 0.077s)

```

Name	Type	Size
student.db	Data Base File	20 KB
student.db-journal	DB-JOURNAL File	13 KB

Figure 10. Listing all user-installed applications on an Android device using the package manager and grep commands, extracting database files of a certain application to the workstation using the adb pull command

After the database files are extracted to the workstation, they can be analyzed using forensic tools or other dedicated applications. Most Android applications use SQLite database engine. The SQLite DB Browser is a free tool that can be used to read what information is stored in SQLite database files. It allows users to see the database structure with all the tables, browse table data and execute SQL queries against the database.

For this example, a simple application with a student database is used. This database contains student names and marks. Using the SQLite browser, we can see all the data entries (figure 11). However, not all applications allow their databases to be easily reached. Some application databases use security features such as various encryption algorithms and passwords to secure their data. Forensic investigators may need to use additional tools to bypass these security features. One example is a private instant messaging application Wickr. It uses encryption to hide user identities and their messages. It also has a feature that enables users to delete their data so that it becomes virtually impossible to recover it. This type of private messaging applications may be used for criminal activities and can be important in many forensic investigations. When the Wickr application database is extracted, no file type is displayed because of the encryption.

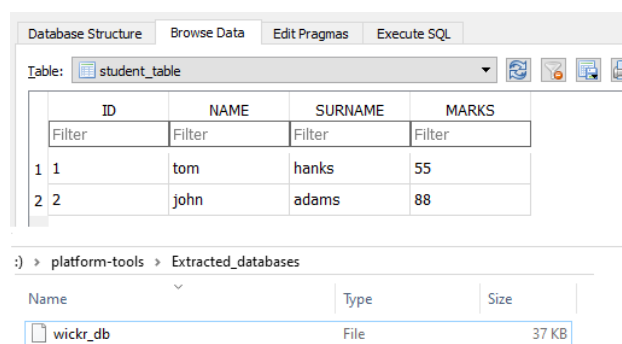


Figure 11. Browsing extracted database data using SQLite Browser. Some application databases may be encrypted.

Data extraction using ADB bridge and a remote device can be useful in many cases. If a forensic investigator needs very specific information, and knows exactly what data is required and where it is stored, he can use the command line interface to manually extract a database file to

the workstation. Scripts could also be used to do this automatically, however small differences between devices and operating system versions would need to be kept in mind. When a complex analysis for an important case needs to be performed, a full physical data backup using ADB is a great choice. Analyzing a full data backup could take a long time, but this method provides the most information.

3.3 Analysis using a native application

Another possible way to forensically analyze an Android smartphone is to use a native application. Such applications used in mobile device forensics are sometimes referred to as software agents. Software agents are small apps that are installed or copied to the device and used to analyze data locally or export required data to later analyze it on a workstation [5]. Such applications can be executed on the mobile device, locate the required data, perform some kind of data filtering procedures and provide the resulting information in an already predefined format. One of the main considerations while creating such an application is Android app data permissions - the application may access less data without superuser rights. Another challenge is dealing with many different databases files. Although most Android applications use the SQLite engine, different databases may have different structures.

Similar applications may be used to automatically collect data from a mobile device and send it to a server for further analysis. These applications can act as part of large system that monitors mobile devices and collect data relevant to not only forensic investigators, but also security auditors in an enterprise environment, application security developers and various institutions. Such systems are sometimes referred to as mobile device management systems, or MDM. One example is DroidWatch - a prototype system that consists of an Android application and an enterprise server [8]. This application was developed in 2013 as part of a research in Rochester Institute of Technology in United States. DroidWatch collects, stores and automatically sends information from an Android device to a remote server. When installed, it asks for user permission to access application data, but does not require root access. Data collected by the app is stored in a local SQLite database and can only be accessed by the app itself. This database is periodically transferred to an outside server over hypertext transfer protocol secure (HTTPS) protocol. Upon receiving the data, the remote server sends back a message to wipe the data from the local SQLite database minimize its size.

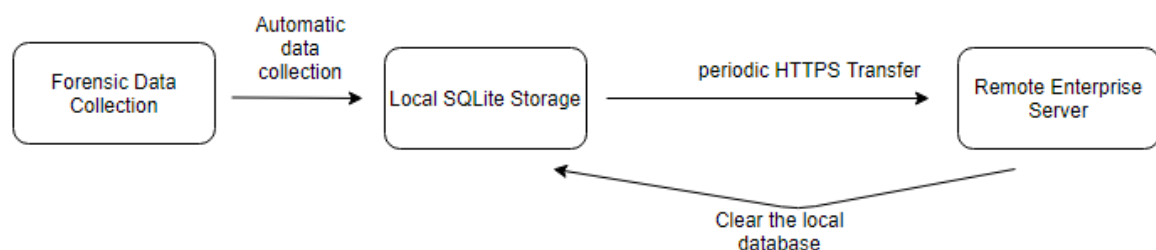


Figure 12. DroidWatch data process flow diagram

4 Prototype forensic application

4.1 Functionality

This section of the work describes a simple forensic application for Android, DataFind. The main goal of this app is to give an example of using a native application to assist a forensic investigation of a mobile device. DataFind can be installed on a smartphone and used to find acquire data related to other apps in an organized way. It automatically locates all applications currently installed on the device and generates a list of files related to them.

This is a Java application with a simple user interface. A user is presented with a list of all applications and packages that are installed on the operating system. This includes both user-installed apps and system applications such as various Google services, sensory hardware and security programs. After selecting a package, the user sees a list of all the database files used by it with their absolute file system paths (figure13). He can then choose a database and press a button to have its data displayed, or press another button to copy the database to external storage so it can be extracted for further analysis. The application also displays which apps have no files with .db extension, meaning that they either use no databases or their databases are encrypted. This application can be installed on the mobile device from the internet or using Android Debug Bridge. Like most Android apps, it can be deployed very quickly, making it useful in investigations where time is limited. Supported operating system version is Android 4.4 or higher. This, and the fact that this app is designed to find all Android packages makes this application portable.

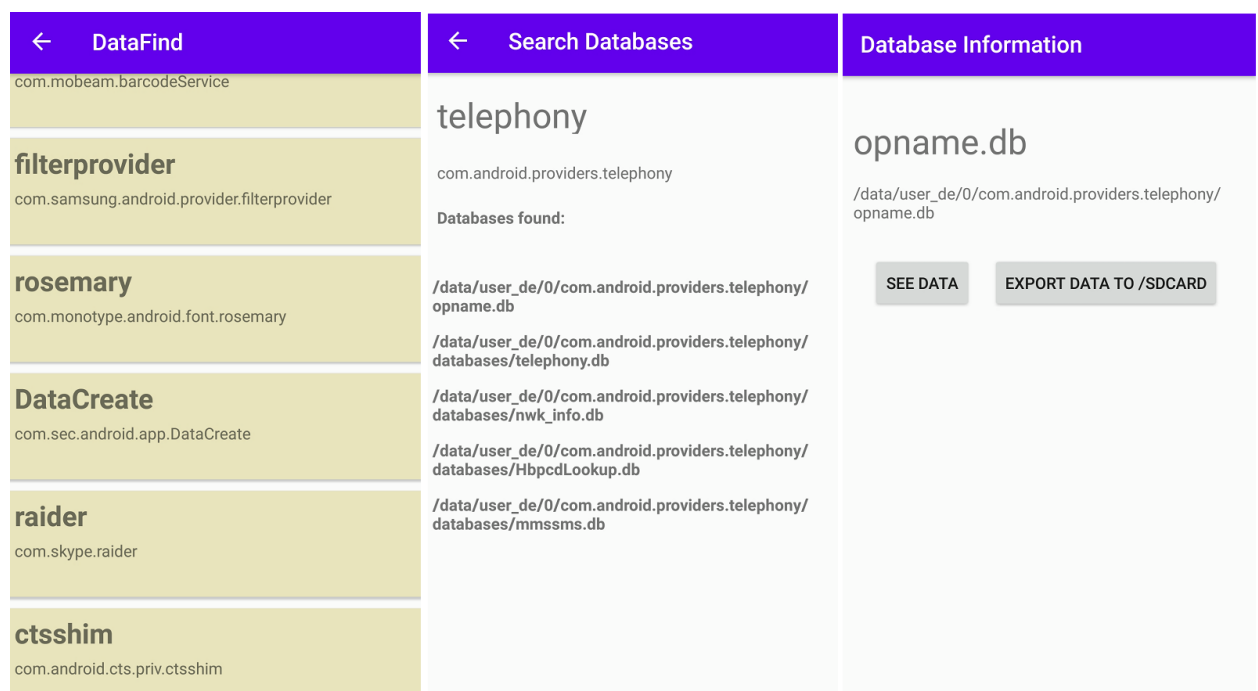


Figure 13. DataFind application user interface

4.2 Implementation

In this section I will take some time to explain how this application was developed. I will also describe some of the functions used.

4.2.1 Finding package folders

The application workflow begins with finding all packages installed on the mobile device. There are several ways to do this, either by executing Linux shell commands directly or using Java libraries. For this application I chose to use the *PackageManager* class.

The *PackageManager* class can be used to get various information about packages currently installed on the device (figure 14). This class contains the *getInstalledApplications* method, that used with the *GET_META_DATA* flag returns a list of various information about each application package. These list elements are *ApplicationInfo* objects that require another class import. Then an enhanced for loop is used to save these package names into list of strings. The *getInstalledApplication* method could also be used with the *GET_UNINSTALLED_PACKAGES* flag, to find applications that were deleted but still have some leftover data in their directories, but this practically never happens in newer android versions, so the flag is deprecated.

```
1 import android.content.pm.ApplicationInfo ;
2 import android.content.pm.PackageManager ;
3
4 final PackageManager pm = getPackageManager() ;
5 //get a list of installed apps .
6 List<ApplicationInfo> packages = pm.getInstalledApplications(
    PackageManager.GET_META_DATA) ;
7
8 for (ApplicationInfo packageInfo : packages) {
9     package_names.add(packageInfo.packageName) ;
10 }
```

Figure 14. Using *PackageManager* to retrieve information about currently installed packages

The resulting list of strings contains packages with their full names, following the naming convention similar to *com.android.<app_name>*. A few small operations are executed to create two string arrays, one containing full package names and another containing application names (figure 15). This is done in order to display the information in the user interface and conveniently use it later.

```
1 String[] packageArray = new String[ package_names.size() ];
2 package_names.toArray(packageArray) ;
3 String [] display_names = new String [package_names.size() ];
4 for( int i = 0; i < packageArray.length - 1; i++)
5     {
6         String [] package_name_split = packageArray[i].split("\\.", -2);
7         display_names[i] = package_name_split[package_name_split.length
            - 1];
8     }
```

Figure 15. Putting package information into array data structures for easier handling

The package names list is converted to the *packageArray* string array using the *toArray* function. Then a loop is used to split the full package names by the "." symbol and create a new array of application names.

4.2.2 Finding databases

Another important part of the application is the logic behind finding database files. Having a certain application name, the program needs to find full file system paths to all databases that it uses. Application databases are kept in */data*. This directory requires Root access, so a function or command must be used with elevated privileges. I chose to implement this using the *su -c* command, which is the Android variant for the Linux *sudo* permission escalation command (figure 12) [10].

```
1 Process process = Runtime.getRuntime().exec("su -c find /data -type d -
    name " + package_name + " | xargs realpath");
2 BufferedReader bufferedReader = new BufferedReader(new InputStreamReader
    (process.getInputStream()));
3 StringBuilder result=new StringBuilder();
4 String line = "";
5 while ((line = bufferedReader.readLine()) != null)
6 {
7     result.append(line + "\n");
8 }
9 output = result.toString();
```

Figure 16. Using the *su* command to real file paths from */data* directory

This code sample displays part of a function that receives a package name and returns full paths to all directories related to it. The process object is used to start a Linux operating system process that executes the shell command provided as argument for the *exec* function. The *find* command is used here to find directories in the */data* directory that belong to the package. The *xargs realpath* portion of the command makes the process return absolute paths to the directories it finds as output. This output is then read from the process using the *getInputStream()* method. The *BufferedReader* is used to read the data stream in an efficient way and later assign it to a string.

When package directory paths are retrieved, another, similar procedure is used to get database paths (figure 17). Most user installed application databases are stored in the */data/data* directory, however some apps, including many system apps may store their databases in other */data* sub-directories. Some apps can store databases in several different locations, so this implementation executes the command displayed below for each package folder to make sure that all database files are located.

```
1 Process process = Runtime.getRuntime().exec("su -c find "+ directory +
    -type f -name '*.db' | xargs realpath");
```

Figure 17. Finding filenames with the database *.db* file extension

4.2.3 Accessing data

When the application database absolute paths are located and displayed to the user, a single database can be chosen for analysis. To attempt to access the data, the *SQLiteDatabase* class

is used (figure18). An object of this class is created and the *openDatabase* method is used. The arguments used for this method are the full path to a database file on the device and a flag which determines that the database should be opened for read-only access. Once the database is opened, the *rawQuery* method is used to execute a *SELECT* statement on it, and a *Cursor* object is used to retrieve the returned data.

```
1    public void DataQuery(View view) {
2        SQLiteDatabase myDataBase;
3        myDataBase = SQLiteDatabase.openDatabase("/data/user_de/0/com.
        android.providers.telephony/databases/mmssms.db",
4            null, SQLiteDatabase.OPEN_READONLY);
5        Cursor result = myDataBase.rawQuery("SELECT name FROM
        sqlite_master WHERE type='table'", null);
6        db_data.setText(result.getExtras().toString());
7    }
```

Figure 18. attempting to access data from another application using the SQLiteDatabase class

However, upon executing this function, the program crashes. Application log provided by Android Studio displays an error message that states that the currently running application has no permission to open this database file (figure19).

```
1    2020-11-27 11:04:56.772 16965-16965/com.example.datafind E/
        SQLiteDatabase: Failed to open database '/data/user_de/0/com.
        android.providers.telephony/databases/mmssms.db'.
2    android.database.sqlite.SQLiteCantOpenDatabaseException: unknown
        error (code 1806): Could not open database
3    #####
4    Error Code : 1806 (SQLITE_CANTOPEN_EACCES)
5    Caused By : Application has no permission to open the specified
        database file.
6    (unknown error (code 1806): Could not open database)
```

Figure 19. error message showing that DataFind has no permission to open database of another application

When checking the permissions of the telephony application database folder, the command shows that all the database files allow read/write access for owner and group, but allow no access to other users. These permissions are the same for the whole */data* directory.

After further investigation, it appears that an Android application can access files located in internal storage only if they are app-specific - belong to the application [2]. One way to overcome this would be copying the database files to external storage - the */sdcard* directory, however this would greatly reduce the forensic application's performance and make lots of changes to mobile device data. Finding ways to directly access data would require searching for loopholes and further research.

Conclusions and Recommendations

Mobile device forensic investigation is a challenging process that requires a lot of knowledge. Many different subjects need to be thought about. Although there may be many different devices using Android, the data partitions and the file system is similar in most operating system versions. Mobile device security features is a significant concern during forensic investigations. Data extraction heavily depends on having full permissions to a mobile device file system, but gaining these permissions becomes increasingly challenging in newer mobile device versions.

Investigating different mobile device data extraction approaches showed that some techniques and tools could be applied in different situations. Analysis using the Android Debug Bridge proved to be a versatile approach that can be used to acquire mobile device data in several different ways. The prototype forensic Android application development process was relatively straightforward because of many learning resources and a user-friendly development environment. The tool succeeded in locating application database paths and providing them to the user in a relatively convenient way. Unfortunately, the application failed to directly access database files on the mobile device because of built-in security features.

The approach demonstrated in this work could be considered in some forensic investigation scenarios. Android apps can be installed and deployed very fast, providing a good option if time is limited. Functions and scripts used in the application for finding packages and databases can be created to work in all Android environments, making such application portable. Although this work was focused on finding databases, a similar solution could be used to analyze other files.

Further Research

There are many ways in which this solution could be improved. Firstly, the forensic and data analysis approaches described in this work could be tried on a higher number of mobile test devices with different Android operating system versions, both newer and older. Such experiments could demonstrate what approaches are still applicable as new versions are released, and what security updates of the operating system prevent data extraction. Another topic that could be researched to improve this work is the Android Linux permission model and process security features. This could help to find security loopholes that would enable data access and extraction. Lastly, more in-depth knowledge of Java programming on Android would be necessary to develop more sophisticated forensic apps and agent application solutions. Learning about various application libraries, coding practices and gaining more practical Android development experience would improve the development process.

References

- [1] Android. Android application development documentation, 2020.
<https://developer.android.com/guide/components/fundamentals>.
- [2] Android. Android application file access documentation, 2020.
<https://developer.android.com/training/data-storage/app-specific>.
- [3] Stefan Brahler. Analysis of the android architecture. *Karlsruhe institute for technology*, 7(8), 2010.
- [4] det. Cynthia A. Murphy. Developing process for mobile device forensics, 2013.
<https://digital-forensics.sans.org/media/mobile-device-forensic-process-v3.pdf>.
- [5] Felix Freiling, Michael Spreitzenbarth, and Sven Schmitt. Forensic analysis of smartphones: The android data extractor lite (adel). 2011.
- [6] Statcounter GlobalStatsl. Mobile operating system market share worldwide, 2020.
<https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- [7] Gradle user guide, 2020.
<https://docs.gradle.org/current/userguide/userguide.html>.
- [8] Justin Grover. Android forensics: Automated data collection and reporting from a mobile device. *Digital Investigation*, 10:S12--S20, 2013.
- [9] Privacy International. A technical look at phone extraction, 2019.
<https://privacyinternational.org/long-read/3256/technical-look-phone-extraction>.
- [10] Jorrit Jongma. An in depth guide on the su command in android, 2016.
<https://su.chainfire.eu/>.
- [11] Max Lee. Various instructions and resources for gaining root access, 2020.
<https://highonandroid.com/android-root-101-series-highonandroid/>.
- [12] Rohit Tamma Oleg Skulkin, Donnie Tindall. *Learning Android Forensics: Analyze Android devices with the latest forensic tools and techniques, 2nd Edition*. Packt, 2018.
- [13] Radhika Padmanabhan, Karen Lobo, Mrunali Ghelani, Dhanika Sujan, and Mahesh Shirole. *Comparative analysis of commercial and open source mobile device forensic tools*. 2016.
- [14] Heather Mahalik Rohit Tamma, Oleg Skulkin. *Practical Mobile Forensics - Third Edition: A hands-on guide to mastering mobile forensics for the iOS, Android, and the Windows Phone platforms*. Packt, 2018.
- [15] Nathan Scrivens and Xiaodong Lin. *Android digital forensics: data, extraction and analysis*. 2017.
- [16] Kolin Stürt. Hack an android app: Finding forensic artifacts, 2019.
<https://www.raywenderlich.com/3419415-hack-an-android-app-finding-forensic-artifacts>.