Forensic Support for Android Device

MINI PROJECT REPORT

Submitted by

KARTHIK M. RAO   CB.EN.P2CYS14006

in partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY
IN
CYBER SECURITY

श्रद्धावान् लभते ज्ञानम्

TIFAC-CORE IN CYBER SECURITY

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

DECEMBER 2015

Forensic Support for Android Device

MINI PROJECT REPORT

Submitted by

KARTHIK M. RAO CB.EN.P2CYS14006

in partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY
IN
CYBER SECURITY

Under the guidance of

Prof. Prabhaker Mateti
Associate Professor
Computer Science and Engineering
Wright State University
USA

TIFAC-CORE IN CYBER SECURITY

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

DECEMBER 2015

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING,COIMBATORE -641 112

BONAFIDE CERTIFICATE

This is to certify that this mini project report entitled **"Forensic Support for Android Device"** submitted by **KARTHIK M. RAO (Reg.No : CB.EN.P2.CYS14006)** in partial fulfillment of the requirements for the award of the **Degree of Master of Technology** in **CYBER SECURITY** is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering.

Dr. Prabhaker Mateti	**Dr. M. Sethumadhavan**
(Supervisor)	(Professor and Head)

This mini project report was evaluated by us on...............

INTERNAL EXAMINER	EXTERNAL EXAMINERS

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ENGINEERING,COIMBATORE
TIFAC-CORE IN CYBER SECURITY

DECLARATION

I, KARTHIK M. RAO (Reg.No: CB.EN.P2.CYS14006) hereby declare that this mini project report entitled **"Forensic Support for Android Device"** is a record of the original work done by me under the guidance of **Prof. Prabhaker Mateti**, Associate professor, Wright State University, and this work has not formed the basis for the award of any degree / diploma / associateship / fellowship or a similar award, to any candidate in any University, to the best of my knowledge.

Place : Coimbatore

Date : Signature of the Student

COUNTERSIGNED

Dr. M. Sethumadhavan
Professor and Head, TIFAC-CORE in Cyber Security

ABSTRACT

Rao, Karthik. M.Tech. Department of Cyber Security, Amrita Vishwa Vidyapeetham, 2015. Forensic Support to Android Device.

The growing advanced features of the smartphones catch the attention of criminal to perform criminal activity. Information placed in a smart phone is plays a crucial role the course of a criminal investigation and while presenting in court of law. An ordinary man with an ordinary mobile device easily gives away his/her contact details, call history and SMS, just by looking at the device. Smart phones contains even more useful information such as social network messages, E-mails, network connections, browser history etc. Such a device in hands of professional criminal is a bad news, but becomes a good news to a forensic invigilator. A knowledgeable criminal might modify or wipe the traces of activities and the mobile data. This data is harder to retrieve once the user deletes it. We propose forensic support to the Android ROM, which monitors all the user activities and stealthily stores it in the cloud. Other than just activity this thesis propose in adding a keylogger and call tapping facility.

Keywords: Android, Forensics, Android Framework, adb tool, Binder

Contents

List of Tables

List of Figures

1 Introduction

The arena of mobile forensics has detonated recently and now it has been one of the most promising areas of research. Primarily, the potentials of cellphones have been greatly enhanced. Mostly because these devices are probably always turned on and, let's face it, mostly in hand. Hence, they continuously record our movements and actions to provide great information about our behavior. Interaction on a cellphone is very unlike to a traditional computer.

Cell phone forensics was always important but never given a prominence. When these devices introduced was only for the call or for text and this information was available for cops with help of the Service providers. Interesting was when Internet was added to cell phones, forensics importance to investigations went off the charts. Higher was the demand so rose the need of better forensic software. Out of nowhere, the higher amount of evidence was available, including Email, Internet searches, and social networking activity. These days almost every computer forensics team/lab has a specifically set aside cell phone forensic team/lab. The duties performed are quite different. Though the standard procedure of forensics is still the same.

There are a lot of the issues with cell phones devices like encrypted mobile platforms, the Burner phone, a plethora of operating systems, etc. An investigator working with a PC will usually come across with OS like Microsoft Windows or Apple's Mac OS X. On other hand investigators who investigates a cell phone encounters an Android, iOS, Windows, Blackberry OS, RIM, Symbian, or others. The competitive market for Android and iOS devices means that the user synced the device to the computers in the home and/or at work, and to the cloud as well. Android undisputedly holds the market shares according to the Gartner 2015 Q2 report[Inc. 2015].

Right from Hello Kitty to Whatsapp to online banking transactions, from kids to youngsters to working professionals, Android provide an immense amount of apps from its chest named Play store. By 2015 about 1.43 million apps published and over 50 billion downloads according to Wikipedia. A high-end android device can do almost all computing which is done by a normal laptop computer

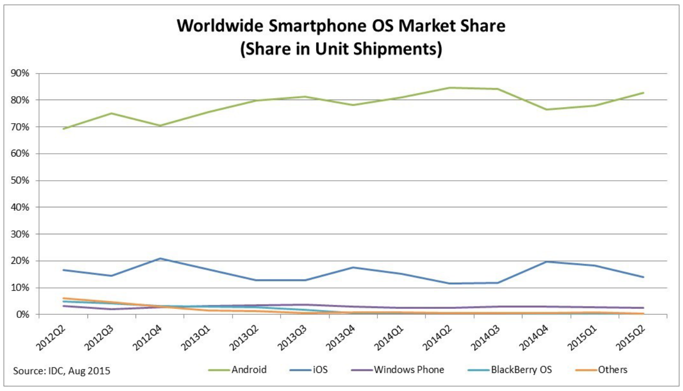


Figure 1.1: Smartphone OS Market Share, 2015 Q2

and much more advanced features like NFC, Global Positioning etc. This increased availability of computing power for smartphone catches the attention of criminals. Where a common man uses the device to explore its immense functionality, a criminal uses it to exploit the common man with it. This opens a wide range of path for forensics.

1.1 Known things about Android

The word *Android* refers to a robot or synthetic organism designed to look and act like a human basically means humanoid robot. In the mobile domain, it denotes to a company, an operating system, an open source project, and a development community. At the start developed by Android, Inc. later Google bought it in 2005.

Though Android celebrates birthday on November 5, 2007, which is the date Android beta was released, it was not until September 23, 2008, version 1.0, was released as the first commercial version. The only version without name were 1.0 and 1.1 rest all followed nomenclature of tasty treats like Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Icecream sandwich, Jelly Bean, KitKat, Lollipop, and recently launched Marshmallow. Android is a mobile OS which is based on the Linux kernel and intended mostly for touchscreen devices. In addition to these devices, Google has further developed Android TV for televisions, Android Wear for wrist watches, and Android Auto for cars, each with a specialized user interface.

Lets skip the history lessons and get into the real stuff. Android provides a application framework that allows oneself to build apps and games for mobile/tablet devices in a Java language environment. The Android operating system consists of a stack of layers running on top of each other. Android architecture can be best understood by taking a look at what these layers are and what they do. The following diagram[Wikipedia 2012] shows the various layers involved in the Android software

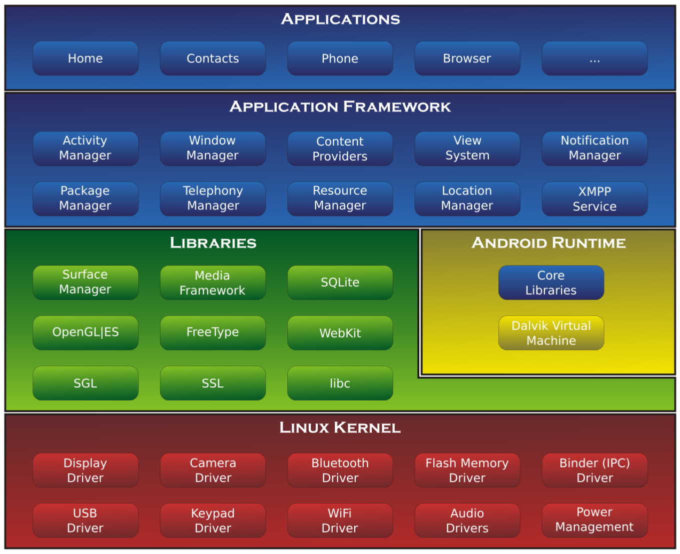


Figure 1.2: Android's architecture diagram

stack:

Linux Kernel: Android OS is built on top of the Linux kernel with some design changes. The Linux kernel is positioned at the bottom of the software stack and provides a level of abstraction between the device hardware and the upper layers. All the core functionalities of Android, such as process management, memory management, security, and networking, are managed by Linux kernel. Linux is a recognized platform when it comes to security and process management.

Libraries: On top of Linux kernel are Android's native libraries. The device handles different types of data with these libraries. These libraries are written in the native C or C++ programming languages and are specific to a specific hardware. Surface Manager, Media framework, SQLite, WebKit, OpenGL, and so on are some of the most important native libraries.

Android Runtime: Android used Dalvik as a process virtual machine with trace-based just-in-time (JIT) compilation to run Dalvik "dex-code", which is usually translated from the Java bytecode. Android 4.4 introduced Android Runtime (ART) as a new runtime environment, which uses ahead-of-time (AOT) compilation to entirely compile the application bytecode into machine code upon the installation of an application.[Yaghmour 2013]

Application framework: Android applications are executed and managed with the help of an Android application framework. It is responsible for performing many crucial functions such as resource management, handling calls, and so on. The Android framework includes the following key services, Activity manager, Content providers, Resource manager, Notifications manager, View system, Package manager, Telephony manager, and Location manager

Applications layer: The topmost layer in the Android stack consists of applications which are programs that users directly interact with. There are two kinds of apps

- **System apps**: These are the applications that are pre-installed on the phone and are shipped along with the phone.
- **User-installed apps**: These are the applications that are downloaded and installed by the user from various distribution platforms such as Google Play.

1.2 Motivation

Most OSs are designed with an intension of general purpose use, not for forensics and Android is no exception. Let's imagine the following scenario; Given that criminals and terrorists use smartphones, what is an Android ROM capable of? What's new techniques and stealth services could it include that could be used to gather forensically sound data? This thesis is an exploration of these questions.

According to layman, perspective mobile contains the information such as call history, contacts and text message data, but as for an investigator, it holds more information which includes the information about his/her whereabouts, networks one connected, his/her search history, his social network history. Now, what happens if he/she has deleted that information? A Basic Forensic methodology fails to get those deleted data may be obsolete data or previous data was overridden. Here we propose a proactive forensic service that runs on Android ROM such that it tracks the user activities, file system changes and transfers it to a cloud server in stealth mode. We also propose client software that runs on Linux machine which connects the device through the `ADB` tool, which is responsible for advanced imaging, recovery and analysis of device data.

1.3 Problem Statement

As Android is dominating the market, it is becoming favorite target platform of hackers/criminals. There are many forensic tools available, these are primarily targeted on application database files and to extract the deleted data is difficult. Usually, data which are used by messaging and mail application stores in an encrypted format and to extract the data is harder with the usual forensic tools. A well knowledgeable criminal can erase the forensic data from the mobile device. There are

Mobile applications like Uninstall-It which can delete all the application data from the device even the leftover files. Here, two services are designed with the expectation that an Android device will be subjected to forensic investigation. We leave the task of convincing a suspected criminal to use a forensics-proactive Android device to others.

1.4 Aim

A proactive forensic service that runs on Android ROM such that it tracks the user activities, file system changes and stores this information to a cloud server in stealth mode. A client software that runs on Linux machine which connects the device through the `ADB` tool, which is responsible for advanced imaging, recovery and analysis of device data.

1.5 Organization

The thesis is organized to six chapters and Appendix. The thesis discusses the design and implementation of our Forensic Support Module and also the retrival tool to collect user specific data on Android device. Chapter 2 provides all needed background knowledge to understand Android forensics. It gives an overview about Android File system and Application data storage. Chapter 3 discusses the architecture, data collection methodology and implementation of our proactive forensic support module as well as the data collection tool. Chapter 4 discusses the related works of the project. Chapter 5 discusses the results and evaluation of the project. Chapter 6 discusses conclusion and future work.

2 Background

There are two operating system design: The traditional monolithic kernel approach, which is based on implementing the operating system entirely in supervisor mode and using system calls as the main interface to OS services, and the micro-kernel approach, which relies on the heavy use of message passing between user applications and user space based system servers that serve, in turn, as the bridge to a minimalist kernel[Artenstein and Revivo 2014]. Android has micro-kernel architecture. This section describes the background of Android and Forensic for our work.

2.1 Android Partitions

Ext4 is the most common file system on modern android devices. Older devices may use older ext* versions as well, or other file systems like YAFFS2. Since everything is built on Linux, ext4 a pretty sensible choice, with solid kernel support and a good track record. Android uses several partitions to organize files and folders on the device. Each of these partitions has a distinct role in the functionality of the device. In this subsection, we will take you on a tour of Android partitions, what they comprise and what can be the possible consequences of modifying their content. Android devices have major two memory partitions[Raja 2011]

- **Internal Memory partitions**
  - **/boot**: This is the partition that enables the phone to boot. It includes the kernel and the ramdisk. Without this partition, the device will basically not be able to boot. Obliterating this partition from recovery should only be done if absolutely required and once done, the device must NOT be rebooted before installing a new one, which can be done by installing a ROM that includes a /boot partition.
  - **/system**: This partition basically contains the entire operating system, other than the

kernel and the RAM disk. This includes the Android user interface as well as all the system applications that come pre-installed on the device. Clearing this partition will remove Android from the device without rendering it unbootable, and one will still be able to put the phone into recovery or bootloader mode to install a new ROM.

  - **/recovery**: The recovery partition can be deliberated as a substitute boot partition that lets you boot the device into a recovery console for performing superior recovery and maintenance manoeuvres on it.
  - **/data** Also called userdata, the data partition contains the user's data – this is where the contacts, settings, messages, and apps that one has installed go. Clearing this partition essentially performs a factory reset on the device, restoring it to the way it was when one first booted it, or the way it was after the last official or custom ROM installation. When one performs a wipe data/factory reset from recovery, it is this partition that one is wiping.
  - **/cache** This is the partition where Android stores frequently accessed data and app components. Clearing the cache doesn't effect ones personal data but simply gets rid of the existing data there, which gets involuntarily reconstructed as one continues using the device.
  - **/misc** This partition contains miscellaneous system settings in form of on/off switches. These settings may include CID (Carrier or Region ID), USB configuration and particular hardware settings etc. This is a valuable partition and if it is corrupt or missing, several of the device's features will not function normally.

- **SD card Memory partitions**
  - **/sdcard** This is not a partition on the internal memory of the device but rather the SD card. In terms of usage, this is ones storage space to use as one see fit, to store your media, documents, ROMs etc. on it. Wiping it is perfectly safe as long as one backup all the data that one require from it to the computer first. Though several user-installed apps save their data and settings on the SD card and obliterating this partition will not leave behind all that data.
  - **/sd-ext** This is not a standard Android partition but has turned out to be popular in the custom scene. It is basically an additional partition on ones SD card that acts as the /data partition when used with certain ROMs that have special features called APP2SD+ or data2ext enabled. It is especially useful on devices with little internal memory allotted to the /data partition. Thus, users who want to install more programs than the internal memory allows can make this partition and use it with a custom ROM that assists this

attributes, to get bonus storage for installing their apps. Wiping this partition is essentially the same as wiping the /data partition – you lose your contacts, SMS, market apps and settings.

2.2 Data Storage in Android

Android provides developers with five methods for storing data to a device. Forensic examiners can uncover data in at least four of the five formats[Hoog 2011]. Android's sandboxing makes sure that each application is assigned with a different user-id and no other application to has access to that data. Files stored in storage can be accessed by any application. Persistent data are stored in either the NAND flash, the SD Card, or the Network. The specific five methods are:

1. **Shared Preferences**: allows a developer to store key-value pairs of primitive data types in a lightweight XML format. Primitive data types that can be stored in the preferences file are

   (a) boolean

   (b) float

   (c) int

   (d) long

   (e) strings

   Shared preferences files are typically stored in an application's data directory in the shared_pref folder and end with .xml. Below shows the shared preferences of the Karbonn Sparkle V:

```
root@Sparkle\_V\_sprout:/data/data/com.android.phone/shared\_prefs # ls -l
-rw-rw---- radio   radio   130 2015-09-11 14:34 _has_set_default_values.xml
-rw-rw---- radio   radio   219 2010-01-01 05:30 com.android.phone_preferences.xml
```

2. **Internal Storage**: The files are stored in the application's /data/data subdirectory and the developer has control over the file type, name, and location. By default, the files can be only be read by the application and to view the files one must posses the root access. The below shows the files saved on the maps.

```
root@Sparkle_V_sprout:/data/data/com.google.android.apps.maps # ls -l
drwxrwx--x u0_a51   u0_a51   2015-11-17 08:36 app_
drwxrwx--x u0_a51   u0_a51   2015-12-14 10:18 cache
```

```
drwxrwx--x u0_a51   u0_a51   2015-09-11 14:35 databases
drwxrwx--x u0_a51   u0_a51   2015-11-18 12:14 files
lrwxrwxrwx install  install  2015-12-13 11:45 lib ->
             /data/app/com.google.android.apps.maps-1/lib/arm
drwxrwx--x u0_a51   u0_a51   2015-11-17 08:36 shared_prefs
```

3. **External Storage**: Files on the various external storage have very few constraints when compared to internal storage. Data could be used from one device to another just by plug-and-play. SD cards generally FAT32 file systems.

```
root@Sparkle_V_sprout:/mnt/sdcard/Android/data/com.google.android.apps.maps #
ls -l
drwxrwx--- u0_a51   sdcard_r         2015-12-14 10:18 cache
drwxrwx--- u0_a51   sdcard_r         2015-11-17 08:36 files
drwxrwx--- u0_a51   sdcard_r         2010-01-09 17:00 testdata
```

4. **SQLite Database** Databases are used for structured data storage and SQLite is a popular database format. The Android SDK provides dedicated APIs that allow developers to use SQLite databases in their applications. The SQLite files are generally stored on the internal storage under /data/data/<packageName>/database.

Application	**Location in the phone**
Phonebook	com.android.providers.contacts/databases/contacts2.db
Call History	com.android.providers.contacts/databases/contacts2.db
Browsing History	com.android.browser/databases/browser.db
Calender	com.android.providers.calendar/databases/calendar.db
SMS	com.android.providers.telephony/databases/mmssms.db
WhatsApp	com.whatsapp/databases/msgstore.db
GMail	com.google.android.gm/mailstore.<username>@gmail.com.db

Table 2.1: SQLite Databases

5. **Network** One can use the network to store and retrieve the data from/to phone. To perform these operations, classes from the below-mentioned packages can be used

   - `java.net.*`
   - `android.net.*`

2.3 Binder

The Binder is an RPC/IPC mechanism developed for Android based on OpenBinder concept. It allows apps to communicate the System Server, and to communicate to each others, although the app doesn't actually communicate to the Binder directly. Instead, they use the proxy and stubs generated by the aidl tool. The developer never actually sees that Binder is being used. The in-kernel driver part of the Binder mechanism is a character driver accessible through `/dev/binder`. It is used to transmit parcels of data between the communicating parties using calls to `ioctl()`[Yaghmour 2013].

2.3.1 Binder Mechanism

Binder spans over the application layer, middleware and kernel. The Binder middleware and API are together known as Binder Framework and the Binder driver is known as Binder Kernel. Each higher layer gives an abstraction of the operations on the lower layer as displayed in figure below. The objective of the Binder IPC is that it fulfills the request, if permitted and allowed, of another app. Basically, the code of a typical service in Android is divided into two parts the service itself and its interface. Each interface has two implementations, a proxy on the client side and a stub on the server side. A client process calls, let's say, a service, is when the proxy interface of the service gets invoked. The proxy interface will in turn invoke the Binder calls to the server. At the server, the stub will be waiting for incoming requests, which will then call the actual implementation on the server[Kaladharan et al. 2016]. The proxy and stub interfaces are implemented using Android Interface Definition Language(AIDL).

Binder transfers data in the form of a structure known as `parcel`. It is marshaled before being sent.

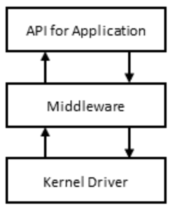


Figure 2.1: Basic diagram of Binder Layers

2.3.2 Binder API

An application, let's say, Gmail that has a registered itself as a service. Another application Facebook wishes to access this service. The `/src` of application Gmail contains an `.aidl` file, which has the signatures of the functions that it defines. Once the application is built, the `.java` will contain a class named stub. The service will then create an object of the stub and will then define the methods in it.

At the client side, `bindService(Intent myservice, ServiceConnection conn, int flags)` is invoked. When `bindService` is called, at the server end another method known as `onBind()` is invoked. This method, in turn, will invoke the `onServiceConnected()` with the `mBinder` object as its argument.

2.3.2.1 Middleware Binder

The communication between two processes in Android is known as the transaction. The client calls the `transact()` using the `IBinder` object.The data transmitted is in a structured form known as Parcel. Hence, the parcels need to be flattened before being sent[Artenstein and Revivo 2014]. The Binder middleware functionalities are achieved by a shared library `libbinder.so`. The client will copy the request to a kernel space, from where the server will obtain it. The server after performing the request will copy the response back to the same kernel space. The client will then copy it using kernel drivers.

2.3.2.2 Kernel Binder

Once the Binder Framework creates the parcels to send, an `ioctl` call is made from within the `transact()`. The syntax goes like `ioctl(fd, BINDER WRITE READ, &bwt)`, where `fd` is a file descriptor to /dev/binder, `BINDER_WRITE_READ` is a data structure and `&bwt` is a reference to it. The Binder driver will identify the server, copy the request to the server's address space and wakes up a waiting thread in the server to handle the request. This thread invokes a thread from the pool. The `onTransact()` of middleware is unmarshaled and requested operation is performed.[Kaladharan et al. 2016]

2.4 Forensic Analysis Techniques

This section provides an overview of the analysis techniques followed in forensic world.

2.4.1 Timeline Analysis

Timeline analysis is a key component to any investigation as the timing of events is always relevant. There are many tools in the market to do this, some are The Sleuth Kit and log2timline. The primary source of timeline information is the file system metadata including the modified, assessed, changed and created.

2.4.2 File System Analysis

There are a number of directories that needs to be examined for an investigation. The file system provides where exactly to look for the particular kind of file. Best way is to determine which file systems are mounted on the system, where they are mounted, and what type are they. Below shows the mount command snippet for Karbonn Sparkle V.

```
rootfs / rootfs ro,seclabel 0 0
tmpfs /dev tmpfs rw,seclabel,nosuid,relatime,mode=755 0 0
devpts /dev/pts devpts rw,seclabel,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,seclabel,relatime 0 0
[...]
```

Number of file system changes from device to device. The file systems present in Android can be divided into three main categories, which are as follows:

1. Flash memory file systems Flash memory is a constantly-powered nonvolatile memory that can be erased and reprogrammed in units of memory called blocks.
   - Extended File Allocation Table (exFAT)
   - Flash Friendly File System (F2FS)
   - Journal Flash File System version 2 (JFFS2)
   - Yet Another Flash File System version 2 (YAFFS2)
   - Robust File System (RFS)
2. Media-based file systems Android devices typically support the following media-based filesystems:
   - EXTended file system (EXT2/EXT3/EXT4)
   - File Allocation Table (FAT)

- Virtual File Allocation Table (VFAT)

3. Pseudo file systems Pseudo file systems are file systems that can be thought of as a logical groupings of files. Here are some of the important pseudo file systems found in an Android device

   - **control group (cgroup)**: This type of pseudo file system provides a way to access and define several kernel parameters.
   - **rootfs**: This type of file system is one of the main components of Android and contains all the information required to boot the device.
   - **procfs**: This type of file system contains information about kernel data structures, processes, and other system-related information in the `/proc` directory.
   - **sysfs**: This type of file system mounts the /sys folder, which contains information about the configuration of the device.
   - **tmpfs**: This type of file system is a temporary storage facility on the device that stores the files in RAM (volatile memory).

2.4.3 File Carving

File carving is a process in which specified file types are searched for and extracted from the device. File carving works by examining the binary data and identifying files based on their known file headers. Traditional file carving techniques expect the data to be sequential in the image and this technique cannot produce the full file as if it is fragmented. Newer file carving techniques are being developed to address the limitations with file fragmentation, call SmartCarving[Hoog 2011]. One basic tool is the scalpel.

2.4.4 Strings

The strings command will extract ASCII printable strings from any file, text or binary. It is quite effective at quickly examining binary data to determine if information of interest might be contained in the file.

```
strings -all --radix=x disk1.img
```

```
2342159b 4mp3

234215a0 downloa

234215ac hindi

234215b3 bluray, @telugu
```

```
234215cb hd v
```

2.4.5 Hex

A hex editor is a type of computer program that allows for manipulation of the fundamental binary data that constitutes a computer file. It is helpful when some cases require a deeper analysis to find deleted data or unknown file structure. Sample output of the msgstore.db.crypt8 in Hex Editor:

```
File: msgstore.db.crypt8          ASCII Offset: 0x00000000 / 0x001B72D2 (%00)
00000000  00 01 01 B5  CC 3F 44 2E   27 DA 40 B7  AB FE B7 15   .....?D.'.@.....
00000010  A2 00 07 1B  93 0B 54 AD   C1 91 F4 D1  42 48 CC 97   ......T.....BH..
00000020  0F 11 1E FC  DD FC 0E BA   A5 4E 94 4D  CD 81 46 21   .........N.M..F!
00000030  B5 6D A4 35  C3 07 AC 8C   C1 8E 7F 17  45 7B D8 77   .m.5........E{.w
00000040  F7 45 8F 50  00 E3 4B 4B   62 19 EF B5  DD EF 33 06   .E.P..KKb.....3.
[...]
```

2.5 Monitoring File and Directory Changes

In this subsection will look upon file monitoring tools supported in Linux and Android. Linux support many file monitoring tools like inotify, incorn, iwatch, lsyncd. Though android supports notify command and FileObserver class, there is always a way around to write API.

2.5.1 FileObserver

Android provides an API for monitoring file events which is FileObserver. It is based on inotify file change notification system in Linux. It is an abstract class,each FileObserver instance monitors the file or a directory associated with it. It is not recursive, i.e. if a directory is monitored, then only the files and sub-folders inside it are monitored, but folders and files inside subfolders are not.

2.5.2 inotify

Inotify (inode notify) is a Linux kernel subsystem that acts to extend file systems to notice changes to the file system, and report those changes to applications. It replaces an earlier facility, dnotify, which had similar goals[Wikipedia 2015a].

2.5.3 Other file monitoring tools

There are many tools available in Linux for file monitoring File Alteration Monitor, dnotify, incorn are some of them.

File Alteration Monitor, also known as FAM subsystem allows applications to watch certain files and notifies when they are modified. It enables applications to work on a greater variety of platform. During the creation of a large number of modification it drags the entire system.

dnotify is a file system event monitor for the Linux kernel, one of the subfeatures of the fcntl call. It was introduced in the 2.4 kernel series. It has been obsoleted by inotify.

Incorn program is an "inotify cron" system consisting of a daemon and a table manipulator. One can use it a similar way as the regular cron. The difference is that the inotify cron handles filesystem events rather than time periods.

The RecursiveFileObserver API has to be created so that it can even monitor the recursive folders and files.

3

Proposed System

Here, we are proposing of adding a forensic support service to the Android ROM[Mateti et al. (2015)]. This service proactively identifies, prioritizes, collects and secretly stores the forensically sound data initially on a stealth file system within the device, that is opportunistically uploaded to a cloud server. This section is an overview of the architecture (Figure 3.1). The next section describes a design and an initial implementation. This is similar architecture mentioned in the Proactive Forensic Support[AIYYAPPAN 2015].

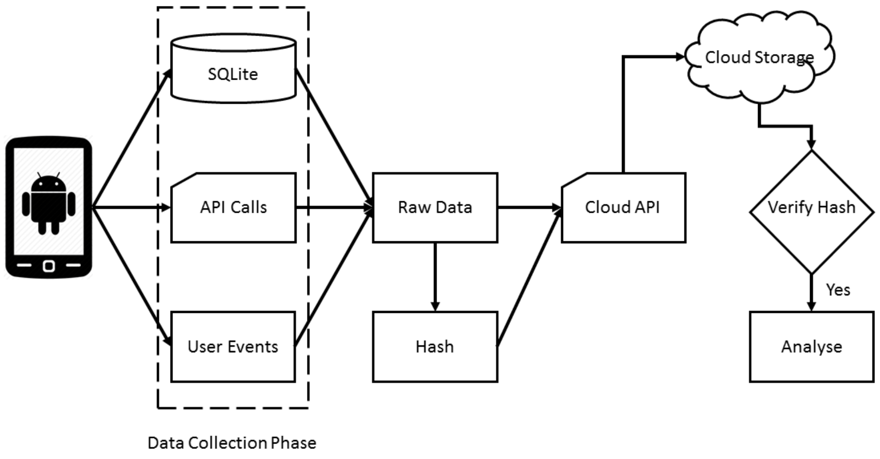


Figure 3.1: Proactive Forensic Support Architecture

The architecture contains three phases; data collection where, as the name says, data is retrieved. The details are discussed in forthcoming topics. Hashing the entire stored data to check for integrity. Linux tool for extracting device/cloud data.

3.1 Architecture of Forensic module

The forensic module we proposed gets data mainly from three sources, the database files(SQLite database files), API calls and the file monitoring tool. These data gets stored in a separate file partition of the device `/forensic`, and this data resides in the device until it gets the chance to upload it to the cloud server.

3.1.1 Data Collection Phase

Data collection plays a vital role from the forensics point of view. The data collected/retrieved is the evidence through which a investigator claims the reality of what crime has been done by which criminal. Android application stores its private data using SQLite database and these files. For example, phone logs are stored in the **contacts2.db** file. We collect these SQLite file in a scheduled manner so that no data should is forgotten when investigating. User activities like camera, network usage, and GPS locations are also collected by using Android provided APIs. Some of the Android APIs are in the below. Filesystem events get captured by using a tweaked version of `inotify` tool.

User Data	API
Phonebook	ContactsContract.CommonDataKinds.Phone.NUMBER
Call History	android.provider.CallLog.Calls.CONTENT_URI
Camera	android.permission.CAMERA
Alarm Clock	android.provider.AlarmClock
Bluetooth	android.permission.BLUETOOTH
GPS Location	android.permission.ACCESS_FINE_LOCATION
Fingerprint	android.permission.USE_FINGERPRINT

Table 3.1: Some Android APIs

The data collection phase can be divided into two categories:

1. Logical Extraction The term logical extraction is typically used to refer to extractions that do not recover deleted data, or do not include a full bit-by-bit copy of the evidence. However, a more correct definition of logical extraction, is any method that requires communication with the base operating system.[Learning Android Forensics] Most of the time any and all user data may be recovered logically:

    - Contacts
    - Call logs

- SMS/MMS
- Application data
- System logs and information

The majority of this data is stored in SQLite databases. Logical extraction technique in android does not require root access to the device.

2. Physical Technique A physical extraction is an exact bit-for-bit image of the electronic media, and this definition remains true for mobile devices too. Forensic techniques that acquire physical images of the targeted data storage typically result in more data being recovered. As a physical acquisition is an exact image of the device, every bit of data on the device is in the image file.

3.1.2 Data Storage Phase

The collected data is stored on the newly created device partition until it gets connected to the Internet. The partition is root owned and this cannot be accessed by any normal applications. The data stored are opportunistically uploaded to the cloud server using a cloud API residing in the device.

3.1.3 Hashing

The collected files are compressed, divided into smaller chunks and hashed. Each compressed file and hash are sent to cloud server to maintain the integrity. The chunks of compressed are hashed. These hash can be calculated based on the functionality available in Java.

```
java.security.MessageDigest.getInstance("MD5");
```

3.2 Real-Time Data Collection

The Data Collection module of the forensic service collects the forensic relevant data and stores them in a tiered stealth file volume. Two companion projects help us. The Network Ombudsman[GEORGE 2015] project gathers all (wlan0, eth0, Bluetooth, 2G/3G/4G/LTE, NFC, etc) network events. The Android Audit[Jamsheed 2015] project logs all non-network events. Both upload to cloud storage.

3.3 Opportunistic Upload of Big Data

Forensic data can accumulate to a very larger extent. As soon as the device is connected to the Internet, the data is secretly transferred to a cloud server. Once the data is uploaded to the cloud, it is cleared from the device. The data upload is done only if the device has the charge of 60% or more or it is in charging mode. The device uploads the data in the WiFi session because if the forensic module uses the mobile data, the user might/will get concerned. The battery as well as the processor usage might attract the user attention, so the cloud upload is only done when the device is idle state.

3.4 Real Time Analysis

Most of the android application encrypts its data and databases. Forensic data can be collected only by analyzing or decrypting those application data. No real time analysis is done on the Android device because of the processing speed, consumption of the battery and slow down of the system. The data can be collected even the absence of the device' physical location. Location of the device is tracked using the GPS location. The data uploaded are tracked with the timestamps to provide provenance of the device to the investigator. Also real time upload is possible if the device if it is in the range of WiFi with sufficient battery back up.

3.5 Stealth Challenges

Since the forensic module utilizes additional device storage space, Internet connection and some amount of processor speed. There should be a covert operation to hide these activities from user. Main challenge is building an undisclosed partition. Other is to hide the forensic process from process listing. Cloud upload is also a challenging job and this has to be done in an opportunistic way so that the user must not concern or notices it.

Stealth: The device that has forensic support enabled should not be visible or interrupt the owner while performing regular actions. Only APKs with root permissions can detect the forensic partition. A major future goal is to make this detection very difficult.

Hiding the forensic service is not easy. Linux internals transact with `proc` and `sysfs` needs to be altered. This can be done exploring `hidepid` command.

Opportunistic Uploads waits for opportune moment to transfer the forensically sound data i.e, availability of the WiFi. There is always a risk of device being running out of memory so to mitigate this the data is deleted from the device as soon as it is sent to cloud.

3.6 Hiding Process

There are several process listing commands and APKs to list running processes in linux systems. These tools uses procfs filesystem to get these data. The forensic process is hidden from user by using `hidepid` option. The values are as follows:

`hidepid=0` - The classic mode - anybody may read all world-readable `/proc/<pid>/*` files (default).

`hidepid=1` - denotes users may not access any `/proc/<pid>/` directories, but their own. Sensitive files like cmdline, sched*, status are now protected against other users.

`hidepid=2` - denotes `hidepid=1` plus all `/proc/<pid>/` will be fully invisible to other users. It doesn't signify that it hides a verity whether a process with a specific pid value exists, but it hides process' `uid` and `gid`, which may be learned by stat()'ing /proc/<pid>/ otherwise[GITE 2014].

3.7 Keystroke logging

Keystroke logging, habitually signified to as keylogging or keyboard capturing, is the action of recording (logging) the keys struck on a keyboard so that the person using the keyboard is unaware that their actions are being monitored.

3.7.1 Using Swiftkey

SwiftKey Keyboard is among the top paid app in the Play store at the moment and it's a great app. The malware-ridden Android app is taking a serious risk of a keylogger being inserted[Casey 2013] and people tracking all their passwords, Google searches and Credit Card numbers. The basic knowledge of java is sufficient to create the app and make it as default keyboard.

3.7.2 Using Man-in-the-Binder

The Man-In-The-Binder exploit was described in [Artenstein and Revivo 2014]. This attack is analogous to the man-in-the-middle attack, where the attacker code stands at the middle of a communication. The traditional approach is to replace a malware with built in keyboard. This can be easily detected, even by a normal user. Using a Man in the Binder attack would be a much more robust and stealthy solution. To receive keyboard data, an application has to register with an Input Method Editor (IME) server. The default IME in most Android is `com.android.inputmethod.latin`. We no longer need to intercept the data going down from the application into Binder - we have to intercept the reply tossed up from Binder to the app.[Artenstein and Revivo 2014]

Though its not been decide which approach to use, as a completeness of work on both is still under progress.

3.8 Call Recording

Telephone tapping is the monitoring of telephone and Internet conversations by a third party, often by covert means. The recorded will be saved .amr (`MediaRecorder.OutputFormat.RAW_AMR`) files and are located in the folder "recordedCalls" in /forensics storage.The Android SDK includes a `MediaRecorder` class, which one can leverage to build audio recording functionality. Doing so enables a lot more flexibility, such as controlling the length of time audio is recorded for.

The `MediaRecorder` class is used for both audio and video capture. After constructing a `MediaRecorder` object, to capture audio, the `setAudioEncoder` and `setAudioSource` methods must be called. Additionally, two other methods are generally called before having the `MediaRecorder` prepare to record. These are `setOutputFormat` and `setOutputFile.` `setOutputFormat` allows us to choose what file format should be used for the recording and `setOutputFile` allows us to specify the file that we will record to.

3.9 Compression

The data, in this era where TB's of data are carried in pocket, piles up when its not transmitted or some huge data is captured like a video or bunch of pics. This usually causes slowdown of the device and an user can easily notice these lags. So as layman solution we compress these data as much as possible to save disk space. Android provides a algorithm called `Deflater()`. A new Deflater instance using the default compression level has to be constructed. This can be found in `java.util.zip` package.

The file will be created not as single but as multiple files of small size. The size may be around 5 Mb. These files are hashed to maintain the Integrity and later are uploaded to the cloud along with the hash file.

3.10 Sensor Details Capture

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful in monitoring the device. With acute observation one can analyse what exactly user was up to.

The Android platform supports three broad categories of sensors:

- Motion sensors
- Environmental sensors
- Position sensors

To monitor raw sensor data you need to implement two callback methods that are exposed through the `SensorEventListener` interface: `onAccuracyChanged()` and `onSensorChanged()`.

4

Related Work

The cry for forensic in mobile devices has been in out loud in recent times as people are tending to use smart phones to a greater extent which are of high end, of course. The cry has been heard and lots of research and experiments are done to pacify.

This section is all about some familiar and related Android Forensics projects.

4.1 Ethics

While there are legal requirements regarding forensic investigations, there are also ethical requirements, and these are just as important. As a forensic investigator, you need to keep two facts in mind. The first is that there are significant consequences resulting from your work. Whether you are working on a criminal case, a civil case, or simply doing administrative investigations, major decisions will be made based on your work. You have a heavy ethical obligation. It is also critical to remember that a part of your professional obligation will be testifying under oath at depositions and at trials. With any such testimony there will be opposing counsel who will be eager to bring up any issue that might impugn your character. Your ethics should be above reproach[Easttom and Murphy 2015]. The ethics of deploying this technique on known criminals is considerably discussed in Encyclopedia of Criminal Justice Ethics[Arrigo 2014]. If the forensic enabled device is used by a criminally-innocent but unauthorized individual it is surely a case of privacy issue and risk to the security of public. Tracking a GPS co-ordinates, tapping a device without the user's knowledge need court-issued warrant before proceeding further.

4.2 Companion Projects

There has been research work going on to build a customized fully fledge security enhanced Android ROM. One is Network Ombudsman[GEORGE 2015] for Android, where the work is related to network monitoring and filtering. It monitors the incoming and outgoing traffic, provides a real time process and network monitoring, provides a context based dynamic filtering rules which facilitates for incident investigation by querying the uploaded logs in the cloud server.

Another is Auditing Android System for Anomalous Behaviour[Jamsheed 2015], which is a Framework level auditing service to Android. It logs all activities of system and infers malicious activities from the log. It identifies the process or app that triggered malicious event.

The main work related to this work is Adding Proactive Forensic Support to Android[AIYYAPPAN 2015], where a client tool which runs on Android Linux Kernel layer interacts with adb tool and performs imaging, recovery and detail analysis of device data. The data is collected on the basis of forensically sound manner. All projects connect to a Cloud server and stores the logs.

4.3 Commercial products

In recent years a number of hardware/software tools have emerged to recover logical and physical evidence from mobile devices. Most tools consist of both hardware and software portions. The hardware includes a number of cables to connect the phone to the acquisition machine; the software exists to extract the evidence and, occasionally even to analyse it[Wikipedia 2015b].

Some of the tools include Cellebrite UFED, XRY, Oxygen Forensics, Mobiledit, etc. These tools are of great help during the investigating phones of newly found devices. They support various devices, applications, which comes up with very interactive GUI. These are capable of recovering the basic device information, Contacts, Call records, Organizer data, SMS, MMS, E-mails, Device dictionary words, Photos, videos, audio files ,voice records, Geo coordinates, Wi-Fi connections, Device logs, and many more. These tools extract the forensic data which is present at the time of device derivation. There may be loss of old data which are either cleared from cache or browser history or deleted and replaced with newer data on the same memory location. These tools, moreover, requires license.

4.4 Academic Research

Data retrieval plays a crucial role as one can search, diagnose potential evidence from the mobile devices. We can excavate these potential evidence to recover deleted messages, call records, Internet trace, and the picture etc. The forensic retrieval of data can be classified as being reactive and proactive forensic methodologies.

4.4.1 Adding Proactive Forensic Support to Android

A proactive method of data gathering tool[AIYYAPPAN 2015], which in the form of the application with a cloud server. It gathers all the forensically relevant data based on the priority assigned and stores in a stealth volume. Data is sent to cloud only when the device is connected to WiFi and has sufficient battery power. It uses basic commands like `adb` and `dd` to interact and Image a data from a device respectively. The data collected is hidden, encrypted and place in stealth volume. The data collected inculedes call records, SMS, Camera events, GPS location, modified files, etc.

4.4.2 Analysis of WhatsApp Forensics in Android Smart- phones

The study is focusing on conducting a forensic data analysis by extracting useful information from WhatsApp, most popular messaging application. WhatsApp uses a customized version of the open standard Extensible Messaging and Presence Protocol (XMPP). WhatsApp data is stored in the Internal Memory of the mobile phone. The database is encrypted with AES encryption algorithm 192-bit key is being used for WhatsApp Android Platform[Sahu 2014]. THe manual backup file will be saved as msgstore.db.crypt8 in /sdcard/WhatsApp/Databases folder. The problem lies in decrypting the msgstore.db.crypt8 file. WhatsApp Xtract tool decrypt and SQLite database files in an organized HTML form. Manually, if the entire WhatsApp is backed up, we can locate the key file which can be hence used to decypt using online websites like www.whatcrypt.com.

4.4.3 Android Forensics: Automated data collection and re-porting from a mobile device

DroidWatch[Grover 2013], an automated forensic tool which has been given an Android application appearance and an enterprise server to collect the data. It continuously gathers, accumulate, and transfers forensically sound data. The useful data is frequently sent to a remote Web server. The collected data is stored for a short time in a local SQLite database on the device and is designed to be non accessible to other app but DroidWatch. Periodically data transfer takes place, the local SQLite database file is sent over HTTPS to the server for processing. The various aspects of the forensics are covered including the browser history and URL. An examination for installed apps may disclose that a device has malware or other security concern.

4.4.4 ANDROPHSY – Forensic Framework for Android

ANDROPHSY[Akarawita et al. 2015] is an open source forensic tool for Android smartphones that helps digital forensic investigator throughout the life cycle of digital forensic investigation. Physical and Logical data Acquition is done with the help of `dd`, `adb`, and Scalpel tools available. Linux kernel commands like `logcat`, `demsg` and `dumpsys` for collecting logs. The collected evidence is hashed with MD5 algorithm. During the analysis of the evidence the hash is matched to ensure the integrity. It decodes the .apk files and evaluates it. Activity time line, file browser and hex view are built-in features of this framework. Finally it accumulates all the available data onto a report format for better analysis by the investigator.

This framework is able to achieve 31 features out of 33 which is believed to help in mobile forensics. There is comparison with ViaExtract CE and Oxygen Forensic Suit. The latest Oxygen Forensic Suit is much more improved and able to collect more data. Though it is very good ope source tool it lags in the performance. Also initial setup and User friendliness is little tricky.

Learning guides There are numerous books and materials available for Android forensics.

Learning Android Forensics[Tamma and Tindall 2015] which explains step-by-step directions on how to acquire and examine evidence and gain a deeper understanding of the Android forensic process. This book goes behind the scenes and shows what many of these tools are actually doing, giving much deeper knowledge of how they work. It teaches techniques and procedures for understanding data that can be carried over to analyzing almost any other application.

Android Forensics[Hoog 2011] which guides using free open source tools to show you how to forensically recover data from Android devices. It even explains how commercial tools works. It explains details of Android devices where data is stored and which are the area of the device is interesting for investigation.

5

Results and Discussion

A journey of a thousand miles begins with a single step.

Lao-tzu, The Way of Lao-tzu
Chinese philosopher

5.1 Journey so far

The forensic service gets hold of data in three different ways. One is from Android APIs, which collects data such as Phone logs, SMS logs, Camera events and GPS data. Second is from inotify tool, which collects file system events and third is from SQLite database files.

5.1.1 File system Changes detection

There are various file monitoring tools. To fulfill the our requirement, we are moving ahead with inotifywait.

5.1.1.1 `inotifwait` in Linux

With the help of inotify tool, we designed and implemented a forensic support module on a Linux machine. E.g., the following command uses the program named `inotifywait` [McGovern 2012] to monitor the subdirectory `/home/tmp` for the events of creation and modification, while logging those events to a file.

```
inotifywait -mr /root/tmp/ -e create -e modify -e close_write
```

Image 7.1 shows the output.

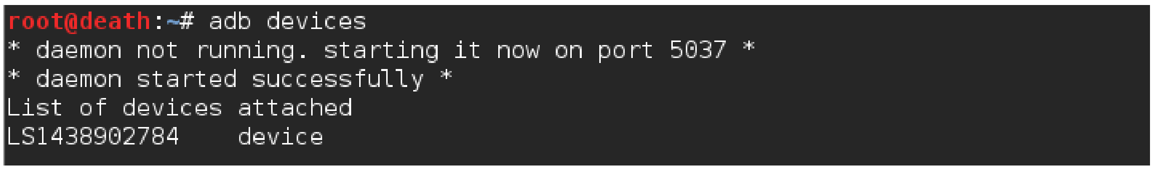
5.1.1.2 `inotifwait` in Android

File events are tracked by inotify tool and it is lightweight compared to its counterparts. The tweeked inotifywait source was compiled using NDK programming and compiled to a native shared object library (.so extension). The native function expects a directory to track and all the file events are tracked. The event constants are the same as of FileObserver

5.2 Logical extraction overview

5.2.1 Manual ADB data extractions

The ADB `pull` command can be used to pull single file or entire directories directly from the device. One should set up the ADB environment properly. USB debugging is the actual method through which the computer will communicate with the device; It must be turned on. Check whether the device connected is visible. The ADB `pull` command is used to transfer files from the device to the



```
root@death:~# adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
LS1438902784    device
```

Figure 5.1: Starting the daemon and listing the devices

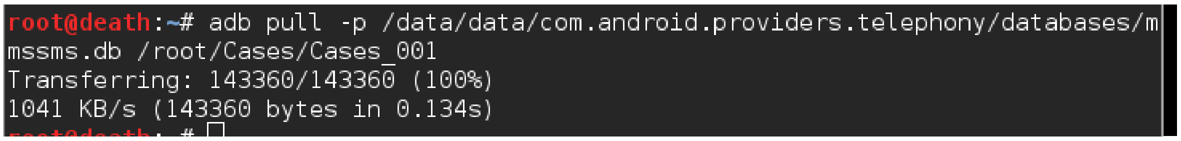
local workstation. The format for the ADB pull command is:

```
adb pull [-p] [-a] <remote> [<local>]
```

To see what an ADB pull command look like, run the following command:

```
adb pull -p /data/data/com.android.providers.telephony/databases/mmssms.db
                                                            /root/Cases/Case001
```

The device should be rooted before executing the command. Image 7.2 shows the output after



```
root@death:~# adb pull -p /data/data/com.android.providers.telephony/databases/m
mssms.db /root/Cases/Cases_001
Transferring: 143360/143360 (100%)
1041 KB/s (143360 bytes in 0.134s)
```

Figure 5.2: adb pull on a particular database

opening the DB. Similarly, to pull the files for an entire application:

```
adb pull -p /data/data/com.google.android.gm /home/Cases/Case_002
```

It is even possible to do the following:

```
adb pull -p /data/data/ /home/Cases/Case_0003
```

5.2.2 ADB Backup extractions

When a developer makes a new app, it is set to allow backups by default, but this can be changed by the developer. In practice, it seems the vast majority of developers leave the default setting, which means that backups do capture most third-party applications.[Tamma and Tindall 2015]

5.2.2.1 Extracting a backup

The format of the ADB backup command is:

```
adb backup [-f <file>] [-apk|-noapk] [-obb|-noobb] [-shared|-noshared] [-all]
                                          [-system|-nosystem] [<packages...>]
```

An example ADB backup command to capture a specific application's data is:

```
adb backup -f /home/Cases/Case_0001/facebook.ab com.facebook.katana
```

More on the Appendix B Similarly, an example ADB backup command to capture all possible application data is:

```
adb backup -f /home/Cases/Case_0001/backup.ab -shared -all
```

5.2.2.2 Parsing ADB backups

The resulting backup data is stored as a `.ab` file, but is actually a `.tar` file that has been compressed with the **Deflate** algorithm[Tamma and Tindall 2015]. Android Backup Extractor is used for this. To use the Android Backup Extractor, simply extract its files into the directory with the backup. The command to run the utility is:

```
java -jar abe.jar unpack facebook.ab facebook.tar
```

5.2.2.3 Data locations within backups

Now that the backup has been converted to a `.tar` file and then extracted. We can investigate and retrive the data like DB, XML, and more.

5.2.3 ADB dumpsys information

Dumpsys is a tool on Android OS to show the status of services running on the device. Dumpsys does not require root access, but it does require USB Debugging. To view the list of all possible services that can be dumped:

```
adb shell service list
```

A dumpsys command, using service iphonesubinfo looks like this:

```
adb shell dumpsys iphonesubinfo
```

The output will be very large for some services, like notification, and should be redirected into a text file.

```
adb shell dumpsys > dumpsys.txt
```

5.2.4 Bypass Android lock screens

Lock screens are the most challenging aspect of Android forensic examinations. There are many methods used to secure a device, and the methods for bypassing each vary:

5.2.4.1 Cracking an Android pattern lock

Only one file needs to be pulled to crack a Pattern lock on Android:

```
/data/system/gesture.key
```

Deleting this key and restarting will do the trick, but carefully analyzing with hex editor and using the script[CCLuser 2012] we can get the exact pattern.

5.2.4.2 Cracking an Android PIN/Password

The files that need to be pulled to crack a PIN/password on Android are:

- `/data/system/password.key`
- `/data/system/locksettings.db`

Editing the both the file in hex editor to get hash and salt letters only to brute force it using a tool[CCLuser 2012]

5.3 Physical extraction overview

An exact bit-for-bit image of the device is captured but different methods than above mentioned.

5.3.1 Extracting data physically with dd

The `dd` command is a Linux command-line utility used by definition to convert and copy files, but is frequently used in forensics to create bit-by-bit images of entire drives[Hoog 2011].The format of the `dd` command is as follows:

```
dd if=/dev/block/mmcblk0 of=/sdcard/data.img bs=4096 conv=notrunc,noerror,sync
```

5.4 Hide Forensic Process

The process like extracting data or sending to the cloud, this should be stealthy. Normal user can easily detect it with apps available on the Play store. Process list command `ps` uses /proc file system to get process' details. Editing `ps` binary wont be a acceptable answer to covert a processes, but hiding folder `/proc/PID/` or refusing access to it will be an fair approach. Linux dynamic linker takes care of loading the various libraries needed by a program at runtime. A solution is suggested by Gianluca Borello[Borello 2014] provides a provision to load shared library before normal system libraries are loaded. The source code overrides libc's `readdir` function, every time the code see that the `/proc/PID` just block that access.

5.5 Dissecting the Whatsapp

Whatsapp is most popular and widely used messaging application, with over 1,000,000,000 downloads and 4.4 user rating, that being said need to analyze and retrieve the data plays a vital role.

Media files are easily available even without root access. The file explorer like ES or File Manager Apks will do the job. The SD card is a treasure chest of WhatsApp data. The **Media** folder contains a folder for each type of media (Wallpaper, WhatsApp Audio, WhatsApp Calls, WhatsApp Documents, WhatsApp Images, WhatsApp Profile Photos, WhatsApp Video, and WhatsApp Voice Notes), and stores all attachments of respective type in the folder. Sent media is stored in a directory called "Sent". Received media is simply stored in the folder.

The **Profile Pictures** folder contains all the profile picture a user viewed by opening it. These contains pictures of user' contacts' images they put on display as Display Picture

The **Databases** folder is greater source of information as it contains backup databases. WhatsApp makes a backup of `msgstore.db` nightly and stores the backups here. This allows to see historical data that may have been deleted. The procedure is kind enough to put the date in the filename, for example, `msgstore-2015-12-15.1.db.crypt8`. The only problem is that these backups are encrypted!

5.5.1 Decrypting WhatsApp backups databases

There are number tools and ways to decrypt Whatsapp backup databases. The manual way I approached, not forensic perspective, is that I copied the encrypted DB to my system. Copied this `key` file to my system which can be get from

`data/data/com.whatsapp/files/key`

Then took help of online site

5.5.2 Files and Folders of the WhatsApp

The `/files/avatars` directory contains thumbnails of the profile pictures of contacts that use the app, and me.jpg is a full-size version of the user's profile picture. The me file contains the phone number associated with the account. The phone number associated with the account can also be recovered in `/shared_prefs/RegisterPhone.xml`. The `/shared_prefs/VerifySMS.xml` file shows the time that the account was verified, indicating when the user first began using the app.

The `msgstore.db` database contains messaging data. Visit Appendix D for more details.

The `wa.db` database is used to store contact information.

5.6 Analysis of Different Apps

In this section we analyze 5 basic and important applications

5.6.1 WiFi

Wi-Fi is not technically an application, but it is an invaluable source of data that should be examined.

`/data/misc/wifi/wpa_supplicant.conf`

file contains a list of access points that the user has chosen to connect automatically. If the access point requires a password, that would also be stored in the file in plain text.

5.6.2 Messaging (SMS/MMS)

SMS and MMS messages are stored in the same database.

App Info

App Name: **Messaging**

Package Name: `com.android.providers.telephony`

Files of Interest: /databases/`mmssms.db`

/databases/`telephony.db`

The telephony.db database is small, but contains one potentially useful source of information. The mmssms.db database contains all information regarding SMS and MMS messages.

5.6.3 Contacts (Phonebook/call)

Contacts and call logs are stored in the same database.

App Info

App Name: **Contacts**

Package Name: `com.android.providers.contacts`

Files of Interest: /files/photos/

/files/profile/

/databases/`contacts2.db`

The files directory contains photos for the user's contacts in the photos directory and the user's profile photo in the profile directory. The `contacts2.db` database contains all of the information about calls made to and from the device and all contacts in the user's Google account.

5.6.4 User dictionary

The user dictionary is an incredible source of data.

App Info

App Name: **User dictionary**

Package Name: `com.android.providers.userdictionary`

Files of Interest: /databases/user_dict.db

5.6.5 Gmail

Gmail is an e-mail service provided by Google.

App Info

App Name: **Gmail**

Package Name: `com.google.android.gm`

Files of Interest: /cache

/databases/`mailstore.<username>@gmail.com.db`

/databases/`suggestions.db`

/shared_prefs/`MailAppProvider.xml`

/shared_prefs/`Gmail.xml`

/shared_prefs/`UnifiedEmail.xml`

The `/cache` directory within the application folder contains recent files that were attached to e-mails, both sent and received. The `mailstore.<username>@gmail.com.db` file contains a variety of useful information. The `suggestions.db` database contains terms that were searched within the application.

6 Conclusion

Forensic analysis of smartphone is extremely useful during the run of criminal investigations. Nowadays mobile devices possess a vast amount of personal data that can help the investigators to track the activities of a suspect and present in the court of law. These data include the call details, location history, messages, browser history etc. The data in the device can be used to identify the individual. The service of data collection is done stealthily that monitors, and collects the user activities and stores it to cloud storage. Keylogger and call recording services to monitor deeply by crossing the line. The Encryption and Hashing techniques has to be improved by finding a better solution. As a future expansion extraction of data from the cloud according to the unique ID of the device (e.g. IMEI Number) has to be done. The Extractor tool also could create reports from the collected data. Data can be filtered according to the user activity such as the Phone logs, SMS or Social media messaged collected and reports can be generated. A real time monitoring without the device can also be done as a future enhancement, i.e. if the physical presence of the device is not available still the data can be collected from cloud server, which can be used to track activities of a suspect. Even though many forensic tools and techniques are available in the market, most of these tools fail to prove a user activity because of the lack of real time information collected. The deleted and encrypted data are hard to recover once the user forcefully does it. Our system collects and store all the user activities and stores stealthy to the central cloud server.

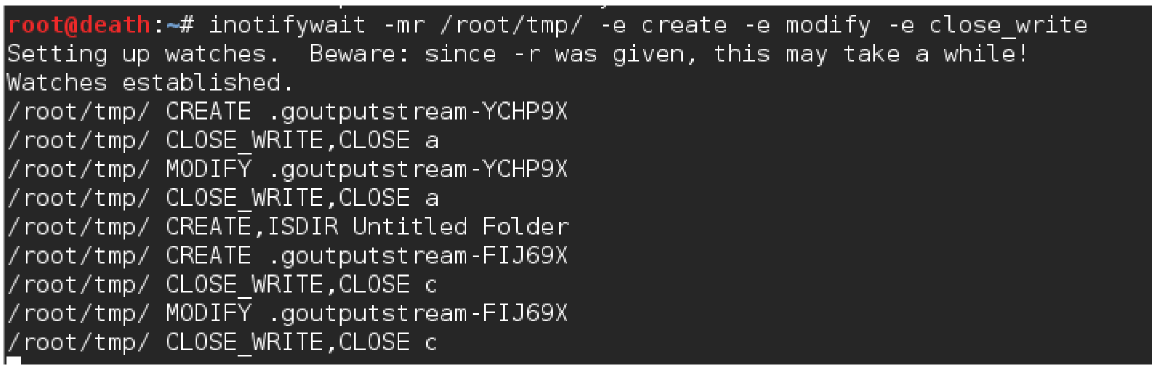
7

Appendix

7.1 Appendix A:

7.1.1 `inotifwait` in Linux

This monitors events of /root/tmp directory and its the built in for linux.



```
root@death:~# inotifywait -mr /root/tmp/ -e create -e modify -e close_write
Setting up watches.  Beware: since -r was given, this may take a while!
Watches established.
/root/tmp/ CREATE .goutputstream-YCHP9X
/root/tmp/ CLOSE_WRITE,CLOSE a
/root/tmp/ MODIFY .goutputstream-YCHP9X
/root/tmp/ CLOSE_WRITE,CLOSE a
/root/tmp/ CREATE,ISDIR Untitled Folder
/root/tmp/ CREATE .goutputstream-FIJ69X
/root/tmp/ CLOSE_WRITE,CLOSE c
/root/tmp/ MODIFY .goutputstream-FIJ69X
/root/tmp/ CLOSE_WRITE,CLOSE c
```

Figure 7.1: Starting the daemon and listing the devices

7.1.2 inotify

```
root@death:~# ./inotify1 ~/tmp /root

Press ENTER key to terminate.

Listening for events.

IN_OPEN: /root/tmp [directory]

IN_OPEN: /root/tmp/ [directory]

IN_OPEN: /root/tmp/a [file]
```

```
IN_CLOSE_NOWRITE: /root/tmp/a [file]
IN_OPEN: /root/tmp/b [file]
IN_CLOSE_NOWRITE: /root/tmp/b [file]
IN_OPEN: /root/tmp/c [file]
IN_CLOSE_NOWRITE: /root/tmp/c [file]
IN_CLOSE_NOWRITE: /root/tmp [directory]
IN_CLOSE_NOWRITE: /root/tmp/ [directory]
```

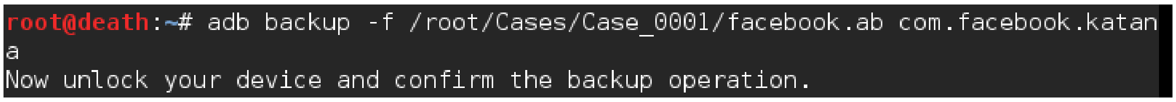
7.1.3 incron

Installation, should be very straight forward. Just use your Linux distribution package manager to install it. Basically to run `incron`

1. Create the file `/etc/incron.allow`, and add root and your user
2. Run: `incrontab -e`
3. Enter the folder to watch, the event to look for, and the command to run if the event happens.

7.2 Appendix B: ADB Backup extractions

It is used to take the backup of the Facebook app. The device asks for confirmation as shown in



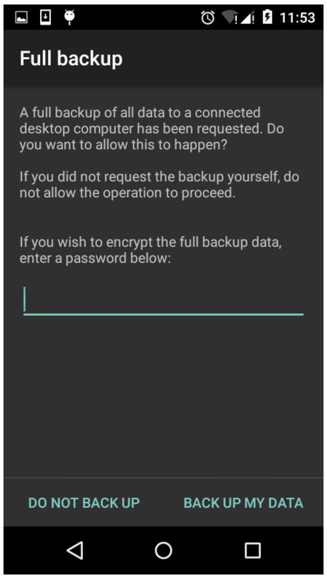
```
root@death:~# adb backup -f /root/Cases/Case_0001/facebook.ab com.facebook.katan
a
Now unlock your device and confirm the backup operation.
```

Figure 7.2: ADB backup of the facebook app

Image 7.3. Click on Back up my data to start the process. The file will be in format of .ab to convert one can use tools like Android Backup Extractor. Paste the .ab file into folder and run below command

```
java -jar abe.jar unpack facebook.ab facebook.tar
```

.tar file can be easily extracted.



11:53

Full backup

A full backup of all data to a connected desktop computer has been requested. Do you want to allow this to happen?

If you did not request the backup yourself, do not allow the operation to proceed.

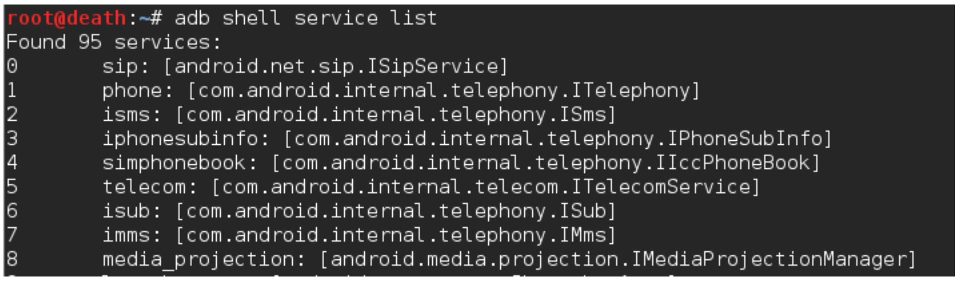
If you wish to encrypt the full backup data, enter a password below:

DO NOT BACK UP BACK UP MY DATA

Figure 7.3: Pops up confirmation before backing up

7.3 Appendix C: ADB dumpsys

The list of all services can be found as shown below



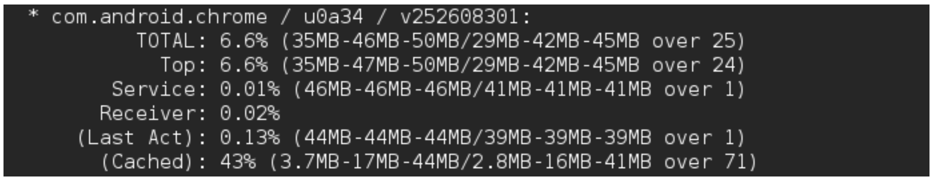
```
root@death:~# adb shell service list
Found 95 services:
0       sip: [android.net.sip.ISipService]
1       phone: [com.android.internal.telephony.ITelephony]
2       isms: [com.android.internal.telephony.ISms]
3       iphonesubinfo: [com.android.internal.telephony.IPhoneSubInfo]
4       simphonebook: [com.android.internal.telephony.IIccPhoneBook]
5       telecom: [com.android.internal.telecom.ITelecomService]
6       isub: [com.android.internal.telephony.ISub]
7       imms: [com.android.internal.telephony.IMms]
8       media_projection: [android.media.projection.IMediaProjectionManager]
```

Figure 7.4: List of different services

The procstats is a service to display the processor usage by running applications. Image 7.5

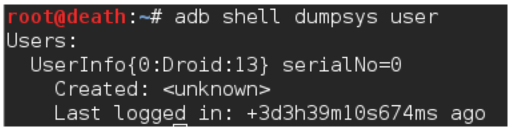
Running dumpsys on the user service will show last login info for all users. Image 7.6

The term App Ops is generally used to refer to permissions accessible by an application. Image 7.7



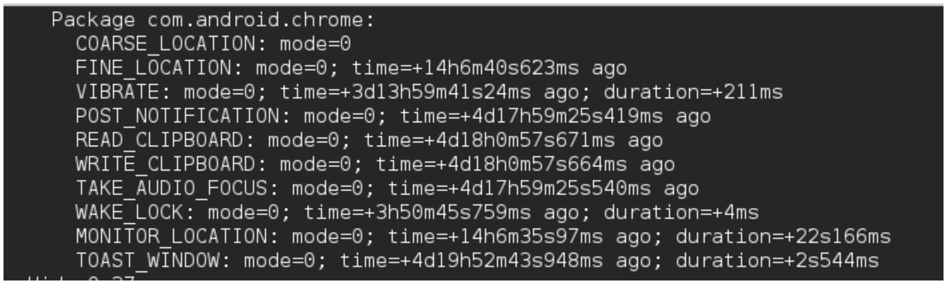
```
 * com.android.chrome / u0a34 / v252608301:
         TOTAL: 6.6% (35MB-46MB-50MB/29MB-42MB-45MB over 25)
           Top: 6.6% (35MB-47MB-50MB/29MB-42MB-45MB over 24)
       Service: 0.01% (46MB-46MB-46MB/41MB-41MB-41MB over 1)
      Receiver: 0.02%
    (Last Act): 0.13% (44MB-44MB-44MB/39MB-39MB-39MB over 1)
      (Cached): 43% (3.7MB-17MB-44MB/2.8MB-16MB-41MB over 71)
```

Figure 7.5: Dumpsys procstats



```
root@death:~# adb shell dumpsys user
Users:
  UserInfo{0:Droid:13} serialNo=0
    Created: <unknown>
    Last logged in: +3d3h39m10s674ms ago
```

Figure 7.6: Dumpsys user



```
    Package com.android.chrome:
      COARSE_LOCATION: mode=0
      FINE_LOCATION: mode=0; time=+14h6m40s623ms ago
      VIBRATE: mode=0; time=+3d13h59m41s24ms ago; duration=+211ms
      POST_NOTIFICATION: mode=0; time=+4d17h59m25s419ms ago
      READ_CLIPBOARD: mode=0; time=+4d18h0m57s671ms ago
      WRITE_CLIPBOARD: mode=0; time=+4d18h0m57s664ms ago
      TAKE_AUDIO_FOCUS: mode=0; time=+4d17h59m25s540ms ago
      WAKE_LOCK: mode=0; time=+3h50m45s759ms ago; duration=+4ms
      MONITOR_LOCATION: mode=0; time=+14h6m35s97ms ago; duration=+22s166ms
      TOAST_WINDOW: mode=0; time=+4d19h52m43s948ms ago; duration=+2s544ms
```

Figure 7.7: Dumpsys Appops

7.4 Appendix D: WhatsApp

Image 7.8 shows the entire tree structure of teh com.whatsapp folder.

The /files/avatars directory contains thumbnails of the profile pictures of contacts that use the app, and me.jpg is a full-size version of the user's profile picture. The me file contains the phone number associated with the account.

The phone number associated with the account can also be recovered in /shared_prefs/RegisterPhone.xml. The /shared_prefs/VerifySMS.xml file shows the time that the account was verified, indicating when the user first began using the app example Image 7.9. The msgstore.db database contains messaging data. Image 7.10

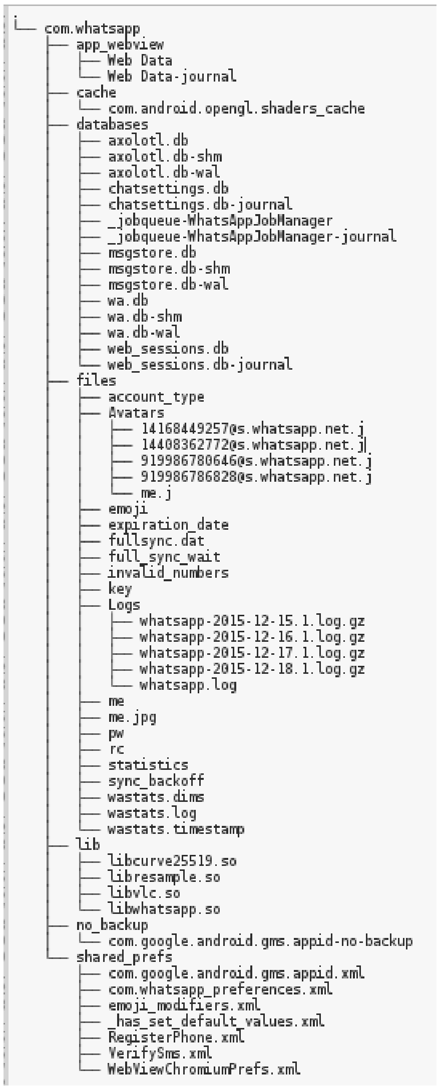
chat_list: The key_remote_jid column shows each account the user has communicated with.

group_participants: This contains metadata about group chats.

messages: This shows all the message data.

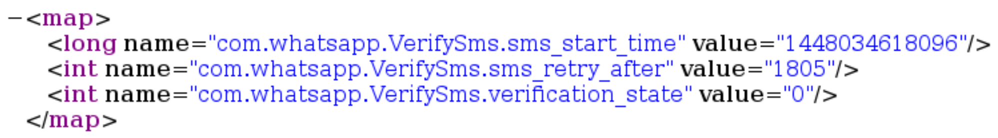
The wa.db database is used to store contact information:Image 7.11

wa_contacts: WhatsApp scrapes and stores the user's entire phone book and stores the information



```
└── com.whatsapp
    ├── app_webview
    │   ├── Web Data
    │   └── Web Data-journal
    ├── cache
    │   └── com.android.opengl.shaders_cache
    ├── databases
    │   ├── axolotl.db
    │   ├── axolotl.db-shm
    │   ├── axolotl.db-wal
    │   ├── chatsettings.db
    │   ├── chatsettings.db-journal
    │   ├── _jobqueue-WhatsAppJobManager
    │   ├── _jobqueue-WhatsAppJobManager-journal
    │   ├── msgstore.db
    │   ├── msgstore.db-shm
    │   ├── msgstore.db-wal
    │   ├── wa.db
    │   ├── wa.db-shm
    │   ├── wa.db-wal
    │   ├── web_sessions.db
    │   └── web_sessions.db-journal
    ├── files
    │   ├── account_type
    │   ├── Avatars
    │   │   ├── 14168449257@s.whatsapp.net.j
    │   │   ├── 14408362772@s.whatsapp.net.j
    │   │   ├── 919986780646@s.whatsapp.net.j
    │   │   ├── 919986786828@s.whatsapp.net.j
    │   │   └── me.j
    │   ├── emoji
    │   ├── expiration_date
    │   ├── fullsync.dat
    │   ├── full_sync_wait
    │   ├── invalid_numbers
    │   ├── key
    │   ├── Logs
    │   │   ├── whatsapp-2015-12-15.1.log.gz
    │   │   ├── whatsapp-2015-12-16.1.log.gz
    │   │   ├── whatsapp-2015-12-17.1.log.gz
    │   │   ├── whatsapp-2015-12-18.1.log.gz
    │   │   └── whatsapp.log
    │   ├── me
    │   ├── me.jpg
    │   ├── pw
    │   ├── rc
    │   ├── statistics
    │   ├── sync_backoff
    │   ├── wastats.dims
    │   ├── wastats.log
    │   └── wastats.timestamp
    ├── lib
    │   ├── libcurve25519.so
    │   ├── libresample.so
    │   ├── libvlc.so
    │   └── libwhatsapp.so
    ├── no_backup
    │   └── com.google.android.gms.appid-no-backup
    └── shared_prefs
        ├── com.google.android.gms.appid.xml
        ├── com.whatsapp_preferences.xml
        ├── emoji_modifiers.xml
        ├── _has_set_default_values.xml
        ├── RegisterPhone.xml
        ├── VerifySms.xml
        └── WebViewChromiumPrefs.xml
```

Figure 7.8: Tree structure of the WhatsApp



```
-<map>
   <long name="com.whatsapp.VerifySms.sms_start_time" value="1448034618096"/>
   <int name="com.whatsapp.VerifySms.sms_retry_after" value="1805"/>
   <int name="com.whatsapp.VerifySms.verification_state" value="0"/>
 </map>
```

Figure 7.9: VerifySms.xml showing the installed date i.e, Friday 20 November 2015 09:20:18 PM IST

in its own database.

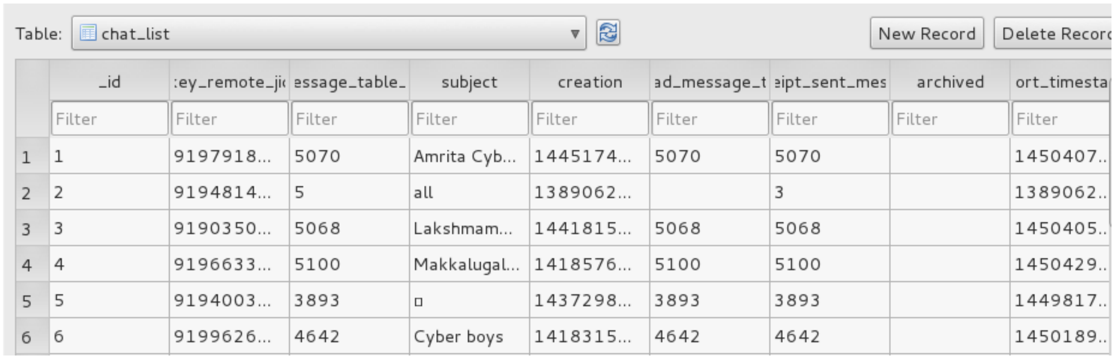


Table: chat_list

New Record | Delete Recor

	_id	:ey_remote_jic	essage_table_	subject	creation	ad_message_t	eipt_sent_mes	archived	ort_timesta
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	9197918...	5070	Amrita Cyb...	1445174...	5070	5070		1450407..
2	2	9194814...	5	all	1389062...		3		1389062..
3	3	9190350...	5068	Lakshmam...	1441815...	5068	5068		1450405..
4	4	9196633...	5100	Makkalugal...	1418576...	5100	5100		1450429..
5	5	9194003...	3893	▫	1437298...	3893	3893		1449817..
6	6	9199626...	4642	Cyber boys	1418315...	4642	4642		1450189..

Figure 7.10: chat_list table of msgstore.db

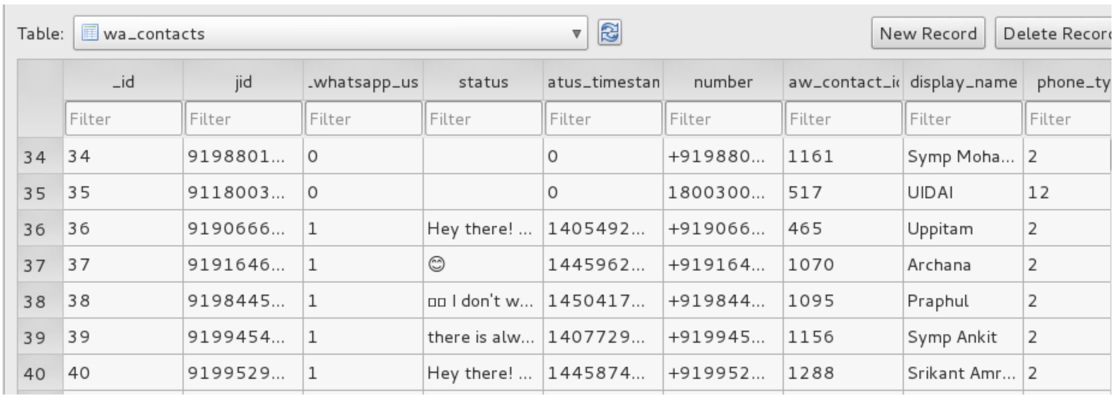


Table: wa_contacts

New Record | Delete Recor

	_id	jid	_whatsapp_us	status	atus_timestan	number	aw_contact_ic	display_name	phone_ty
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
34	34	9198801...	0		0	+919880...	1161	Symp Moha...	2
35	35	9118003...	0		0	1800300...	517	UIDAI	12
36	36	9190666...	1	Hey there! ...	1405492...	+919066...	465	Uppitam	2
37	37	9191646...	1	☺	1445962...	+919164...	1070	Archana	2
38	38	9198445...	1	▫▫ I don't w...	1450417...	+919844...	1095	Praphul	2
39	39	9199454...	1	there is alw...	1407729...	+919945...	1156	Symp Ankit	2
40	40	9199529...	1	Hey there! ...	1445874...	+919952...	1288	Srikant Amr...	2

Figure 7.11: wa_contacts table of wa.db

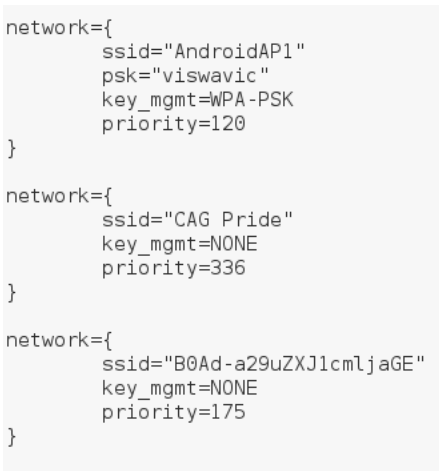
7.5 Appendix E: Analysis of Apps

7.5.1 WiFi

In the following example, Image 7.12, the ANDROIDAP1 access point required a password (viswavic), while CAG Pride did not.

7.5.2 Messaging

The mmssms.db database contains all information regarding SMS and MMS messages Image 7.13
part: This contains information about files attached to an MMS.
pdu: This contains metadata about each MMS.
sms: This contains metadata about each SMS Image 7.14
The telephony.db database is small, but contains one potentially useful source of information Image 7.15.

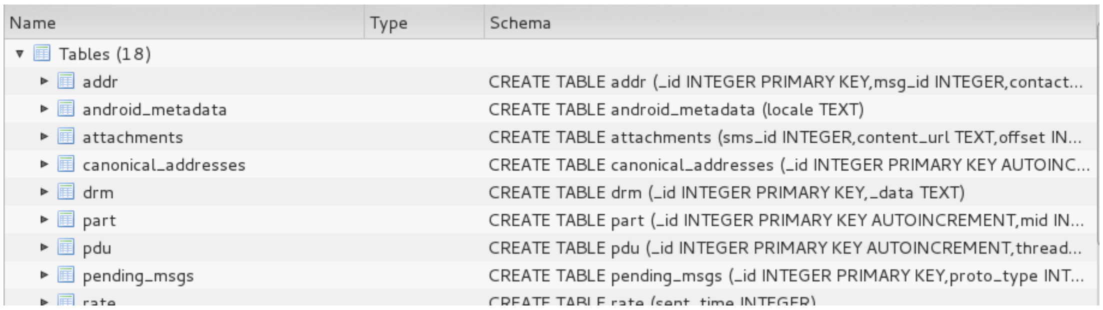


```
network={
        ssid="AndroidAP1"
        psk="viswavic"
        key_mgmt=WPA-PSK
        priority=120
}

network={
        ssid="CAG Pride"
        key_mgmt=NONE
        priority=336
}

network={
        ssid="B0Ad-a29uZXJ1cmljaGE"
        key_mgmt=NONE
        priority=175
}
```

Figure 7.12: WiFi details



Name	Type	Schema
▼ Tables (18)		
▶ addr		CREATE TABLE addr (_id INTEGER PRIMARY KEY,msg_id INTEGER,contact...
▶ android_metadata		CREATE TABLE android_metadata (locale TEXT)
▶ attachments		CREATE TABLE attachments (sms_id INTEGER,content_url TEXT,offset IN...
▶ canonical_addresses		CREATE TABLE canonical_addresses (_id INTEGER PRIMARY KEY AUTOINC...
▶ drm		CREATE TABLE drm (_id INTEGER PRIMARY KEY,_data TEXT)
▶ part		CREATE TABLE part (_id INTEGER PRIMARY KEY AUTOINCREMENT,mid IN...
▶ pdu		CREATE TABLE pdu (_id INTEGER PRIMARY KEY AUTOINCREMENT,thread...
▶ pending_msgs		CREATE TABLE pending_msgs (_id INTEGER PRIMARY KEY,proto_type INT...
▶ rate		CREATE TABLE rate (sent_time INTEGER)

Figure 7.13: list of tables in mmssms.db

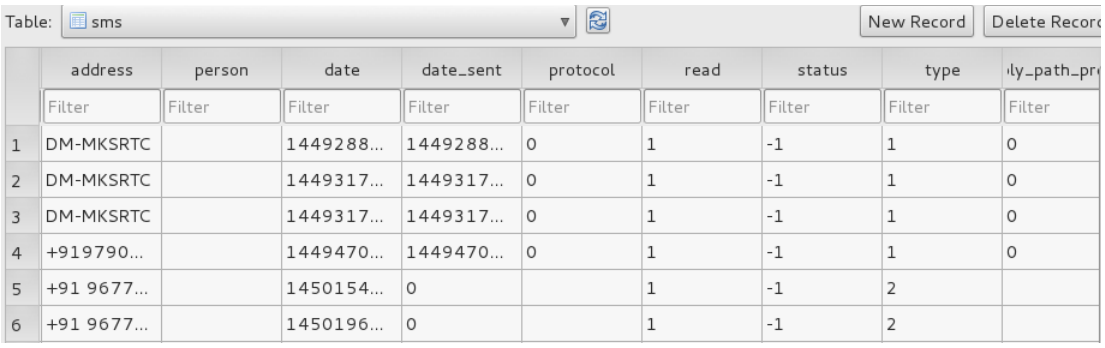


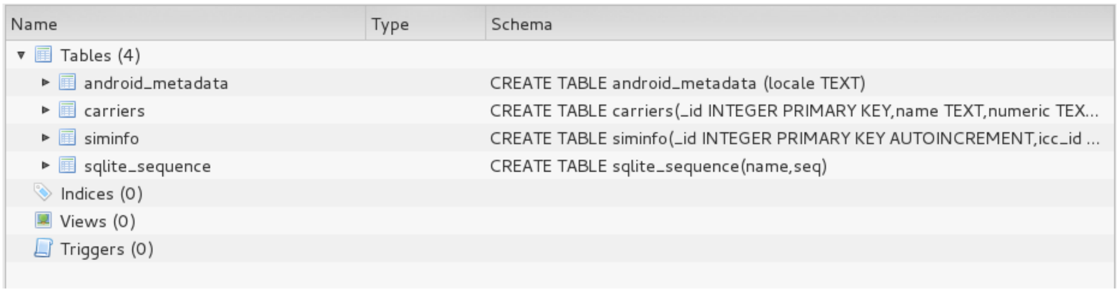
Table: sms

New Record | Delete Recor

	address	person	date	date_sent	protocol	read	status	type	ly_path_pr
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	DM-MKSRTC		1449288...	1449288...	0	1	-1	1	0
2	DM-MKSRTC		1449317...	1449317...	0	1	-1	1	0
3	DM-MKSRTC		1449317...	1449317...	0	1	-1	1	0
4	+919790...		1449470...	1449470...	0	1	-1	1	0
5	+91 9677...		1450154...	0		1	-1	2	
6	+91 9677...		1450196...	0		1	-1	2	

Figure 7.14: sms table of mmssms.db

siminfo: This contains data for all SIMs that have been used in the device, including the ICCID, phone number, and the mobile country code (MNC). Image 7.16



Name	Type	Schema
▼ Tables (4)		
▸ android_metadata		CREATE TABLE android_metadata (locale TEXT)
▸ carriers		CREATE TABLE carriers(_id INTEGER PRIMARY KEY,name TEXT,numeric TEX...
▸ siminfo		CREATE TABLE siminfo(_id INTEGER PRIMARY KEY AUTOINCREMENT,icc_id ...
▸ sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (0)		
Views (0)		
Triggers (0)		

Figure 7.15: list of table in telephony.db

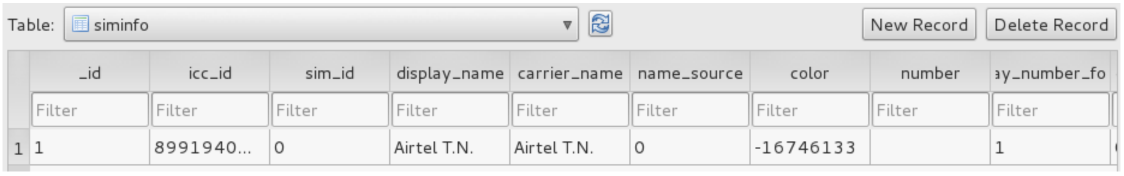


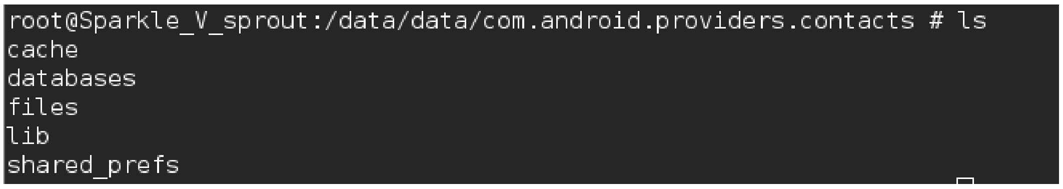
Table: siminfo

	_id	icc_id	sim_id	display_name	carrier_name	name_source	color	number	ay_number_fo
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	8991940...	0	Airtel T.N.	Airtel T.N.	0	-16746133		1

Figure 7.16: siminfo table in telephony.db

7.5.3 Contacts

List of folders inside `com.android.providers.contacts`



```
root@Sparkle_V_sprout:/data/data/com.android.providers.contacts # ls
cache
databases
files
lib
shared_prefs
```

Figure 7.17: ls of contacts

accounts: This shows the accounts on the device that have access to the contacts list.

calls: This contains information regarding all calls to and from the device Image 7.19. The number column shows the caller's phone number, whether the call was sent or received. The date column is the date/time of the call, stored in the Linux epoch format. The duration column is the length of the call, in seconds. The type column indicates the type of call:

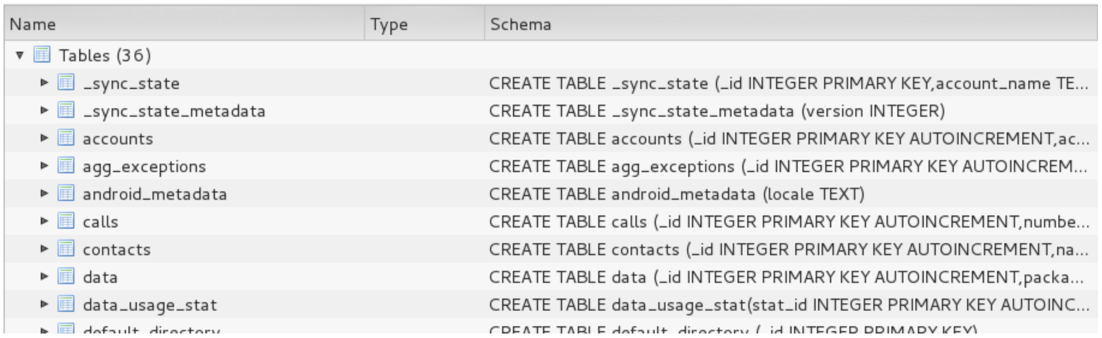
1 = incoming

2 = outgoing

3 = missed

The name column shows the remote user's name.

contacts: This contains partial information for contacts



Name	Type	Schema
▼ Tables (36)		
▸ _sync_state		CREATE TABLE _sync_state (_id INTEGER PRIMARY KEY,account_name TE...
▸ _sync_state_metadata		CREATE TABLE _sync_state_metadata (version INTEGER)
▸ accounts		CREATE TABLE accounts (_id INTEGER PRIMARY KEY AUTOINCREMENT,ac...
▸ agg_exceptions		CREATE TABLE agg_exceptions (_id INTEGER PRIMARY KEY AUTOINCREM...
▸ android_metadata		CREATE TABLE android_metadata (locale TEXT)
▸ calls		CREATE TABLE calls (_id INTEGER PRIMARY KEY AUTOINCREMENT,numbe...
▸ contacts		CREATE TABLE contacts (_id INTEGER PRIMARY KEY AUTOINCREMENT,na...
▸ data		CREATE TABLE data (_id INTEGER PRIMARY KEY AUTOINCREMENT,packa...
▸ data_usage_stat		CREATE TABLE data_usage_stat(stat_id INTEGER PRIMARY KEY AUTOINC...
▸ default_directory		CREATE TABLE default_directory (_id INTEGER PRIMARY KEY)

Figure 7.18: List of tables in contacts2.db

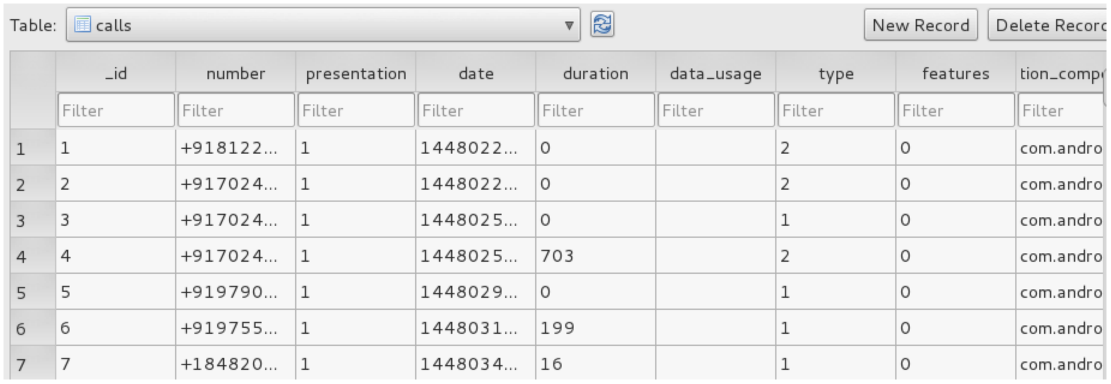


Table: calls

	_id	number	presentation	date	duration	data_usage	type	features	tion_comp
1	1	+918122...	1	1448022...	0		2	0	com.andro
2	2	+917024...	1	1448022...	0		2	0	com.andro
3	3	+917024...	1	1448025...	0		1	0	com.andro
4	4	+917024...	1	1448025...	703		2	0	com.andro
5	5	+919790...	1	1448029...	0		1	0	com.andro
6	6	+919755...	1	1448031...	199		1	0	com.andro
7	7	+184820...	1	1448034...	16		1	0	com.andro

Figure 7.19: Snippet of calls table

data: This table contains all of the information for each contact: e-mail address, phone numbers, and so on.

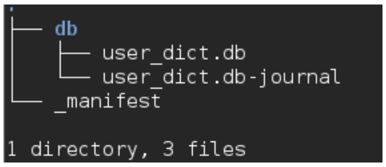
deleted_contacts: This contains a contact_id value and deleted_contact_timestamp

groups: This shows groups in the contact list.

raw_contacts: This contains all information for every contact in the contact list.

7.5.4 User dictionary

Tree structure of `com.android.providers.userdictionary`

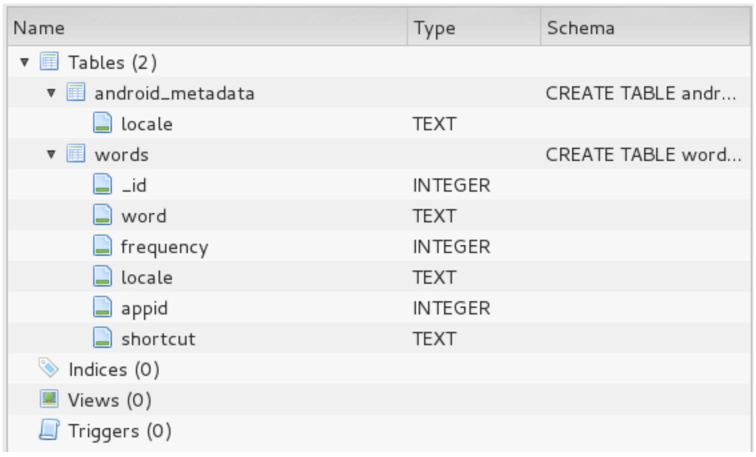


```
.
├── db
│   ├── user_dict.db
│   └── user_dict.db-journal
└── _manifest

1 directory, 3 files
```

Figure 7.20: Tree diagram of userdicionary

Image 7.21 shows the list of tables in `user_dict.db`



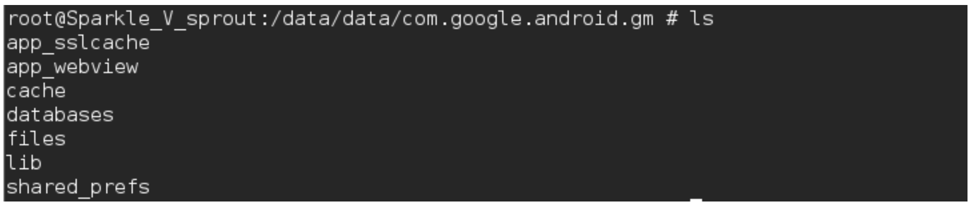
Name	Type	Schema
▾ Tables (2)		
▾ android_metadata		CREATE TABLE andr...
locale	TEXT	
▾ words		CREATE TABLE word...
_id	INTEGER	
word	TEXT	
frequency	INTEGER	
locale	TEXT	
appid	INTEGER	
shortcut	TEXT	
Indices (0)		
Views (0)		
Triggers (0)		

Figure 7.21: Tables of userdicionary

The word column contains the word that was added to the dictionary. The frequency column should likely be ignored; it displayed the same value (250) regardless of the number of times we used the word.

7.5.5 Gmail

List of directories in `com.google.android.gm`



```
root@Sparkle_V_sprout:/data/data/com.google.android.gm # ls
app_sslcache
app_webview
cache
databases
files
lib
shared_prefs
```

Figure 7.22: ls of Gmail directory

The `mailstore.<username>@gmail.com.db` file contains a variety of useful information Image 7.23 and Image 7.24 Similarly, the suggestions.db database contains terms that were searched within the application.

`Gmail.xml` contained another account that was linked with our test account example Image 7.25. `UnifiedEmail.xml` contained a partial list of senders who e-mailed the account, but with no discernible rationale as shown in Image 7.26. `Gmail.xml` also contained the last time that the application was synced in the Linux epoch format. From the screenshot we were able to deduce it to Friday 18 December 2015 10:33:24 PM IST from 1450458204466.

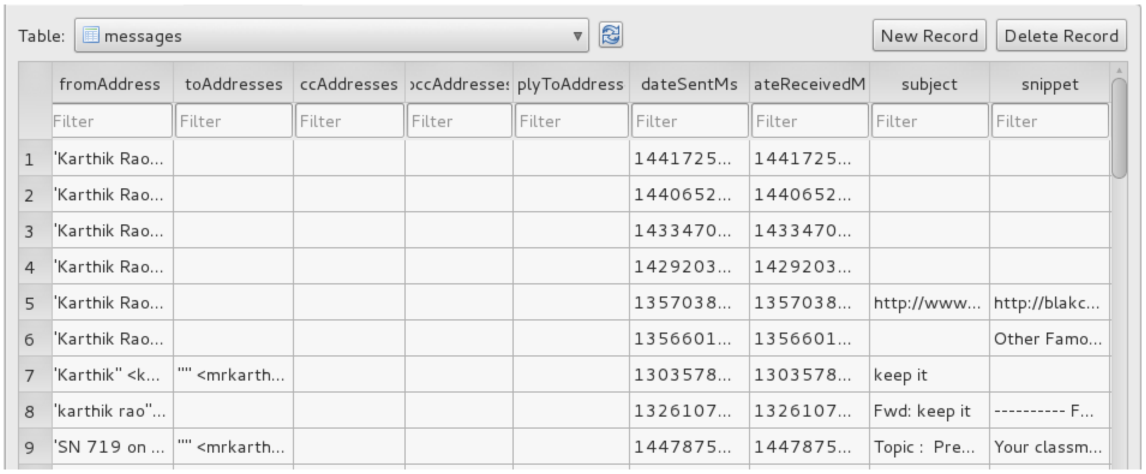


Table: messages

	fromAddress	toAddresses	ccAddresses	›ccAddresse:	plyToAddress	dateSentMs	ateReceivedM	subject	snippet
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	'Karthik Rao...					1441725...	1441725...		
2	'Karthik Rao...					1440652...	1440652...		
3	'Karthik Rao...					1433470...	1433470...		
4	'Karthik Rao...					1429203...	1429203...		
5	'Karthik Rao...					1357038...	1357038...	http://www...	http://blakc...
6	'Karthik Rao...					1356601...	1356601...		Other Famo...
7	'Karthik" <k...	"" <mrkarth...				1303578...	1303578...	keep it	
8	'karthik rao"...					1326107...	1326107...	Fwd: keep it	---------- F...
9	'SN 719 on ...	"" <mrkarth...				1447875...	1447875...	Topic : Pre...	Your classm...

Figure 7.23: Data of messages table

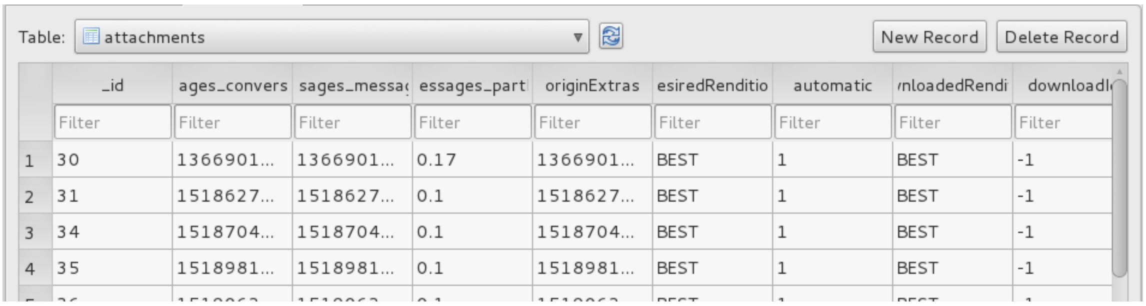
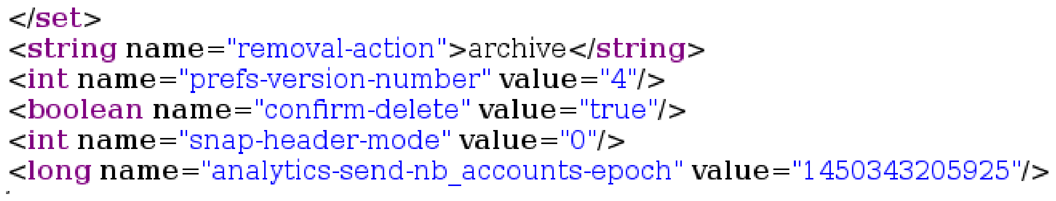


Table: attachments

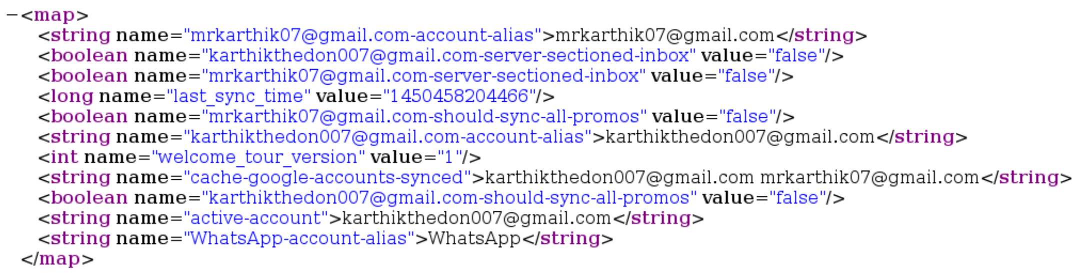
	_id	ages_convers	sages_messa	essages_part	originExtras	esiredRenditio	automatic	›nloadedRendi	downloadl
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	30	1366901...	1366901...	0.17	1366901...	BEST	1	BEST	-1
2	31	1518627...	1518627...	0.1	1518627...	BEST	1	BEST	-1
3	34	1518704...	1518704...	0.1	1518704...	BEST	1	BEST	-1
4	35	1518981...	1518981...	0.1	1518981...	BEST	1	BEST	-1

Figure 7.24: Data of attachment table



```
</set>
<string name="removal-action">archive</string>
<int name="prefs-version-number" value="4"/>
<boolean name="confirm-delete" value="true"/>
<int name="snap-header-mode" value="0"/>
<long name="analytics-send-nb_accounts-epoch" value="1450343205925"/>
```

Figure 7.25: Portion ofUnifiedEmail.xml



```
-<map>
   <string name="mrkarthik07@gmail.com-account-alias">mrkarthik07@gmail.com</string>
   <boolean name="karthikthedon007@gmail.com-server-sectioned-inbox" value="false"/>
   <boolean name="mrkarthik07@gmail.com-server-sectioned-inbox" value="false"/>
   <long name="last_sync_time" value="1450458204466"/>
   <boolean name="mrkarthik07@gmail.com-should-sync-all-promos" value="false"/>
   <string name="karthikthedon007@gmail.com-account-alias">karthikthedon007@gmail.com</string>
   <int name="welcome_tour_version" value="1"/>
   <string name="cache-google-accounts-synced">karthikthedon007@gmail.com mrkarthik07@gmail.com</string>
   <boolean name="karthikthedon007@gmail.com-should-sync-all-promos" value="false"/>
   <string name="active-account">karthikthedon007@gmail.com</string>
   <string name="WhatsApp-account-alias">WhatsApp</string>
 </map>
```

Figure 7.26: Gmail.xml

References

AIYYAPPAN. 2015. Adding proactive forensic support to android.

Akarawita, I. U., Perera, A. B., and Atukorale, A. 2015. Androphsy–forensic framework for android. In *International Conference on Advances in ICT for Emerging Regions (ICTer)*. Vol. 250. 258.

Arrigo, B. A. 2014. *Encyclopedia of Criminal Justice Ethics*. SAGE Publications.

Artenstein, N. and Revivo, I. 2014. Man in the binder: He who controls ipc, controls the droid. *BlackHat Europe*.

Borello, G. 2014. Hiding linux processes for fun and profit. `https://sysdig.com/hiding-linux-processes-for-fun-and-profit/`.

Casey, G. 2013. Inserting keylogger code in android swiftkey using apktool . `http://www.georgiecasey.com/2013/03/06/inserting-keylogger-code-in-android-swiftkey-u`

CCLuser. 2012. Android pattern lock scripts. `http://www.cclgroupltd.com/product/android-pattern-lock-scripts/`.

Easttom, C. and Murphy, G. 2015. *CCFP Certified Cyber Forensics Professional All-in-One Exam Guide*. McGraw-Hill Education.

GEORGE, N. 2015. Network ombudsman for android.

GITE, V. 2014. Linux: Hide processes from other users. `http://www.cyberciti.biz/faq/linux-hide-processes-from-other-users/`.

Grover, J. 2013. Android forensics: Automated data collection and reporting from a mobile device. *Digital Investigation 10*, S12–S20.

Hoog, A. 2011. *Android forensics: investigation, analysis and mobile security for Google Android*. Elsevier.

Inc., I. R. 2015. Smartphone os market share, 2015 q2. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

Jamsheed, K. 2015. Auditing android system for anomalous behavior.

Kaladharan, Y., Mateti, P., and Jevitha, K. 2016. An encryption technique to thwart android binder exploits. In *Intelligent Systems Technologies and Applications*. Springer, 13–21.

McGovern. 2012. Inotifywait for android. `https://github.com/mkttanabe/inotifywait-for-Android`.

Raja, H. Q. 2011. Android partitions explained: boot, system, recovery, data, cache and misc. `http://www.addictivetips.com/mobile/android-partitions-explained-boot-system-recovery-data-cache-misc/`.

Sahu, S. 2014. An analysis of whatsapp forensics in android smartphones. *International Journal of Engineering Research 3,* 5, 349–350.

Tamma, R. and Tindall, D. 2015. Learning android forensics.

Wikipedia. 2012. Android's architecture diagram. `https://en.wikipedia.org/wiki/Android_(operating_system)#/media/File:Android-System-Architecture.svg`.

Wikipedia. 2015a. inotify. `https://en.wikipedia.org/wiki/Inotify`.

Wikipedia. 2015b. Mobile device forensics. `https://en.wikipedia.org/wiki/Mobile_device_forensics`.

Yaghmour, K. 2013. *Embedded Android: Porting, Extending, and Customizing.* " O'Reilly Media, Inc.".