Proactive Forensic Support for Android Devices

DISSERTATION

Submitted by

KARTHIK M. RAO CB.EN.P2CYS14006

in partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY
IN
CYBER SECURITY

TIFAC-CORE IN CYBER SECURITY
AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

JULY 2016

Proactive Forensic Support for Android Devices

DISSERTATION

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
JULY 2016

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ENGINEERING, COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that this dissertation entitled **"Proactive Forensic Support for Android Devices"** 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
(Supervisor)

Dr. M. Sethumadhavan
(Professor and Head)

This dissertation 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 final project report entitled **"Proactive Forensic Support for Android Devices"** 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

The advanced features of smartphones catch the attention of the criminals. Information placed in a smart phone plays a crucial role in the course of a criminal investigation and while presenting it in a court of law. An ordinary man in an ordinary mobile device easily gives away his/her contact details, call history and SMS. Smartphones contain even more useful information such as social network messages, E-mails, network connections, browser history, etc. Such a device in the hands of a professional criminal is bad news, but becomes a good news to a forensic investigator. 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 designed forensic support to the Android ROM, which monitors all the user activities and stealthily stores it in the cloud. Other than SMS, contacts, call log, browser history, etc this thesis also adds a keylogger and call tapping facility. An Android app was produced which would stealthily monitor all the activities of the criminal user, record it, log it, save the records onto a local file and later upload to the cloud opportunistically.

Keywords: Android, Forensics, Digital Evidence, Android Framework, adb tool,

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I express my gratitude to my guide, Prof. Prabhaker Mateti, Associate Professor, Computer Science and Engineering, Wright State University, USA, for his valuable suggestion and timely feedback during the course of this dissertation. I express my special thanks to my colleague, Anusuya, Rahul and Sridhar, who were with me from the starting of the dissertation, for the interesting discussions and the ideas they shared. Thanks also go to my friends for sharing so many wonderful moments. They helped me not only enjoy my life, but also enjoy some of the hardness of this dissertation. In particular, I would like to thank Mr. Praveen K, Assistant Professor, Centre for Cyber Security, Amrita Vishwa Vidyapeetham, Coimbatore, for providing me with the advice, infrastructure and all the other faculties of TIFAC-CORE in Cyber Security and all other people who have helped me in many ways for the successful completion of this dissertation. And of course, thanks to my parents for helping me get where I am today.

1

Introduction

The area of mobile forensics has detonated recently and now it is one of the most promising areas of research. Because these devices are always turned on and, on the body. These can continuously record our movements and actions to provide great information about our behaviour. Interaction on a cellphone is very unlike that on a traditional computer.

Cell phone forensics was always important but never given a prominence. When these devices introduced it was only for the call or for text and this information was available for cops with help of the service providers. When Internet was added to cell phones, forensics importance to investigations went off the charts. Demand for better forensic software rose. Out of nowhere, high 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 issues with cell phones devices like encrypted mobile platforms, the Burner[PureTalkUSA 2015] 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 investigate a cell phone encounters an Android or iOS. The user synced the device to the computers in the home and/or at work, and to the cloud as well.

Right from Hello Kitty to Whatsapp to online banking transactions, from kids to youngsters to working professionals, Android provides an immense amount of apps from its chest named Play store. By 2016 about 2.2 million apps were published by [AppBrain 2016] and over 50 billion downloads by 2013 according to Mashable. A high-end Android device can do almost all computing done by a normal laptop computer and include 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

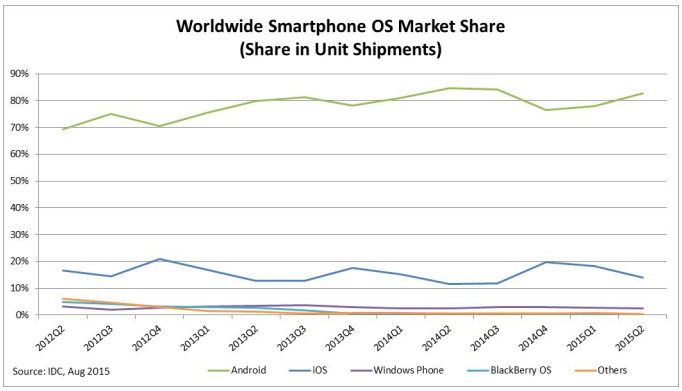

Figure 1.1: Smartphone OS Market Share, 2015 Q2 [Inc. 2015]

exploit the common man with it. This opens a wide range of path for forensics.

1.1 Motivation for Proactive Forensics

Most OSs are designed with an intention 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 new techniques and stealth services could it include so that we can 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 information about his/her whereabouts, networks 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. Here we propose a proactive forensic service that runs on Android ROM and 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.

As Android is dominating the market, it is becoming favourite target platform of hackers/criminals. There are many forensic tools available, targeted application database files to and to extract the deleted data is difficult. Usually, data which are used by messaging and mail application is stored in an encrypted format and to extract the data is harder with the usual forensic tools. A knowledgeable criminal can erase the forensic data from the mobile device. There are mobile applications like Uninstall-It[Inc. 2014] which can delete all the application data from the device and 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.

1.2 Need for Proactive Forensics

The degree of sophistication of computer crimes has advanced. An example of such sophistication is use of anti-forensics methods as in Zeus Botnet crimeware toolkit[Shah 2010] can sometimes counter-aid digital forensics investigation through its obfuscation levels. The term anti-forensics refer to methods that prevent forensic tools, investigation and investigators from achieving their goals. Two examples are data overwriting and data hiding. From digital forensics perspective anti-forensics can do following

- Prevent evidence collection
- Increase the investigation time
- Provide misleading evidence that can jeopardize the whole investigation.
- Prevent detection of digital crime[Alharbi et al. 2011]

Also there are dozens of reactive forensics tools and the limitations of existing forensic tools are

- Real-time monitoring and analysis is not possible
- Deleted and obsolete data are hard to retrieve
- Data gathered may not be sufficient to present in court of law to convict a victim
- Encrypted data are hard to recover

To investigate crimes that rely on anti-forensics methods, more digital forensics investigation techniques and tools need to be developed, tested, and automated. Such techniques and tools are called proactive forensics processes.

1.3 Aim

A proactive forensic service that runs on Android ROM such that it tracks the user activities, file system changes, app data, saves on a stealth file location and opportunistically uploads 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.

1.4 Organization

The thesis is organized into six chapters and an an 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 of 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.

1.5 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 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.

1.6 Why Android?

Research company Canalys estimated in the second quarter of 2009 that Android had a 2.8% share of worldwide smartphone shipments. By the fourth quarter of 2010, this had grown to 33% of the

market becoming the top-selling smartphone platform, overtaking Symbian. By the third quarter of 2011 Gartner estimated that more than half (52.5%) of the smartphone sales belonged to Android. By the third quarter of 2012 Android had a 75% share of the global smartphone market according to the research firm IDC[Inc. 2015]. After the 4th quarter of 2015 Android had 80.7% share[**?**].

So that concerns us why? Why Android is so popular and sweeping the market share? There are several contributing factors. The sheer variety of Android smartphones from manufacturers like Samsung, HTC, Sony, Motorola, LG, Huawei, ZTE, and others is staggering. You can get a compact phone, something with a huge touchscreen, a stylus, a rotating camera, an edge screen, or even a physical keyboard. The device starts from Rs899 of cost. Any common man can purchase these devices without selling their kidney. And if you can afford to spend a few bucks in range to Rs 15k-25k, one can get a high-end device.

The hardware is pretty robust and comes with really latest technology. Some of the devices come up with quad-core processor, 4GB RAM, 13MP camera, and so on. Android comes up with a well-equipped app store, Play Store, consisting 2.2 million apps which include wide range of genre. Other than the above, manufacturer provides features like expandable memory, removable battery, water and dust resistant, ability to record 4K HD video, IR (infrared) blaster to control devices, haptic feedback, ability to access all your files by using your phone as a mass storage device, fingerprint recognition, and more. With the huge amount of devices being purchased it becomes easy for a criminal to get access to these devices easily for usage. Thus, forensics of the Android devices is crucial in supporting the cases in the court of law.

2

Background on Forensics

This chapter describes the background of Android and forensics for our work.

2.1 Components of Android

The word *Android* refers to a robot or synthetic organism designed to look and act like a human. In the mobile domain, it denotes 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 its birthday on November 5, 2007, it was not until September 23, 2008, that version 1.0 was released as the first commercial version. The only versions 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, Marshmallow and Nougat. 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 an 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 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

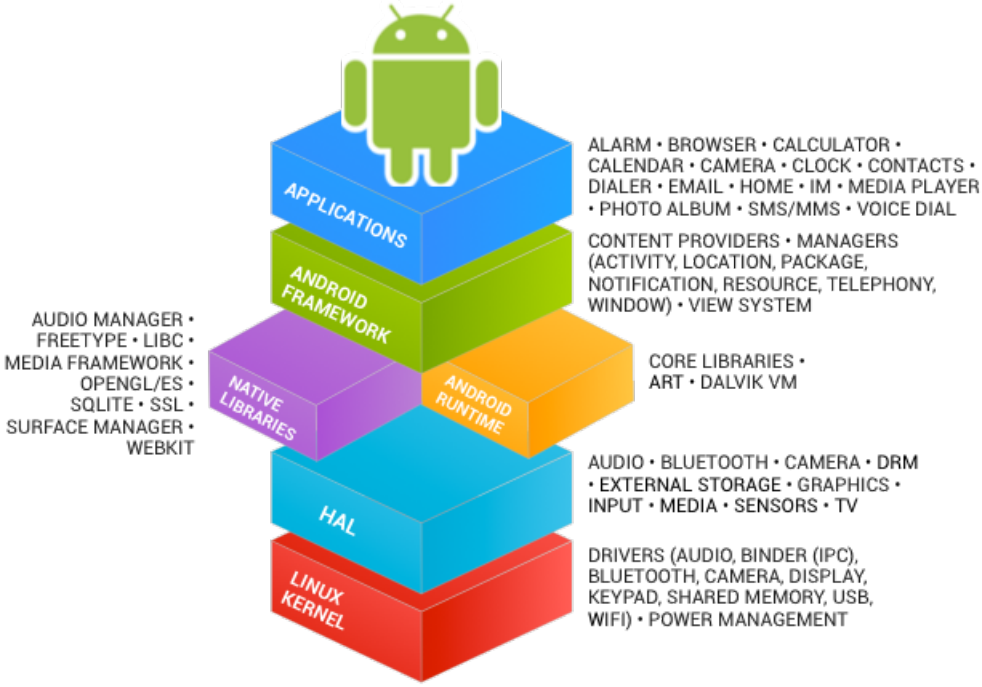

Figure 2.1: Android's architecture diagram
[Android 2015]

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.

2.2 Mobile Forensics

Mobile device forensics is a branch of digital forensics which deals with extracting, recovering and analysing digital evidence or data from a mobile device under forensically sound conditions, so that it can be presented in a court of law and to maintain the integrity of the evidence. The original device must not be tampered with. [Tamma and Tindall 2015]

There are several reasons why the need for mobile forensics is rising. Some of the prominent reasons are:

1. Mobile phones are common in several crimes
2. Mobile phones use is increasing to perform online activity
3. Mobile phones are used to store personal information

2.3 Forensic Analysis Techniques

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

2.3.1 Timeline Analysis

Timeline analysis is a key component of 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[Carrier 2012] and log2timline[Gudjonsson 2012]. The primary source of timeline information is the file system metadata including the modified, accessed, and created.

2.3.2 File System Analysis

There are a number of directories that need to be examined for an investigation. Examiners will need to expand the list to include the new directories and files, as Android devices are changing rapidly. The best approach this problem is to first run the following command to determine which file systems are mounted on the system, where they are mounted, and what type they are. Below shows the mount 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
[...]
```

2.3.3 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 it is fragmented. Newer file carving techniques are being developed to address the limitations with file fragmentation, e.g., SmartCarving[Hoog 2011]. One basic tool is the scalpel.

2.3.4 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
```

2.3.5 Hex Editor

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.
[...]
```

2.4 Challenges in Mobile Forensics

The increased usage of Android devices and the wide-ranging communication platforms that they support escalated the demand for forensic examination. Working with mobile devices is/was never a baby's play for forensic analysts. They face a number of challenges and following points cast light on some of the mobile forensics challenges faced:

1. **Data preservation on the device:** One of the elementary and vital rule to learn by heart in forensics is to not modify the evidence. The data present on the device should not be altered after forensic techniques are applied to extract information. But this is not feasible in regard to mobile forensics as simply switching ON a device may possibly change dependable state variables that are present on the device as there are background processes always running. Therefore, there is a likelihood that data may be altered either intentionally or unintentionally.

2. **Remotely wiping**: Remote data wiping would enable an attacker to remotely wipe the entire device just by sending an SMS or by simply pressing a button that sends a wipe request to the Android device.

3. **The range of operating systems and device models**: Android is the most dominant operating system in the mobile world, immediately followed by iOS, Windows, Blackberry, and so on. And for a given operating system, there are truckloads of devices available that differ in OS versions, hardware, and various other features. For example, within the Android operating

system, there are 10 running versions, and for each version, there are distinctive customizations made by different manufacturers. To survive the competition, manufacturers release new models and updates so rapidly that it's hard to keep track of all of them. Sometimes within the same operating system, the data storage options and file structures also change, making it harder. It is important for forensic analysts to be updated on all the latest changes and techniques as there is no single tool that can work on all the available types of mobile operating systems.

4. **Inherent security features**: As "privacy" is gaining importance, mobile manufacturers are implementing robust security controls on devices, which thwart gaining access to the data. For example, full disk encryption mechanisms that are implemented on some of the latest devices prevent forensic analysts from accessing the information on the device. Apple's iPhone encrypts all the data present on the device by default, using hardware keys built into the device.

5. **Legal issues**: Mobile devices can be involved in crimes that span the globe. In order to tackle these multi-jurisdictional issues, the forensic examiner needs to be aware of the nature of the crime and also regional laws. **Easy to destroy**: It's very easy to destroy device before getting caught by the criminal/lawbreaker. Though breaking the device doesn't erase all the data on it but most of it will be which is most of the time enough for delinquent to walk out of the court of law.

2.5 Android Partitions

A partition is simply a logical division of mass storage or memory into isolated subdivisions. This is normally done to help reduce the burden of the internal storage on the device. It is usually created on the SD Card in order to save more space on the internal storage. Partitioning can help to enhance the disk efficiency. Moreover, it is said that a partition can speed up the Android operating system by a huge margin.

2.5.1 Common Partitions in Android

The partition layout varies between vendors and versions. However, a few partitions are present in all the Android devices. The following sections explain some of the common partitions found in most of the Android devices.

boot loader This partition stores the phone's boot loader program. This program takes care of initializing the low level hardware when the phone boots. Thus, it is responsible for booting the

Android kernel and booting into other boot modes, such as the recovery mode, download mode, and so on.

boot As the name suggests, this partition has the information and files required for the phone to boot. It contains the kernel and RAM disk. So, without this partition, the phone cannot start its processes.

recovery Recovery partition allows the device to boot into the recovery console through which activities such as phone updates and other maintenance operations are performed. For this purpose, a minimal Android boot image is stored. This boot image serves as a failsafe.

userdata This partition is usually called the data partition and is the device's internal storage for application data. A bulk of user data is stored here, and this is where most of our forensic evidence will reside. It stores all app data and standard communications as well.

system All the major components other than kernel and RAM disk are present here. The Android system image here contains the Android framework, libraries, system binaries, and preinstalled applications. Without this partition, the device cannot boot into normal mode.

cache This partition is used to store frequently accessed data and various other files, such as recovery logs and update packages downloaded over the cellular network.

radio Devices with telephony capabilities have a baseband image stored in this partition that takes care of various telephony activities.

2.5.2 Identifying Partition Layout

For a given Android device, partition layout can be determined in a number of ways. The `partitions` file under `/proc` gives us details about all the partitions available on the device. The following snippet shows the contents of the `partitions` file:

```
shell@OnePlus2:/ $ cat /proc/partitions
major     minor  #blocks  name
 179        0   61071360 mmcblk0
 179        1      81920 mmcblk0p1
 179        2       1024 mmcblk0p2
 179        3        128 mmcblk0p3
 179        4        128 mmcblk0p4
 179        5          1 mmcblk0p5
```

```
 179        6       1024 mmcblk0p6
 179        7        128 mmcblk0p7
 179        8       1536 mmcblk0p8
 179        9          1 mmcblk0p9
 179       10       1024 mmcblk0p10
 [...]
```

The entries in the preceding snippet show only the block names. To get a mapping of these blocks to their logical functions, check the contents of the `by-name` directory present under `/dev/block/platform/dw_mmc`.

2.5.3 Android File Hierarchy

In order to perform forensic analysis on any system (desktop or mobile), it's important to understand the underlying file hierarchy. A basic understanding of how Android organizes its data in files and folders helps a forensic analyst narrow down his research to specific locations. If one are familiar with Unix-like systems, one will understand the file hierarchy in Android very well. In Linux, the file hierarchy is a single tree, with the top of the tree being denoted as `/`. This is called the **root**. Whether the filesystem is local or remote, it will be present under the root. Android file hierarchy is a customized version of this existing Linux hierarchy. Based on the device manufacturer and the underlying Linux version, the structure of this hierarchy may have a few insignificant changes. To see the complete file hierarchy, one need to have root access. The following snippet shows the file hierarchy on an Android device:

```
shell@OnePlus2:/ $ ls
acct
cache
d
data
dev
init
mnt
proc
root
sbin
sdcard
```

```
system
ueventd.rc
[...]
```

2.5.4 An Overview of Directories

The following sections provide an overview of the directories present in the file hierarchy of an Android device.

acct This is the mount point for the acct cgroup (control group) that provides for user accounting.

cache This is the directory (`/cache`) where Android stores frequently accessed data and app components. Wiping the cache doesn't affect your personal data, but simply deletes the existing data there. There is also another directory in this folder called `lost+found`. This directory holds recovered files (if any) in the event of filesystem corruption. The cache may contain forensically relevant artifacts, such as images, browsing history, and other app data.

d This is a symbolic link to `/sys/kernel/debug`. This folder is used to mount the debugfs filesystem and to debug kernel.

data This is the partition that contains the data of each application. Most of the data belonging to a user, such as the contacts, SMS, dialled numbers, and so on, is stored in this folder. This folder has significant importance from a forensic point of view as it holds valuable data. The following sections provide a brief explanation of other important subdirectories present under the `data` folder.

dalvik-cache Android applications contain `.dex` files that are optimized versions of Java bytecode. When an application is installed on an Android device, some modifications are performed on the corresponding `.dex` file, and a resultant file called `.odex` file is created. It is then cached in the `/data/dalvik-cache` directory so that it doesn't have to perform the optimization process every time it loads `application.log`.

This folder contains several logs that might be useful during examination, depending on the underlying requirements.

data The `/data/data` partition contains the private data of all the applications. Most of the data belonging to the user is stored in this folder. This folder has significant importance from a forensic point of view as it holds valuable data.

dev This directory contains special device files for all the devices. This is the mount point for the tempfs filesystem. This filesystem defines the devices available to the applications.

init When booting the Android kernel, the init program is executed. This program present under this folder.

mnt This directory serves as a mount point for all the filesystems, internal and external SD cards, and so on. The following snippet shows the mount points present in this directory:

```
shell@OnePlus2:/mnt $ ls
asec
expand
media_rw
obb
runtime
sdcard
secure
user
```

proc This is the mount point for the procfs filesystem that provides access to the kernel data structures. Several programs use `/proc` as the source for their information. It contains files that have useful information about the processes. For instance, as shown in the following snippet, `meminfo` present under `/proc` gives information about the memory allocation:

```
shell@OnePlus2:/proc $ cat meminfo
MemTotal:        3796556 kB
MemFree:          186608 kB
Buffers:           92416 kB
Cached:          1365360 kB
SwapCached:            0 kB
Active:          2145968 kB
Inactive:         762448 kB
Active(anon):    1454812 kB
Inactive(anon):    39372 kB
Active(file):     691156 kB
[...]
```

root This is the home directory for the root account. This folder can be accessed only if the device is rooted.

sbin This contains binaries for several important daemons. This is not of much significance from a forensic perspective.

misc As the name suggests, this folder contains information about miscellaneous settings. These settings mostly define the state, that is, ON/OFF. Information about hardware settings, USB settings, and so on can be accessed from this folder.

sdcard This is the partition that contains the data present on the SD card of the device. Note that this SD card can be either removable storage or non-removable storage. Any app on your phone with the `WRITE_EXTERNAL_STORAGE` permission may create files or folders in this location. There are some default folders, such as `android_secure`, `Android`, `DCIM`, `media`, and so on, present in most of the mobiles. The following snippet shows the contents of `/sdcard` location:

```
shell@OnePlus2:/sdcard $ ls
Android
ApkEditor
AzRecorderFree
CrystalExpressGlobal
Custom Keyboard
DCIM
DICTIONARY
Download
[...]
```

Digital Camera Images (DCIM) is the default directory structure for digital cameras, smartphones, tablets, and related solid-state devices. Some tablets have a `photos` folder that points to the same location. Within `DCIM`, you will find photos you have taken, videos, and thumbnails (cache) files. Photos are stored in `/DCIM/Camera`. Android developer's reference explains that there are certain public storage directories that are not specifically tied to a specific program. Here is a quick overview of these folders:

- `Music`: Media scanner classifies all media found here as user music.
- `Podcasts`: Media scanner classifies all media found here as a podcast.
- `Ringtones`: Media files present here are classified as ringtones.
- `Alarms`: Media files present here are classified as alarms.
- `Notifications`: Media files under this location are used for notification sounds.
- `Pictures`: All photos, except the ones taken with a camera, are stored in this folder.

- `Movies`: All movies, except the ones taken with a camera, are stored in this folder.
- `Download`: Miscellaneous downloads are stored in this folder.

system This directory contains libraries, system binaries, and other system-related files. The pre-installed applications that come along with the phone are also present in this partition. The following snippet shows the files present in the system partition on an Android device:

```
shell@OnePlus2:/system $ ls
app
bin
build.prop
[...]
framework
[...]
```

Here are some of the interesting files and folders present in the /system partition that are of interest to a forensic investigator.

build.prop This file contains all the build properties and settings for a given device. For a forensic analyst, this file gives an overview about the device model, manufacturer, Android version, and many other details. Contents of this file can be viewed by issuing a cat command, as shown in the following snippet:

```
shell@OnePlus2:/system $ cat build.prop
# autogenerated by oem_buildinfo.sh
ro.build.product=OnePlus2
ro.build.user=OnePlus
ro.build.flavor=OnePlus2-user
ro.build.description=OnePlus2-user 6.0.1 MMB29M 31 dev-keys
ro.common.soft=OnePlus2
ro.display.series=OnePlus2
persist.sys.oem.region=OverSeas
ro.build.oemfingerprint=6.0.1/MMB29M/1447840820:user/release-keys
ro.build.date.Ymd=160604
[...]
```

As shown in the preceding output, you can find out the product model, CPU details, and Android version by viewing this file content. On a rooted device, tweaking the `build.prop`

file could lead to a change in several system settings.

app This folder contains system apps and preinstalled apps. This is mounted as read only to prevent any changes. Along with the APK files, one might have also noticed `.odex` files in the preceding output. In Android, applications come in packages, with the `.apk` extension. These APKs contain `.odex` files whose supposed function is to save space. The `.odex` files are collection of certain parts of an application that are optimized before booting.

framework This folder contains the sources for the Android framework. In this partition, one can find the implementation of key services, such as the system server with the package and activity managers. A lot of the mapping between the Java application APIs and the native libraries is also done here.

ueventd.goldfish.rc and ueventd.rc These files contain configuration rules for the `/dev` directory.

2.6 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 assign each application 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:

```
# 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 only be read by the application and to view the files one must posses the root access. Below shows the files saved on the 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.

```
#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.

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

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

2.7 Stealth File Volumes

We live in a world where privacy is rare and our files can be easily accessed by just about anyone. Having your phone protected by a passcode is great, but if someone figures it out, they can easily

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

access anything on your phone. So we create stealth file volumes.

There are several methods to hide data partition, Linux disk encryption [Linux Kernel Organization 2016] and various file volume designs that are stealthy. That is their presence is not visible to ordinary tools such as ls, df and cat /proc/partitions. We make use of such a stealth volume to temporarily store forensic data gathered so far.

2.8 Monitoring File and Directory Changes

Linux supports many file monitoring tools like inotify, incorn, iwatch, lsyncd. Android supports notify command and FileObserver class.

2.8.1 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.

2.8.2 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.

2.8.3 Other file monitoring tools

There are many tools available in Linux for file monitoring likeFile 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. However 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 in a similar way as the regular cron. The difference is that the inotify cron handles filesystem events rather than time periods.

2.9 Sandbox

Android security relies heavily on security restrictions at the level of the Linux operating system, specifically on process and user-level boundaries. Since Android is designed for personal devices, Android makes an interesting use Linux’s inherent multi-user support[Mednieks et al. 2012]. This means each application runs with a different user privileges. Files owned by one application are, by default, inaccessible by other applications. Android allows a single logged-in phone user to see multiple applications that are running as different Linux-level users. The net effect is increased security as a result of each application running in its own “silo”. Android’s designers projected a world of plentiful small applications from many vendors who cannot all be examined for trustworthiness. Hence applications don’t have direct access to other applications’ data.

2.10 Sensor Data as Evidence

The greater part of smartphones includes multimedia hardware such as a microphone and cameras other hardware like

a. location sensor b. motion sensors and c. environmental sensors

Researchers have already used smartphone sensors in order to get hold of context awareness either for research reasons ([Mylonas et al. 2012][Mylonas 2008]), or for commercial ones (FlexiSPY2[1]). Sensors also provide data that can be collected in relation to criminal investigations. This can be

[1]FlexiSPY, http://www. exispy.com/

used in a forensic investigation to construct a suspect's activities, which aid the creation of a theory, or dismissal or acceptance of an alibi, in accordance with legal conviction.

The majority of sensor data cannot be gathered during a post-mortem investigation. Sensor data are volatile and are time-sensitive. Thus, time-stamped evidence of sensor data can scarcely be found on the device after a crime's occurrence, so they have to be explicitly collected. Currently, only GPS data are collectable in post-mortem forensics. Those are treated as metadata, which is either hidden by the operating system or implanted in other files or by other applications.

One may infer that investigators can collect data from a service/network provider, without accessing the suspect's device. For example, device location can be approximated from data provided by the carrier's network and device speed can be calculated approximately from the distance between the cell towers that the device has been to. However, the accuracy of this approximation can't stand the test with the data that are obtained from the device's dedicated hardware, which is more accurate. These dedicated sensors provide matchless measurements from:

1. a single hardware and
2. measurements deriving from two or more sensor hardware combinations

In a digital investigation, sensor hardware can provide direct evidence, such as location evidence from a GPS. They also provide indirect evidence, i.e., evidence that is deduced from the acquired data values. For instance, an investigator may use the light sensor to infer whether a suspect is indoors or outdoors. A few sensor data are easily graspable whereas the values from other sensors are not so. It is taken into consideration that a forensic investigator can benefit from these data to infer other activities, such as walking, running, etc. of a criminal/law breaker.

The combination of sensor data provides stronger indications about the smartphone's context. For example, GPS, accelerometer and magnetometer combined data provide a smartphone's navigation. Also, if we assume device owner keeps the phone in his/her pocket and based on height one can map the entire whereabouts. Even in the case of murder one can map the devices location near to the victim and make an alibi.

2.11 Integrity of Forensically Sound Data

One of the checklist in forensics is maintaining forensic soundness in documentation. Documenting how the chain of custody on the device is done from the beginning is very important. The acquisition process should conserve the original data and its authenticity and integrity can be justified at any given point of the investigation and after that. The slightest possibility of compromise to the evidence

can cast reasonable doubt as to its authenticity, so there should be proof that the evidence was not manipulated or damaged in any way from the point of collection to presentation in court or wherever applicable. Evidence integrity guarantees that the evidence has not been tampered with from the time it was collected. Integrity checks are done by comparing the digital fingerprint of the evidence taken at the time of collection with the digital fingerprint of the evidence in the current state.

Cryptographic hash functions and digital signatures can be used to prove the integrity of digital evidence, and also time stamps. Hashing algorithms are primarily used for the verification of the integrity at the time of use of the evidence. The hashing algorithm is applied to a particular digital evidence to produce a hash value. The hash value acts as a fingerprint of the evidence and is initially calculated at the time of creation of it or at other times when the information in the evidence is known to be complete and accurate. Whenever the information in the evidence is about to be used, the calculation is performed again and compared to the original hash value. If the two values are identical, it is concluded that the message says exactly what it did at the time it was known to be good.

2.12 Encryption

Android being a most popular operating system has among the finest suite of cipher libraries. One of the great things about Android using Java as the core programming language is that it includes the Java Cryptographic Extensions (JCE). JCE is a well-established, tested set of security APIs. Android uses Bouncy Castle as the open source implementation of those APIs. However, the Bouncy Castle version varies between Android versions; and only the newer versions of Android get the latest fixes. However, Spongy Castle library, which provides a higher level of security as it is more up-to-date and supports more cryptographic options can also be used.

A symmetric key encryption is described as a single key that is used for both encryption and decryption. To create cryptographically secure encryption keys in general, we use securely generated pseudorandom numbers. AES is the preferred encryption standard to DES and typically used with key sizes 128 bit and 256 bit.

2.13 Evidence Transport

Smartphones can use four data transport channels (or interfaces) that provide different transport services[Mylonas et al. 2012]. This section discusses their ability to support evidence transfer during a proactive forensic investigation.

1. **GSM Messaging interface** (e.g. SMS, etc.) provides a remote channel appropriate for small volume data transfers, which is nearly always available. Apart from the restriction in volume, another refers to cost. Increased cost
   (a) may limit the messaging service availability, thus, large data may not be transferable and
   (b) may alert suspects, who thoroughly check their carrier bills in the case a proactive investigation taking place.
2. **Personal Area Network (PAN) interface** (e.g. Bluetooth, IrDA, etc.) provides a cost free, ad hoc, remote data channel, appropriate if the data collector is in the smart-phone's proximity. By not relying on any base station existence, it avoids network monitoring mechanisms, such as Intrusion Detection System (IDS). Furthermore, as this channel requires no cost, it is stealthier than others. Potential shortcomings are:
   (a) distance constraint between the device and the collector, which increases attack complexity,
   (b) the average transfer speed and
   (c) the requirement for a pairing bypass without alerting the smartphone user.
3. WLAN interface (e.g. Wi-Fi) provides a fast, remote channel that is appropriate for any data volume often without a cost. Due to the general popularity of Wi-Fi, availability is considered high. A shortcoming is that data transfer speed and availability rely on the distance from a base station. This distance, though, is considerably larger than the PAN's requirement, thus, it does not add considerable complexity on evidence collection.
4. **Cellular network** interface provides a data transport channel of variable speed, which is dependent on the supported carrier network technology (e.g. GPRS, HSDPA, etc). This channel is not restrained by the antenna range. It provides greater mobility than any of the aforementioned channels, since the smartphone may travel through cells. Nonetheless, this channel is not considered suitable for large data volume, as:
   (a) it suffers from connection drops,
   (b) the network speed may vary as the user moves inside a cell or visits others, and
   (c) this channel use has considerable cost, and thus it may be discovered by the owner.

Table 2.2 summarizes each channel's ability to effectively transfer each source data volume. For the association notation three symbols were used, i.e.: 1. (✗) transfers small subset of the data source, 2. ( ) transfers most data in this data source and 3. (✓) can transfer source type. Obviously, the WLAN channel is able to transfer all data sources.

Data type	GSMMe	PAN	WLAN	CN
Messaging Data	✓	✓	✓	✓
Device data	✗	∼	✓	∼
(U)SIM Card Data	✓	✓	✓	✓
Application Data	✗	✓	✓	∼
Usage History Data	✗	✓	✓	∼
Sensor Data	✗	✓	✓	∼
User Input Data	✗	✓	✓	✓

Table 2.2: Correlation between transport channels and data sources [Mylonas et al. 2012]

2.14 Proactive Forensics

Proactive forensics is the practice of looking for something in advance based on high level futuristic rules. Rather than responding to a situation, proactive forensics can be used as an early warning system by using key characteristics to identify certain behavioural changes in applications, detect anomalies in network traffic or unexpected alterations to system configurations or abnormalities in the user activities. It requires a very high level view of everything that's going on across the entire device. However, to be truly effective it must also be capable of issuing timely alerts when something erroneous occurs.

2.15 Cloud

Cloud is a set of physical resources of a data centre that are designed for that role. In other words, Cloud represents a number of hosts, network devices and accessories that are treated as a single resource called Cloud. This also means that, when a virtual environment is created, resources that are set aside for Cloud are treated as a single item, i.e. as an available resource. In this way, control and utilization of resources within data centre are simplified.

Exploring the multiple variations of cloud storage as it has evolved over the last several years may be useful for agencies to assure the security of their digital evidence. Cloud storage offers efficiency, agility and innovation that typically cannot be achieved with in house or traditional on site servers. Storage of law enforcement digital data which may include documents, images and critical evidence files requires more storage space which can be accomplished with online cloud storage. Private Cloud offers the ability to virtualize an environment using software and hardware. In other words, it offers

a way to take an existing physical environment and transfer it to a virtual one.

With Cloud Storage services it is easier for anyone to access our data and share it with other people. We can access the files by using a computer, a smartphone or a tablet device. We can choose between free and commercial solutions. We obviously have a great problem with the security of the files stored on Cloud Storage servers. The most popular Cloud Storage services on personal computers and mobile devices are Dropbox, Google Drive, OneDrive, iCloud, Mega, etc.

3

Proactive Forensics System

We propose 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.

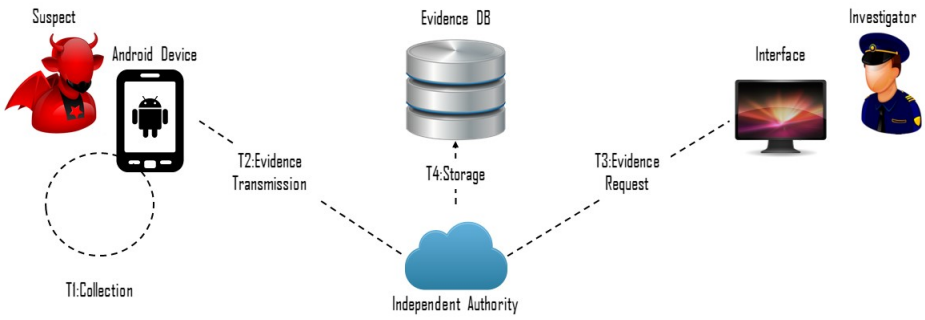

Figure 3.1: Proactive Forensic Support Architecture

There are four phases. (i) Data collection, (ii) Transmission of the Evidence, (iii) Evidence request for investigation, and (iv) Evidence storage

3.1 Architecture of Forensics Module

This section is an overview of the architecture (Figure 3.1). The next section describes a design and an initial implementation. The forensic module we proposed gets data mainly from two sources (Figure 3.2). The Primary source contains database files, API calls and the file monitoring tool. Secondary source deals with keyloggers, call recording and Vital sensor details. This data gets stored in a separate file partition named `/forensic`, and this data resides in the device until gets the chance

to upload it to the cloud server.

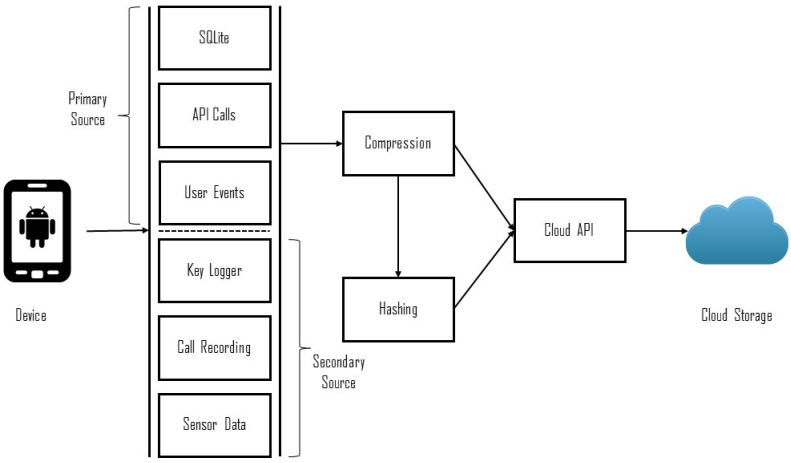

Figure 3.2: Detail architecture of proactive forensic support

3.1.0.1 Primary Sources

Android applications store their private data using SQLite databases and these files are the main source of information. For example, phone logs are stored in the **contacts2.db** file. We collect these SQLite files in a scheduled manner so that no data is forgotten when investigating. User activities like camera, network usage, and GPS locations are also collected by using Android APIs Table 3.1. Filesystem events get captured by using a tweaked version of `inotify`[Wikipedia 2015a] 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: Selected Android APIs

3.1.1 Secondary Sources

These include keyloggers which collect the key strokes on the device. Call recording/tapping records the conversation between two or more people. The sensor data provides useful information like the motion, environmental, and position of the device and its surroundings.

3.1.2 Temporary Storage

The collected data is stored on a newly created device partition named `/forensic` until it gets uploaded to the cloud. 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 Evidence Transmission

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.

The data stored in the device accumulate a lot of space which eventually hinders the performance which causes user to be suspicious. So these data are compressed into smaller units and then uploaded. To verify these transfers are bound to integrity we are hashing these units. Both are explained in-details in upcoming sections.

3.1.4 Real Time Analysis

Most of the Android applications encrypt their data and databases. Forensic data can be collected only by analysing or decrypting those application data. We do not propose real time analysis be 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.

The Evidence present in the cloud can be requested by the investigator, on a PC, to investigate. This can be easily downloaded or copied to the system for further analysis. The files will be smaller units of compressed file and along with a hash value of it. Investigator before proceeding can match the value for integrity.

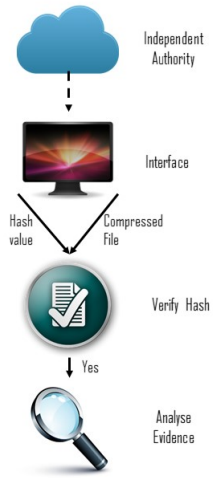

Figure 3.3: Architecture of Proactive Forensic Support

3.1.5 Data Storage Phase

The data present in the cloud should be transferred to the separate DB called as Evidence Database to safe guard evidence. Here we are not working on this piece of architecture. This has been written for completion purpose and may be for the future work.

3.2 Hiding Techniques

There are several process listing commands and APKs to list running processes in Android. These tools use 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 invisible to other users. It doesn't signify 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].

3.3 Keystroke Logging

Keystroke logging is the action of recording (logging) the keys struck on a keyboard without making the person using the keyboard aware that their actions are being monitored.

3.3.1 Using Swiftkey

SwiftKey Keyboard is among the top paid app in the Play store and it's a great keyboard app. Malware-ridden Android app is taking a serious risk of a keylogger being inserted and people tracking all their passwords, Google searches and Credit Card numbers. Basic knowledge of Java is sufficient to create app and make it as default keyboard.

We can inject a Keylogger snippets of code into a legitimate Android Keyboard application that infects a mobile device with Trojan, which reads all the keystroke typed on the device. It writes output to a text file which will be saved on the stealth file system inducing all the key logs.

3.3.2 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 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.

It has not been decide which approach to use. Completeness of work on both is still under progress.

3.4 Call Recording

Telephone tapping is the monitoring of telephone and Internet conversations by a third party, often by covert means. The recording 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.

3.5 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 a user can easily notice the lag. So as a simple solution we compress this data as much as possible to save disk space. Android provides an algorithm called `Deflater()`. A new Deflater instance using the compression level will be constructed. This can be found in `java.util.zip` package.

The zip 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.

3.6 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()`.

3.7 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. These hashes can be calculated based on the functionality available in Java.

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

3.8 Uploading to the Cloud

Storage and maintenance of digital evidence has never been more important than it presently is in the law enforcement community. Video, photo, voice recordings and documents are all critical evidence pieces which, with the assistance of advanced technology, can store and maintain the security of data at a level which most agencies can never hope to establish on their own.

All the evidence collected from the device will be uploaded to cloud for further investigation. The data presented in the \forensic will be uploaded to the cloud whenever Wi-Fi connection is present. These evidence comprises of user personal data, network data and phone specific data.

3.9 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[Kamardeen 2015] project logs all non-network events. Both upload to cloud storage.

3.10 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 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.

4

Implementation and Results

Our 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 the inotify tool, which collects file system events. Third is from SQLite database files.

4.1 File System Changes Detection

There are various file monitoring tools. To fulfill our requirement, we used inotifywait.

4.1.1 `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
```

FigureA.1 shows the output.

4.1.2 `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

4.1.3 FileObserver

Android FileObserver is based on FileObserver. It provides similar monitoring mechanism as inotify does. One thing worth-mentioning is that the API documentation says "If a directory is monitored, events will be triggered for all files and subdirectories (recursively) inside the monitored directory", but the FileObserver API is actually not recursive.

In other words, a FileObserver is attached to a folder, then only the files and sub-folders inside it are monitored, but its sub sub-folders and files are not. For example, there is a folder B inside folder A, and a folder C inside folder B. If a FileObserver is created to monitor folder A, then any modifications done for folder B is detected, but not for folder C.

Anyway, then we looked in more and found here it's been reported to Google[Evgeni 2012] and they have fixed it in 2.3 or 3.0 it seems. We tried to integrate the same code into our code and it failed because it's using Native functions. Then we found this code on here with few compile errors, then we fixed it.

As FileObserver is an abstract class, one must create a subclass that extends FileObserver and implement the event handler onEvent(int, string). Below is an example,

```
public class MyFileObserver extends FileObserver {
    public String absolutePath;
    public MyFileObserver(String path) {
        super(path, FileObserver.ALL_EVENTS);
        absolutePath = path;
    }
    @Override
    public void onEvent(int event, String path) {
        if (path == null) {
            return;
        }
        if ((FileObserver.CREATE & event)!=0) {
            FileAccessLogStatic.accessLogMsg+= absolutePath+"/"+path+"is created";
        }
        if ((FileObserver.OPEN & event)!=0) {
            FileAccessLogStatic.accessLogMsg += path + " is opened";
        }
        if ((FileObserver.ACCESS & event)!=0) {
            FileAccessLogStatic.accessLogMsg+=absolutePath+"/"+path+"is accessed/read";
```

```
        }
        if ((FileObserver.MODIFY & event)!=0) {
            FileAccessLogStatic.accessLogMsg+=absolutePath+"/"+path+"is modified";
        }
        if ((FileObserver.CLOSE_NOWRITE & event)!=0) {
            FileAccessLogStatic.accessLogMsg += path + " is closed";
        }
        if ((FileObserver.CLOSE_WRITE & event)!=0) {
            FileAccessLogStatic.accessLogMsg+=absolutePath+"/"+path+"written & closed";
        }
        if ((FileObserver.DELETE & event)!=0) {
            FileAccessLogStatic.accessLogMsg+=absolutePath+"/"+path+"is deleted";
        }
        [...]
    }
}
```

Once the subclass is defined, one needs to create instances of this class in order to use it.

```
MyFileObserver fileOb = new MyFileObserver("/sdcard/testf/");
```

Similar to `inotify_add_watch` and `inotify_remove_watch`, one can call startWatching() and stopWatching() methods to start or stop monitoring.

```
fileOb.startWatching();
fileOb.stopWatching();
```

4.2 Tracking User Activities

The user events such as GPS Location change, SMS and Call events are tracked by using broadcast receiver and all those data are stored to a log file in the respective folder i.e, “Forensic”. The events like browser data, calendar data, Dictionary words are obtained by content observers.

4.2.1 GPS and Network Location

An application requires the following to access the location services from the Android system:

- `LocationManager`-Class providing access to Android system location services

- `LocationListener`-Interface for receiving notifications from the `LocationManager` when the location has changed
- `Location`-Class representing a geographic location determined at a particular time

The `LocationManager` class needs to be initialized with the Android system service called `LOCATION_SERVICE`. This provides the application with the device's current location and movement and can also alert when the device enters or leaves a defined area. An example of initialization follows:

```
LocationManager mLocationManager;
mLocationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

After the `LocationManager` instance is initiated, a location provider needs to be selected. Different location technologies might be available on the device, and a general way to find a proper location provider is to define the accuracy and power requirements. This can be done using the `Criteria` class defined in `android.location.Criteria`. This enables the Android system to find the best available location technology for the specified requirements. Following is an example of selecting a location provider based on criteria

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
String locationprovider = mLocationManager.getBestProvider(criteria, true);
```

It is also possible to specify the location estimation technology of the device using the location manager's `getProvider()` method. The two most common providers are the satellite based GPS (specified by `LocationManager.GPS_PROVIDER`) and cell-tower/Wi-Fi identification (specified by `LocationManager.NETWORK_PROVIDER`). The former is more accurate, but the latter is useful when a direct view of the sky is not available, such as indoors.

Permission to use location information needs to be granted in the `AndroidManifest.xml` file. For a more accurate location, such as GPS, add the `ACCESS_FINE_LOCATION` permission. Otherwise, add the `ACCESS_COARSE_LOCATION` permission. It should be noted that `ACCESS_FINE_LOCATION` also enables the same sensors that are used for `ACCESS_COARSE_LOCATION`.

Because it might take time to produce a location estimation, `getLastKnownLocation()` can be called to retrieve the location last saved for a given provider. The location contains a latitude, longitude, and Coordinated Universal Time (CUT) timestamp. Depending on the provider, information on altitude, speed, and bearing might also be included (use `getAltitude()`, `getSpeed()`,

and `getBearing()` on the location object to retrieve these and `getExtras()` to retrieve satellite information). Latitude and longitude are displayed to the screen in this recipe. Another option that may be used is `PASSIVE_PROVIDER`, which is a constant that is a special location provider that stores the last request for location.

The `LocationListener` interface is used to receive notifications when the location has changed. The location manager's `requestLocationUpdates()` method needs to be called after a location provider is initialized to specify when the current activity is to be notified of changes. It depends on the following parameters:

- `provider`-The location provider the application uses
- `minTime`-The minimum time between updates in milliseconds
- `minDistance`-The minimum distance change before updates in meters
- `listener`-The location listener that should receive the updates

The `Geocoder` class provides a method to translate from an address into latitude longitude coordinates (geocoding) and from latitude-longitude coordinates into an address (reverse geocoding). Reverse geocoding might produce only a partial address, such as city and postal code, depending on the level of detail available to the location provider.

Google Maps can be used on the Android system in two ways: user access through a browser, and application access through the Google Maps API. The `MapView` class is a wrapper around the Google Maps API. To use `MapView` and version 2 of Google Maps.

Adding Google Maps to an Application To display a Google map, the main activity should extend `MapActivity`. It also must point to the layout ID for the map in the main layout XML file. Note that the `isRouteDisplayed()` method needs to be implemented, too. The `ItemizedOverlay` class provides a way to draw markers and overlays on top of a `MapView`. It manages a set of `OverlayItem` elements, such as an image, in a list and handles the drawing, placement, click handling, focus control, and layout optimization for each element. Created a class that extends `ItemizedOverlay` and override the following:

- `addOverlay()`-Adds an OverlayItem to the ArrayList. This calls populate(), which reads the item and prepares it to be drawn
- `createItem()`-Called by populate() to retrieve the given OverlayItem.
- `size()`-Returns the number of OverlayItem elements in the ArrayList.

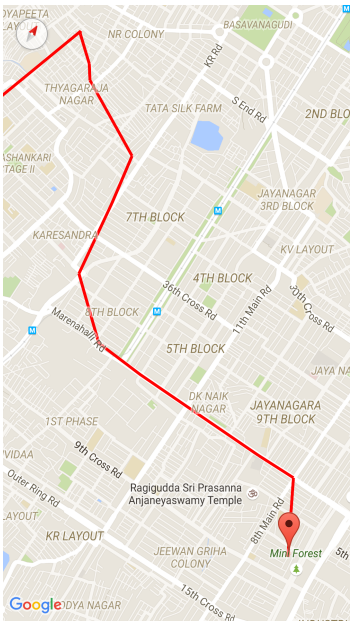

Figure 4.1: GPS track line

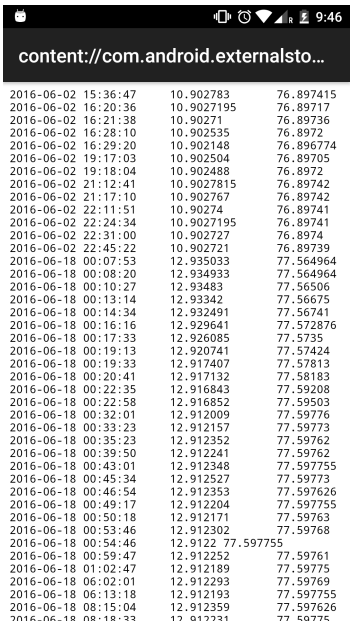

Figure 4.2: Screenshot of GPS data

- `onTap(`-Callback method when a marker is clicked.

are screenshots of our APK Figure 4.1 and Figure 4.2.

4.2.2 Sensor Data

Sensors that detect physical and environmental properties offer an exciting avenue for forensic investigation. Rich array of sensor hardware in modern devices provides new possibilities for user interaction and application development, including augmented reality, movement-based input, and environmental customizations. We have included 12 very basic and regularly used sensors.

The Sensor Manager is used to manage the sensor hardware available on Android devices. Use `getSystemService` to return a reference to the Sensor Manager Service, as shown in the following snippet:

```
String service_name = Context.SENSOR_SERVICE;
SensorManager sensorManager = (SensorManager)getSystemService(service_name);
```

Virtual Sensors that present simplified, corrected, or composite sensor data in away that makes them easier to use. To monitor a Sensor, we need to implement a `SensorEventListener`, using the `onSensorChanged` method to monitor Sensor values, and `onAccuracyChanged` to react to changes in a Sensor's accuracy. To listen for Sensor Events, we have to register the Sensor Event Listener

with the Sensor Manager. We need to specify the Sensor to observe, and the rate at which we want to receive updates.

```
Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
sensorManager.registerListener(mySensorEventListener, sensor,
                                        SensorManager.SENSOR_DELAY_NORMAL);
```

It's also important to unregister the Sensor Event Listeners when the application no longer needs to receive updates. But in our application it should always run in background; Though it affects the battery its not the priority.

```
sensorManager.unregisterListener(mySensorEventListener);
```

Register and unregister of the Sensor Event Listener in the `onResume` and `onPause` methods of your Activities to ensure they're being used only when the Activity is active

SENSOR TYPE	**VALUE COUNT**	**VALUE COMPOSITION**	**COMMENTARY**
TYPE_ACCELEROMETER	3	value[0] : X-axis(Lateral) value[1] : Y-axis [1] value[2] : Z-axis(Vertical)	Acceleration along three axes in m/s2. The SensorManager includes a set of gravity constants of the form SensorManager.GRAVITY_*.
TYPE_GRAVITY	3	value[0] : X-axis(Lateral) value[1] : Y-axis[1] value[2] : Z-axis(Vertical)	Force of gravity along three axes in m/s2. The SensorManager includes a set of gravity constants of the form SensorManager.GRAVITY_*.
TYPE_AMBIENT_ TEMPERATURE	1	value[0] :Temperature	Ambient temperature measured in degrees Celsius (C).

Table 4.1: *continues*

[1]Longitudinal

SENSOR TYPE	**VALUE COUNT**	**VALUE COMPOSITION**	**COMMENTARY**
TYPE_GYROSCOPE	3	value[0] : X-axis value[1] : Y-axis value[2] : Z-axis	Rate of rotation around three axes in radians/second (r/s).
TYPE_LIGHT	1	value[0] : Illumination	Ambient light measured in lux (lx). The Sensor Manager includes a set of constants representing different standard illuminations of the form SensorManager.LIGHT_*.
TYPE_LINEAR_ ACCELERATION	3	value[0] : X-axis(Lateral) value[1] : Y-axis[1] value[2] : Z-axis(Vertical)	Linear acceleration along three axes in m/s2 without the force of gravity.
TYPE_MAGNETIC_FIELD	3	value[0] : X-axis(Lateral) value[1] : Y-axis[1] value[2] : Z-axis(Vertical)	Ambient magnetic field measured in microteslas (μT).
SensorManager. getOrientation()	3	values[0]: azimuth values[1]: pitch axis values[2]: roll	Computes the device's orientation based on the rotation matrix.
TYPE_PROXIMITY	1	value[0] : Distance	Distance from target measured in centimeters (cm).
TYPE_PRESSURE	1	value[0] :Atmospheric Pressure	Atmospheric pressure measured in millibars (mbars).
TYPE_STEP_DETECTOR TYPE_STEP_COUNTER	1	value[0]: Step counter	Counts the number of steps taken.

Table 4.1: *continues*

SENSOR TYPE	VALUE COUNT	VALUE COMPOSITION	COMMENTARY
TYPE_SIGNIFICANT_ MOTION	1	value[0] = 1.0	Automatically wake the device to notify when significant,motion is detected

Table 4.1: Sensor Return Values.

The screenshot can be seen on Figure 7.2.

4.2.3 WiFi Metadata and Router Details

The Connectivity Manager provides a high-level view of the available network connections. The `getActiveNetworkInfo` method returns a `NetworkInfo` object that includes details on the currently active network:

```
NetworkInfo activeNetwork = connectivity.getActiveNetworkInfo();
boolean isConnected = ((activeNetwork != null) &&
                                (activeNetwork.isConnectedOrConnecting()));
boolean isWiFi = activeNetwork.getType() == ConnectivityManager.TYPE_WIFI;
```

The above code determines the connectivity.

The `WifiManager`, which represents the Android Wi-Fi Connectivity Service, can be used to configure Wi-Fi network connections, manage the current Wi-Fi connection, scan for access points, and monitor changes in Wi-Fi connectivity. To use the Wi-Fi Manager, our application must have uses-permissions for accessing and changing the Wi-Fi state included in its manifest:

```
<uses-permission android:name=''android.permission.ACCESS_WIFI_STATE''/>
<uses-permission android:name=''android.permission.CHANGE_WIFI_STATE''/>
```

Access the Wi-Fi Manager using the `getSystemService` method, passing in the `Context.WIFI_SERVICE` constant, as shown in snippet.

```
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager)getSystemService(service);
```

Monitoring Active Wi-Fi Connection Details When an active Wi-Fi connection has been established, you can use the `getConnectionInfo` method on the Wi-Fi Manager to find information

on the connection's status. The returned `WifiInfo` object includes the SSID, BSSID, MAC address, and IP address of the current access point, as well as the current link speed and signal strength, as shown in below snippet

```
WifiInfo info = wifi.getConnectionInfo();
if (info.getBSSID() != null) {
     int strength = WifiManager.calculateSignalLevel(info.getRssi(), 5);
     int speed = info.getLinkSpeed();
     String units = WifiInfo.LINK_SPEED_UNITS;
     String ssid = info.getSSID();
     String cSummary = String.format(“Connected to %s at %s%s.
                            Strength %s/5”, ssid, speed, units, strength);
}
```

Scanning for Hotspots We can also use the Wi-Fi Manager to conduct access point scans using the `startScan` method. An Intent with the `SCAN_RESULTS_AVAILABLE_ACTION` action will be broadcast to asynchronously announce that the scan is complete and results are available. Call `getScanResults` to get those results as a list of `ScanResult` objects. Each Scan Result includes the details retrieved for each access point detected, including link speed, signal strength, SSID, and the authentication techniques supported. The sample screenshot in shown in the Figure 5.3.

4.2.4 SMS

Android provides support for sending both SMS and MMS messages using a messaging application installed on the device with the `SEND` and `SEND_TO` Broadcast Intents. Android also supports full SMS functionality within your applications through the `SmsManager` class. At this time, the Android API does not include simple support for creating MMS messages from within your applications.

When a device receives a new SMS message, a new Broadcast Intent is fired with the `android.provider.T` action. Note that this is a string literal; the SDK currently doesn't include a reference to this string, so we must specify it explicitly when using it in your applications. The SMS Broadcast Intent includes the incoming SMS details. To extract the array of `SmsMessage` objects packaged within the SMS Broadcast Intent bundle, use the `pdu` key to extract an array of SMS PDUs (protocol data units — used to encapsulate an SMS message and its metadata), each of which represents an SMS message, from the extras Bundle. To convert each PDU byte array into an SMS Message object, call `SmsMessage.createFromPdu`, passing in each byte array:

```
Bundle bundle = intent.getExtras();
```

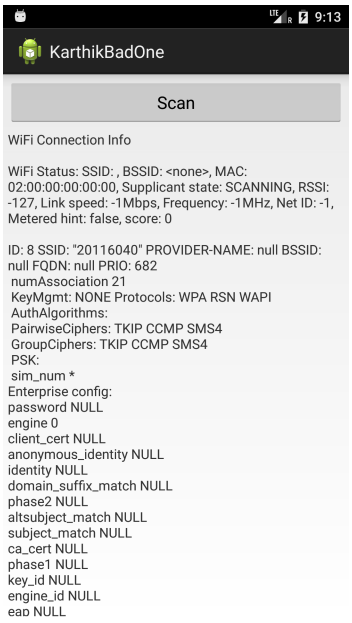

Figure 4.3: WiFi scan screen

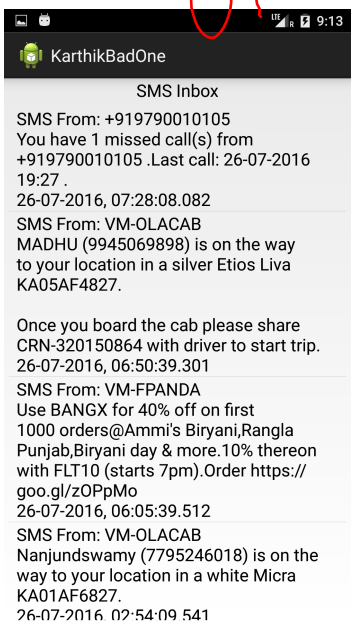

Figure 4.4: SMS screen

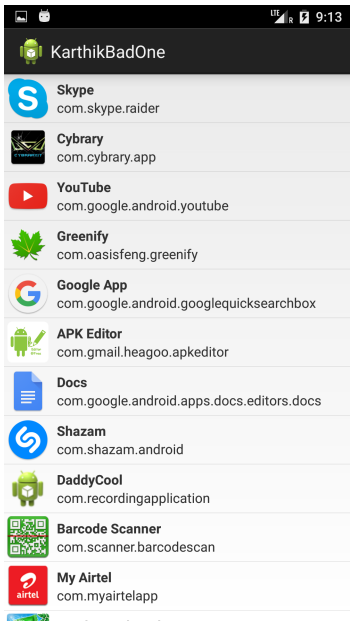

Figure 4.5: Applications

```
if (bundle != null) {
     Object[] pdus = (Object[]) bundle.get(‘‘pdus’’);
     SmsMessage[] messages = new SmsMessage[pdus.length];
     for (int i = 0; i < pdus.length; i++)
          messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
}
```

Each `SmsMessage` contains the SMS message details, including the originating address (phone number), timestamp, and the message body, which can be extracted using the `getOriginatingAddress`, `getTimestampMillis`, and `getMessageBody` methods, respectively:

```
for (SmsMessage message : messages) {
     String msg = message.getMessageBody();
     long when = message.getTimestampMillis();
     String from = message.getOriginatingAddress();
}
```

Usually SMS package will have 4 different folders where the messages are found in precise. Those are Inbox, Outbox, Sent, and Draft and can be accessed through URI

```
Uri.parse("content://sms/inbox")
```

```
Uri.parse("content://sms/outbox")
Uri.parse("content://sms/sent")
Uri.parse("content://sms/draft")
```

respectively. And to access contents of these folders we have to write

```
String address = smsInboxCursor.getColumnIndex("address");
String body = smsInboxCursor.getColumnIndex("body");
Long.valueOf(smsInboxCursor.getString(4));
```

The sample screenshot in shown in the Figure 4.4.

4.2.5 Applications Installed

Starting from the your Activity context one can obtain an instance of PackageManager through the method called `getPackageManager()`. Using that class is it possible to get a list of `ApplicationInfo` objects containing details about apps such as MetaData, Permissions, Services or Activities. Flag are very import for PackageManager . For example `PackageManager.GET_META_DATA` will retrieve only meta data for all packages. Useful data are for example the `name` of the app, the `packageName` used to retrieve additional information with PackageManager methods and the `publicSourceDir` that represent a simple way to identify system or user applications. The sample screenshot in shown in the Figure 5.5.

4.2.6 Browser Artefacts

A content provider presents data to external applications. Android comes with a number of content providers that store common data such as contact informations, calendar information, and media files, browser history and bookmarks. Here we will use Browser Provider to read default browser "Chrome" Search history and display the history of Bookmark saved. The result of querying the content provider "using `contentResolver`" is a cursor. We can map the results to the listview using `CursorAdapter`. `CursorAdapter` is a way to map each record in the cursor to a `ListView` item. Using `BOOKMARKS_URI` we can get the history of visited and bookmarked URLs. Using `SEARCHES_URI` we can get the history search terms.

```
Cursor cursor = getContentResolver().query(Browser.BOOKMARKS_URI,
                                                null, null, null, null);
```

In `AndroidManifest.xml` we need to add the following permission.

```
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
```

4.2.7 Calendar Data

The Calendar Content Provider is designed to support multiple synchronized accounts. As a result, one can choose to read from, and contribute to, existing calendar applications and accounts. Content Resolver to query any of the Calendar Provider tables using their `CONTENT_URI` static constant. Each table is exposed from within the `CalendarContract` class, including:

`Calendars`

`Events`

`Instances`

`Attendees`

`Reminders`

The Calendar Content Provider includes an Intent-based mechanism that allows you to perform common actions without the need for special permissions using the Calendar application.

```
Cursor cur = getContentResolver().query(android.provider.CalendarContract.Events.
                                        CONTENT_URI, null, null, null,null);
```

4.2.8 Contacts

Android makes the full database of contact information available to any application that has been granted the `READ_CONTACTS` permission. The Contacts Contract Provider provides an extensible database of contact-related information. This allows users to specify multiple sources for their contact information. Rather than providing a single, fully defined table of contact detail columns, the Contacts Contract provider uses a three-tier data model to store data, associate it with a contact, and aggregate it to a single person using the following `ContactsContract` subclasses:

`Data`

`RawContacts`

`Contacts`

The `ContactsContract.Data` Content Provider is used to store all the contact details, such as addresses, phone numbers, and email addresses. Android provides the `ContactsContract.Contacts.CONTENT_` query URI.

```
Intent intent = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);
```

4.2.9 Call Log Monitoring

Android can access call logs with just few lines of code. In other words call logs are built in content providers which can manage access to structured set of data. To get this step we need to have

Cursor in Android. Cursor represents results of a query pointing to one row of a result set. This way android can buffer the query results. Cursor provides various get*() methods that can be used to query the result sets.

```
Cursor managedCursor = managedQuery(CallLog.Calls.CONTENT_URI, null,null, null, null);
```

This returns a result-set with rows and column manner, using `getColumnIndex()` method, we can understand the column index of various fields like phone number, call Type, call date and call duration.

```
int number = managedCursor.getColumnIndex(CallLog.Calls.NUMBER);
int type = managedCursor.getColumnIndex(CallLog.Calls.TYPE);
int date = managedCursor.getColumnIndex(CallLog.Calls.DATE);
int duration = managedCursor.getColumnIndex(CallLog.Calls.DURATION);
```

Call code can be of 3 types Outgoing, Incoming and missed call. This can be interpreted as below:

```
switch (callcode) {
     case CallLog.Calls.OUTGOING_TYPE:
          callType = "Outgoing";
          break;
     case CallLog.Calls.INCOMING_TYPE:
          callType = "Incoming";
          break;
     case CallLog.Calls.MISSED_TYPE:
          callType = "Missed";
          break;
}
```

The sample screenshot in shown in the Figure 4.6.

4.2.10 Dictionary Word Changes

A provider of user defined words for input methods to use for predictive text input. Applications and input methods may add words into the dictionary. Words can have associated frequency information and locale information. URI of the dictionary is as given below

```
Uri uri = UserDictionary.Words.CONTENT_URI;
```

where `CONTENT_URI` is

Figure 4.6: Call log screen

Figure 4.7: SIM details screen

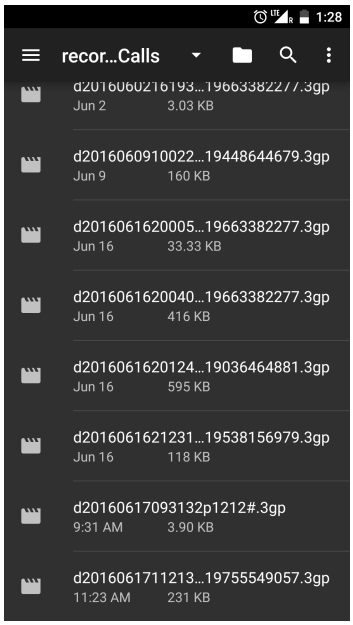

Figure 4.8: List of call recorded

```
Uri CONTENT_URI = Uri.parse("content://user_dictionary/words");
```

There are few constants which returns the value like

APP_ID - The uid of the application that inserted the word

FREQUENCY - The frequency column

LOCALE - The locale that this word belongs to

WORD - The word column

SHORTCUT - An optional shortcut for this word.

4.2.11 SIM Details

If our application is running on a GSM device, it will usually have a SIM. We can query the SIM details from the Telephony Manager to obtain the ISO country code, operator name, and operator MCC and MNC for the SIM installed in the current device. These details can be useful in the field of forensics. If one have included the `READ_PHONE_STATE` uses-permission in your application manifest, we can also obtain the serial number for the current SIM using the `getSimSerialNumber` method when the SIM is in a ready state. Before we can use any of these methods, we must ensure that the SIM is in a ready state. We can determine this using the `getSimState` method. Below is the sample snippet.

```
int simState = telephonyManager.getSimState();
```

```
String simCountry = telephonyManager.getSimCountryIso();
String simOperatorCode = telephonyManager.getSimOperator();
String simOperatorName = telephonyManager.getSimOperatorName();
String simSerial = telephonyManager.getSimSerialNumber();
```

Tracking Cell Location Changes We can get notifications whenever the current cell location changes by overriding `onCellLocationChanged` on a Phone State Listener implementation. The `onCellLocationChanged` handler receives a `CellLocation` object that includes methods for extracting different location information based on the type of phone network. In the case of a GSM network, the cell ID (`getCid`) and the current location area code (`getLac`) are available. For CDMA networks, we can obtain the current base station ID (`getBaseStationId`) and the latitude (`getBaseStationLatitude`) and longitude (`getBaseStationLongitude`) of that base station. The sample screenshot in shown in the Figure 4.7.

4.3 Call Recording

Audio Forensics has been a very important advancement in forensic science because the potential to assist law enforcement is well worth the effort it takes to defend the proponents and practitioners. Call recording is becoming increasingly important, with technology changing and working habits becoming more mobile.

In the app we have incorporated the call recording service which records the call conversation when it is received or made. The call recorder model is built up with broadcast receivers which will keep waiting for EXTRA_STATE to change so that it can start recording.

```
String extraState = intent.getStringExtra(TelephonyManager.EXTRA_STATE);
```

This extraState keeps looking for PHONE_STATE intent in TelephonyManager in the device for any changes in it. TelephonyManager.EXTRA_STATE_OFFHOOK, this lookup key used with the ACTION_PHONE_STATE_CHANGED broadcast for a String containing the incoming phone number. Only valid when the new call state is RINGING. TelephonyManager.EXTRA_STATE_IDLE value used with EXTRA_STATE corresponding to CALL_STATE_RINGING. TelephonyManager.The EXTRA_STATE_RINGING value used with the EXTRA_STATE corresponding to the CALL_STATE_OFFHOOK.

The audio should have proper codec so that the recordings should be clear to distinguish or identify the voice and less in size. The encoder and file format should be assigned.

```
recorder = new MediaRecorder();
```

```
recorder.setAudioSource(MediaRecorder.AudioSource.VOICE_CALL);
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

The file saved should be with proper name. So came up with a below naming convention.

```
file.getAbsolutePath() + "/d" + myDate + "p" + phoneNumber + ".3gp"
```

where `myDate` is

```
myDate = (String) DateFormat.format("yyyyMMddkkmmss", new Date());
```

and `phoneNumber` is

```
phoneNumber = phoneNumber.replaceAll("[\\*\\+-]", "")
```

To work this app we need to set permission in the Manifest.xml file. Some of the very basic permissions like WRITE_EXTERNAL_STORAGE, RECORD_AUDIO, STORAGE and some related to state of the phone like PROCESS_OUTGOING_CALLS, READ_PHONE_STATE, and READ_PHONE_STATE The sample screenshot in shown in the Figure 4.8.

4.4 Keylogger

Monitoring keystrokes in Android is not possible as each app runs on the its respective sandbox. So we have to write a fully functional keyboard. Only a small method needs to be added to it. The snippet is given below

```
public void onKey(int primaryCode, int[] keyCodes) {
    String keypress = String.valueOf((char)primaryCode);
    Log.d("Key Pressed",keypress);
    try{
        String SDCARD = Environment.getExternalStorageDirectory().getAbsolutePath();
        String FILENAME = "keylogger.txt";

        File outfile = new File(SDCARD+File.separator+FILENAME);
        FileOutputStream fos = new FileOutputStream(outfile,true);
        fos.write(keypress.getBytes());
        fos.close();
    }catch(Exception e) {
```

```
        Log.d("EXCEPTION",e.getMessage());
    }
    [...]
}
```

4.5 Uploading to Cloud

Google Drive Google wants more developers to integrate their Android apps with its storage service Drive, and has released an API that aims to make it easier. While Drive integration on Android was possible in the past, the new API offers developers better performance and more features. The native, Java-based Drive Android API is still under development, but developers can now test a preview version that's included in Play Services. The API includes the ability to temporarily use storage on the device if it is not connected to a network. The upshot for developers is that they don't have to worry about failed API calls when the phone is offline or experiencing network connectivity problems. Data stored locally will then be automatically transferred to Drive by Android's sync scheduler when connectivity is reestablished. The API also includes Android user interface components and specialized functionality for interacting with files stored on Drive.

Smartphones and tablets running Android operating system are all compatible with the Drive Android API. The addition of the API will have little effect on the total size of an app. Applications compiled using the preview version will continue to work on devices using future versions of Play services. However, changes to the interface are expected in future releases, which means that developers have to rewrite their apps to take of that.

Online storage is a crowded field, and Google isn't the only company that wants to make its service more useful by opening the door for third party app integration. For example, competitor Dropbox offers a number of APIs for developers of Android apps, as well as other platforms. The ability to save files using the another user id which is infact the perfect for our app where we have to send to investigator id. This feature was not available in Google Drive so we switching to Dropbox.

Dropbox Dropbox is a popular free service that helps share your files easily. It also provides certain API's that developers can use to create applications that upload files directly on Dropbox. In order to connect to Dropbox, you first need to Create an Account on their website. Once the account is created successfully, we can start writing our own applications using the Dropbox API's. In order to create an application, visit the App Console and select the Create App option.

We first have to define the app key and app secret, generated from the dropbox developer console. The detail configuration is explained in Appendix E. In addition, for full dropbox interaction, you

should provide Dropbox AccessType.

```
private final static AccessType ACCESS_TYPE = AccessType.DROPBOX;
```

Moreover, we define the folder that we'll use from our app to interact with our dropbox account (upload/list files); if the provided name doesn't correspond to an existing folder, a new folder will automatically be created. Furthermore, for better interaction, session handling and user authentication, we created the private, `loggedIn` function.

```
public void loggedIn(boolean userLoggedIn) {
     isUserLoggedIn = userLoggedIn;
     uploadFileBtn.setEnabled(userLoggedIn);
     uploadFileBtn.setBackgroundColor(userLoggedIn ? Color.BLUE : Color.GRAY);
     listFilesBtn.setEnabled(userLoggedIn);
     listFilesBtn.setBackgroundColor(userLoggedIn ? Color.BLUE : Color.GRAY);
     loginBtn.setText(userLoggedIn ? "Logout" : "Log in");
}
```

We are running this app in background so no need of button on the activity. On the very first access to our app, we want the user to be unable to press any other, except the login button. Thus, we call the `loggedIn` function with a false value.

```
AppKeyPair appKeyPair = new AppKeyPair(ACCESS_KEY, ACCESS_SECRET);
AndroidAuthSession session;
SharedPreferences prefs = getSharedPreferences(DROPBOX_NAME, 0);
String key = prefs.getString(ACCESS_KEY, null);
String secret = prefs.getString(ACCESS_SECRET, null);
if (key != null && secret != null) {
     AccessTokenPair token = new AccessTokenPair(key, secret);
     session = new AndroidAuthSession(appKeyPair, ACCESS_TYPE, token);
} else {
     session = new AndroidAuthSession(appKeyPair, ACCESS_TYPE);
}
dropboxApi = new DropboxAPI(session);
```

The above snippet describe the authentication process, either if the user has selected to save his credentials or not. Especially, if not, we create a new session with the existing app key, app secret and access type; otherwise, we retrieve his credentials with the help of `SharedPreferences` and create a session using them. We override the `onResume` method to continue a valid session in case

of pausing the app. Again, this happens with the help og `SharedPreferences`. Obviously, we here pass a true value to our `loggedIn` function.

4.6 Hide Forensic Process

The process extracting data or sending to the cloud, should be stealthy. Normal users 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 process, but hiding folder `/proc/PID/` or refusing access to it will be an fair approach.

The command `hidepid` hides processes and its information to other users; it accepts 3 values. Default is hidepid=0, where all settings will be default and any user can see processes running in background.
When hidepid=1, normal user would not see other processes but their own about ps, top etc, but he is still able to see process IDs in /proc.
When hidepid=2, user can only able too see their own processes also the process IDs are hidden from /proc also.

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.

4.7 Impact on Battery Consumption

Even in 2016, it's tough to go much longer than 24 hours without charging the smartphone. Better battery technology simply hasn't arrived yet, which means it's down to software and settings to eke out the limited power for as long as possible. Most smartphones have either a Lithium Ion battery or a Lithium Polymer battery. Both are Lithium Ion though, and as such, do not have a 'memory', which means one does'nt have to fully charge or discharge them at the beginning, and partial charging is fine throughout their life. Unfortunately, we are never going to get a week's use out of a smartphone because of those big, bright screens along with Wi-Fi, Bluetooth, GPS and 3G.

Connecting to Internet affects the battery consumption due to process running in the background. Especially if we continuously record the video and uploading to the cloud without connecting to charger. Battery consumption will be unnoticeable when we are transmitting text files as they are small in size. Even with weak network it's easy to transmit. The problem comes when we are uploading larger files like video or audio or other files.

In this application when we set video recording it consumes lot of battery. When we disable the it we can have noticeable battery life. But still we are monitoring always with GPS on tracking trail of user recording it on text file. Similarly other modules are continuously keeping track. There will be almost unnoticeable amount of battery consumption. But not negligible as this app runs continuously, monitors, tracks, trails, copies files, writes to the files, uploads to the cloud, deletes which should provide decent amount of battery life.

4.8 Contribution to Lag

Whenever an app is installed and running obviously it will consume some amount of RAM. When app is not closed it keeps consuming the memory. Having resource-hungry apps running in the background can really cause a huge drop in battery life. GPS tracking, background syncs and video recording can cause the device to never sleep or at times cause noticeable lag in the running of applications. When we have a dozen of monitors monitoring everything in the device that will cause a performance dip on the Android device. Some devices with lesser storage capacity suffer from a low-memory issue and most perform poorly once they hit around 80% of their device storage capacity especially in case of larger unsent videos.

The lag caused by cloud upload is less visible to the user as it is achieved by uploading when the device is idle.

4.9 Local Storage

Forensic data is stored separately in the folder ‘ `forensic`’ and it is hidden from the users. The proposed plan to create a separate partition which will be hidden from the user. Forensic data storage location being separated from user data partition, ensures that the changes in the storage space because of the forensic data remains hidden. The data collected gets cleared from this partition once it gets uploaded to the Cloud. To create a separate partition in Android is to write our own firmware.

So, as an alternative we have created a hidden folder which is easy to create and basically invisible to common user where all the files are stored either in text files or otherwise. All the data irrespective of the type once uploaded to cloud and deleted. So whenever device is connected to longer time the data in the folder will be empty. TO create a hidden file in Android is similar to creating hidden folder in Linux, just add an extra dot (.) right before all the characters and save it.

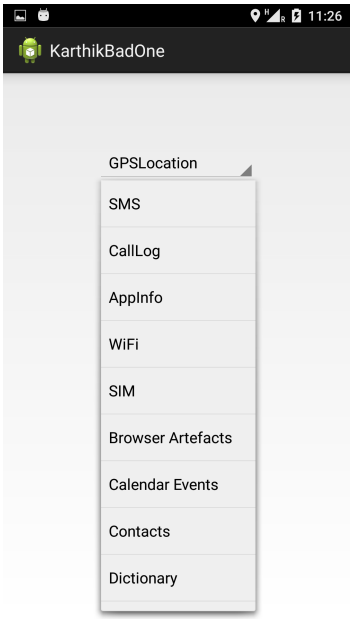

Figure 4.9: KarthikBadOne home screen

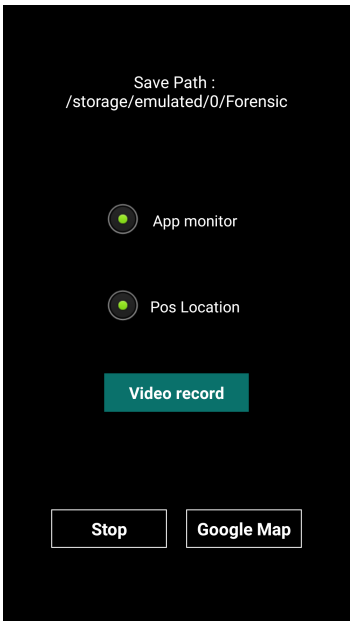

Figure 4.10: KarthikDaddy_cool home screen

4.10 Evaluation of the APK

We created two APKs as part of implementation where all the above mentioned features has been implemented and explained briefly its functionality with screenshot. The app is named as KarthikBadOne(hence forward referred as APK1) and KarthikDaddyCool(hence forward referred as APK2); These names have nothing in reference to the project but just a name. The Figure 4.9 shows the home screen of the APK1 where all the different modules are listed and the Figure 4.10 shows the home screen of the APK2. APK1 is straight forward where the user can select from the dropdown box and respective activity will be opened. In APK2 we can actually select which option we want to run. Though the radio button it works like checkbox. To run the app we need to click on Start button.

When the user selects the SMS from dropdown it shows all the SMS present in the device which is shown in the Figure 4.11. This has From, message, date fields. If user selects CallLog then it would show from whom it was called or called to plus it shows the number and name, date of call made and duration of the call. The screenshot is shown in the Figure 4.12. AppInfo, if selected, shows the application installed in the user device with the package name as shown in Figure 4.13.

WiFi shows all the currently connected WiFi data as well as previously connected data with full details available from the connection as shown in the Figure 4.14. SIM details are collected like Cell location, serial number etc as shown in Figure 4.15. Also if Browser artefacts is selected then it

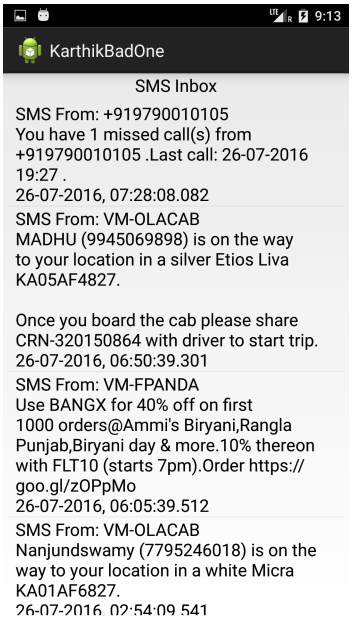

Figure 4.11: SMS screen

Figure 4.12: Call log screen

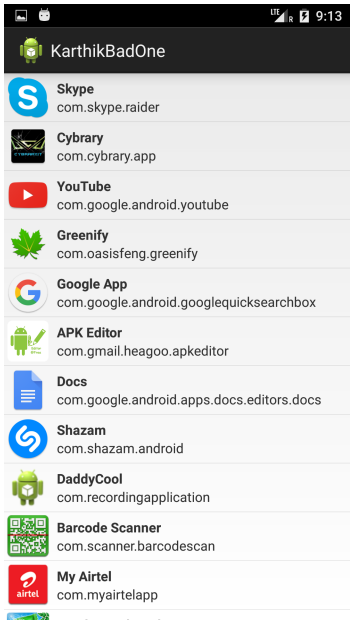

Figure 4.13: Applications

shows the details of browser history which contains the data like Date, Title, Type and URL like shown in the Figure 4.16.

Now if user selects Contacts then it shows all the contacts available on the device. It would also includes the names of user if number is not available as shown in the Figure 4.17. The event created by user can be seen from the Calendar dropdown as shown in the Figure 4.18. The dictionary words user saved can be seen using the Dictionary selection from the dropdown which is shown in Figure 4.19.

Figure 4.20 only appears once as asking to select Enable Recording feature and since then whenever the call is made or received a Toast message is displayed. These are saved in the folder as shown in Figure 4.21. The sensor, if selected from dropdown list, then it shows screen like Figure 4.22 where all details of sensor data are shown on the screen.

GPS Location shows the exact latitude and longitude of the device in the text format. It shows GPS and Network location as shown in Figure 4.23. Till here it was APK1. Now will see APK2. The same GPS is implemented with Google Maps API where data of phone travelled can be seen as trail on the map as shown in the Figure 4.24. All the data is captured onto a text as shown in Figure 4.25.

The video is recorded stealthily and saved onto a sdcard. Figure 4.26 shows the quality of the video recorded through a sample screenshot. Figure 4.27 shows the list of video recorded. The list of app running its active time is recorded as shown in the Figure 4.28.

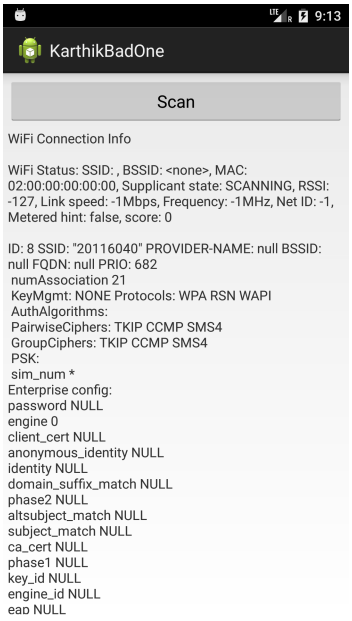

Figure 4.14: WiFi scan screen

Figure 4.15: SIM details screen

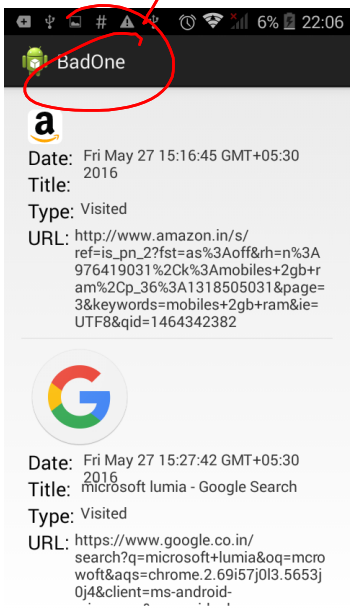

Figure 4.16: Browser History screen

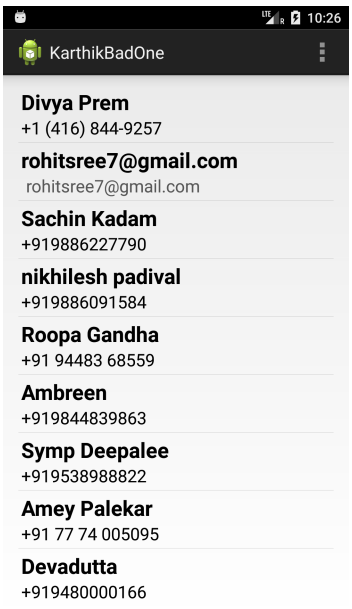

Figure 4.17: Contacts screen

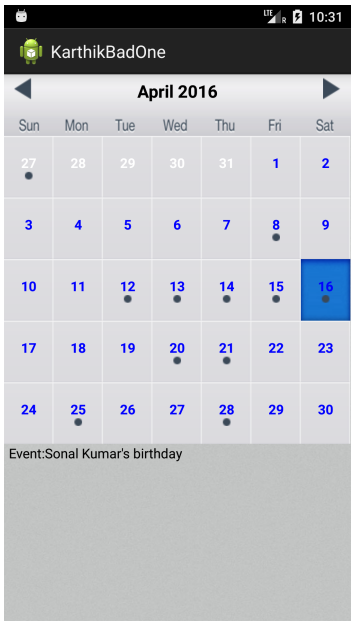

Figure 4.18: Calendar events screen

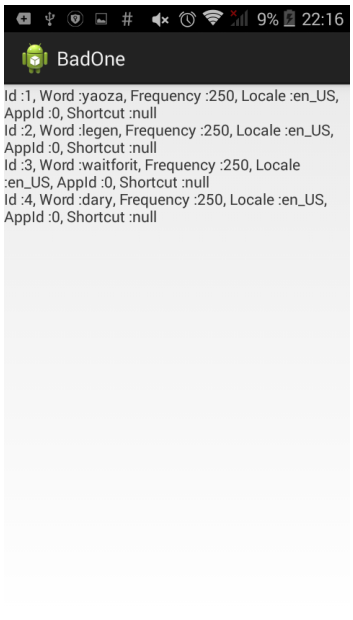

Figure 4.19: Dictionary word screen

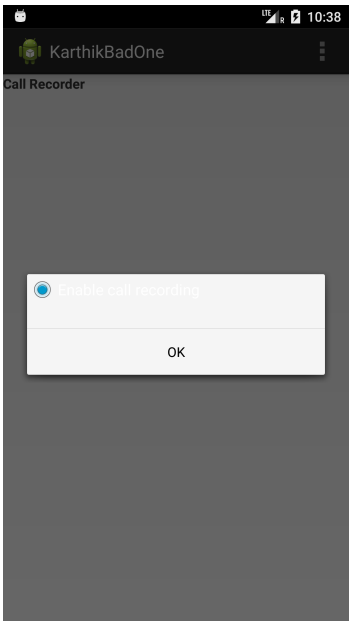

Figure 4.20: Pop-up for call recording

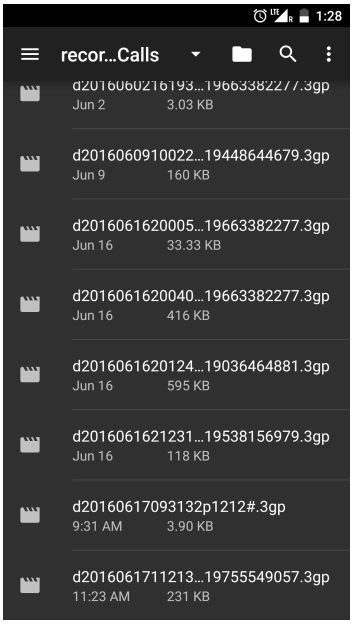

Figure 4.21: List of call recorded

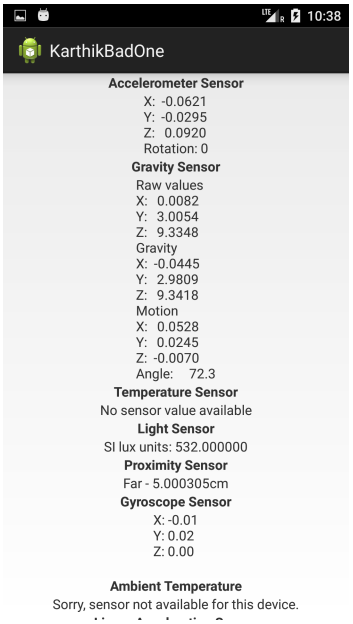

Figure 4.22: Sensor data

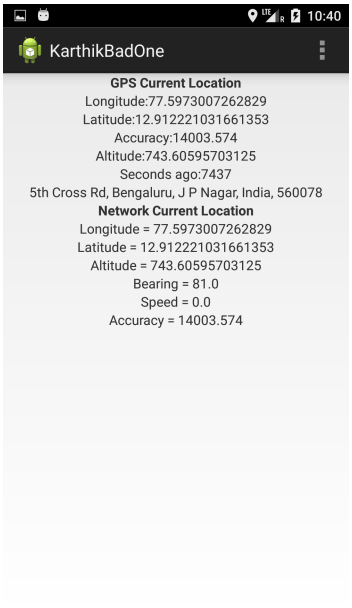

Figure 4.23: GPS from APK1

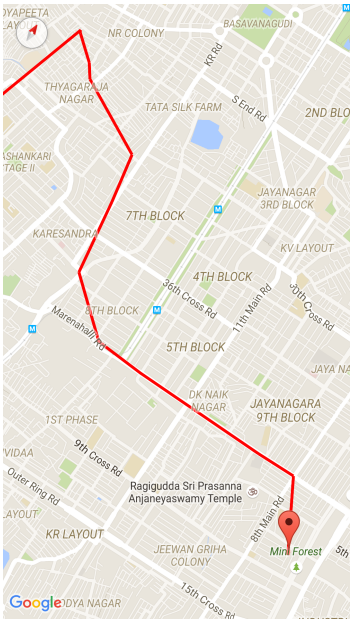

Figure 4.24: GPS from APK2

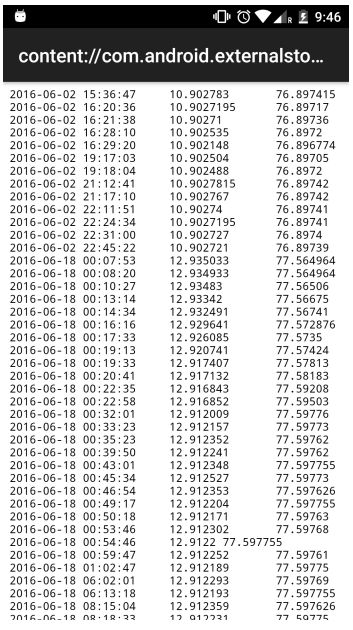

Figure 4.25: List of recorded location

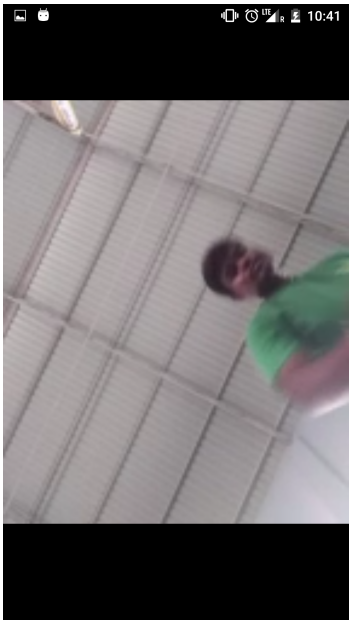

Figure 4.26: Screenshot of the video

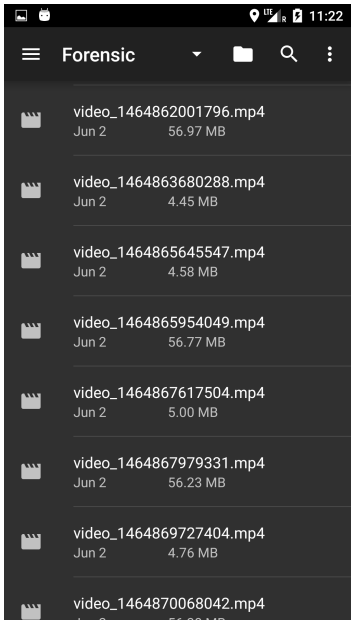

Figure 4.27: List of video recorded

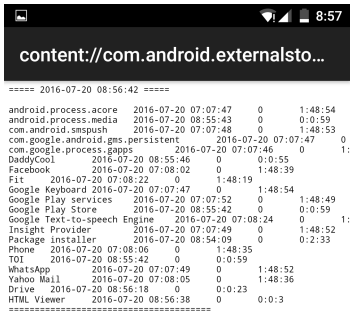

Figure 4.28: List of App active

4.10.1 Development details

There are basically 2 apps that are part thesis implementation. Though it was developed separately the aim was to merge. I'm finding difficulty in merging the app its with either APK1 to APK2 or vice-versa. Both app works fine when separated. SLOC shows the number of lines of code. APK1 has

```
cloc BadOne
      98 text files.
      96 unique files.
      42 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.55 s (163.1 files/s, 13325.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                            50            927            469           4162
XML                             40            204             51           1542
-------------------------------------------------------------------------------
```

```
SUM:                            90           1131            520           5704
-------------------------------------------------------------------------------

cloc All_Gps-Rec_Map_Monitor\(com\)App     117 text files.
     114 unique files.
     192 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.69 s (135.9 files/s, 12347.3 lines/s)
-------------------------------------------------------------------------------
Language                      files          blank        comment           code
-------------------------------------------------------------------------------
XML                              66             44              9           3987
Java                             25            584           1527           2124
Bourne Again Shell                1             19             20            121
DOS Batch                         1             24              2             64
IDL                               1              2              0             15
-------------------------------------------------------------------------------
SUM:                             94            673           1558           6311
-------------------------------------------------------------------------------
```

Source code of APK1 can be found at `https://github.com/mrkarthik07/Forensidroid` and APK2 would be available at `https://github.com/mrkarthik07/Daddy\_cool`.

5

Related Work

The cry for forensics in mobile devices has been out loud in recent times as people are tending to use smart phones to a greater extent which are of high end, of course.

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

5.1 Ethics

While there are legal requirements regarding forensic investigations, there are also ethical requirements, and these are just as important. As a forensic investigator, one need to keep two facts in mind. The first is that there are significant consequences resulting from the work. Whether one is working on a criminal case, a civil case, or simply doing administrative investigations, major decisions will be made based on ones work. One heavy ethical obligation. It is also critical to remember that a part of 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 the character. The 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.

5.2 Companion Projects at Amrita

There has been research work going on to build a customized full fledged security enhanced Android ROM at Amrita.

Network Ombudsman for Android. 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.

Auditing Android system for Anomalous Behaviour Another is Auditing Android System for Anomalous Behaviour[Kamardeen 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.

5.3 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.

5.4 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.

5.4.1 Proactive

Smartphone Forensics: A Proactive Investigation Scheme for Evidence Acquisition [Mylonas et al. 2012]

Here authors clearly explains the meaning of the term proactive, and all the questions like who, where, when, what, why, how regarding the retrieval of the data. Below are the evidence types in the smartphone Identity Evidence - Data identify subjects that are part of an event. Location Evidence - Data define the approximate or exact location, where an event takes place. Time Evidence - Data can be used to infer the time that an event takes place. Context Evidence - Data provide adequate context, such as user actions and activities for an event description, or the event nature. Motivation Evidence - Data can be used to determine event motivation. Means Evidence - Data describe the way that an event took place, or the mean that were used.

The sources of the smartphone data are Messaging Data, Device Data, (U)SIM Card Data, Usage History Data, Application Data, Sensor Data and User Input Data

Correlation between evidence types and data sources has been done to establish the relation. The mode of transportation has been chosen based on these data type. The authors analysed the different modes of transportation

1. GSM Messaging interface
2. Personal Area Network (PAN) interface
3. WLAN interface
4. Cellular network

It was clear that WLAN provides the best solution among the above.

Proactive Smartphone Forensics Scheme an investigator, an Independent Authority (IA), and the suspect are the three entities. When an investigator requests, the session is established and evidence type request is sent. IA sends the request to suspects device. The data is collected on the device and sends to IA. The sent data is stored on evidence DB. In other words, the proposed scheme consists of six building blocks - processes, namely: investigation engagement, evidence type selection, evidence collection, evidence transmission, evidence storage and investigation completion

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 the 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 includes call records, SMS, Camera events, GPS location, modified files, etc.

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 out of 33 features which 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 of the software is little tricky and User friendliness is little lacking.

Android Forensics: Automated data collection and reporting 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.

5.4.2 Reactive

Forensic Analysis of Smartphones: The Android Data Extractor Lite (ADEL) [Freiling et al. 2011] extracts, analyses and reports the forensic data from the mobile device. SQLite database are extracted through software based approach with help of a software agent stored in Android device, that collects and analyses data locally. The paper explains about 3 different methods of extracting data

1. Software-based Approach: Software Agents

   Agent based approach has a software agent resides in Android device, that collects and analyses data locally. It uses content provider to get application database. ADEL can get deleted data as long as it exists in dbExport data in CSV format.

2. Using the adb

   This method uses SDK and dump the data to investigators machine by using adb. Android devices in general deny the access to databases via the adb. Rooted devices can be used as solution or use a bootable 'goldcard' in order to change the status of the phone in a way so that these security restrictions have been deactivated or can be bypassed.

3. Hardware-based Approach: De-soldering Memory Chips

   A hardware based approach is also introduced in which it uses desoldering of chips to get the data back. By this method memory chips is removed and is desoldered and the chip is analysed using special hardwares named PC-3000 Flash.

DB engine is integral part of applications which store the data like Contacts, GPS, Call list, SMS, etc. The leading 100 bytes of the first page of a database file are used to store the database header and stores general data about the database. Each table is internally represented as B-tree structure. First page contains sqlite_master table which stores the DB schema.

ADEL extracts the contents from SQLite DB and stores within the dump DB files. ADEL makes use of the Android SDK to dump database files to the user's machine. ADEL reads the database header and extracts the values for each of the header fields. Complete b-tree structure is parsed for each table.

Extracted data should be presented as a report formatted in XML format.

New acquisition method based on firmware update protocols for Android smartphones [Yang et al. 2015] The authors developed a tool Android Physical Dump(APD), which acquire physical memory of android based on firmware. The process requires a physical acquisition method

to acquire the entire flash memory, rather than a logical acquisition method. The analysis has been done for 3 devices LG, Pantech, and Samsung. There is a detail explanation on firmware updates. It clearly explains the address location. Also it explains all the flash memory dump commands required for these three devices. The analysis and differentiation between these 3 devices' firmware and memory location of it are explained in details. The smartphone is booted in the firmware update mode and data is collected with the help of ADP tool. The proposed system claims to be fastest in extracting the data when compared to all the professional tools available in the market. This claims to be the best in preserving the integrity of the dump images. Also this proposed software claims to extract the data from the locked device.

5.4.3 Forensic Related

Analysis of WhatsApp Forensics in Android Smart-phones [Sahu 2014] 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.

Fingerprints On Mobile Devices Abusing And Leaking [Zhang et al. 2015] The authors revealed some severe issues with the current Android fingerprint frameworks that have been neglected by vendors and users. Authors provided in-depth security analysis of the popular mobile fingerprint authentication/authorization frameworks and discussed the security problems of existing designs, including

- the **confused authorization attack** that enables malware to bypass pay authorizations protected by fingerprints,
- **insecure fingerprint data storage**, the fingerprint in HTC One is saved as /data/dbgraw.bmp with 0666 permission
- fingerprint sensor exposed to the untrusted world, malware can be so that it can use IOCTL_REGISTER_DRDY_SIGNAL to register as the event signal listener of the sensor

device.

- **pre-embedded fingerprint backdoor**.

Authors also provided suggestions for vendors and users to better secure the fingerprint, like normal users should choose mobile device vendors with timely patching/upgrading to the latest version

Ensuring the Authenticity and Non-Misuse of Data Evidence in Digital Forensics [He et al. 2015] The study is focused mainly preserving the extracted data's integrity, authenticity, validity and making sure the data is if fallen into wrong hands not to be misused. This is done using cryptographic algorithm AES/DES, Digital Signatures and one way hash functions.

The proposed system is as follows

1. **Authenticity** The incorporation of the device ID in the hashing of the data ensures that the data obtained are indeed from the very target device while the timestamp attached would mark the exact time when the data are extracted. Through a Hash operation to bind the data together with the device ID and the timestamp, the forensic server can be sure that the data are acquired from the indicated device at a specific time. Thus, the authenticity of the digital evidence is guaranteed. **Non misuse** The use of the DES/AES and RSA cryptographic algorithms can help ensure the confidentiality of the data evidence. The data evidence is encrypted with DES/AES algorithm to ensure its confidentiality and the secret key using in DES/AES is encrypted with the public key of the forensic server. That is, the digital evidence is protected with two level encryptions. Even though some one can touch or intercept the data evidence in transmission, the data itself cannot be disclosed. In addition, the incorporation of the device ID in the whole process makes sure that the digital evidence can hardly be misused.

2. **Integrity** The forensic server can check whether the digital evidence is modified through a Hash algorithm. If the evidence is modified accidentally or intentionally, the forensic server can be able to detect the modification through comparing the two Hash values. Thus,the integrity of the digital evidence is guaranteed.

3. **Validity** Firstly, the use of device ID ensures that the data evidence is indeed obtained from the target device that really exists, which proves the objectivity of the evidence. Secondly, all the mechanisms used would ensure the authenticity and non misuse of the data evidence to guarantee its reality. Thirdly, since all the data are extracted from the target device that is related to a case under investigation, the data evidence should have some relationship with

the case, which ensures the relevance of the data evidence. Thus, the data evidence obtained through the proposed method is valid.

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 analysing 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.

6

Conclusion and Future Work

Forensic investigators who do not have access to expensive commercial Android forensic tools need lot of efforts to accomplish their job. Also the existing tools are all reactive. Though there are impressive research findings, the Android forensic field lacks collaboration among findings, and has reduced their worth. In this study we implemented a forensic framework for Android smartphones to serve the Android forensic community with powerful features that serve proactive measures. Our work supports forensic investigator in all four phases of a mobile forensic investigation. We made changes to the device to extract comprehensive set of evidence. Traditional forensics avoids changes to the device. However due to the Android architecture, valuable evidence cannot be collected in the traditional reactive approach. Therefore this study established the fact that 100% forensic soundness is not possible in Android forensics. This thesis contains a very good model for the proposed idea with a minimum implementation.

There is a lot of scope for future work in the implementation as well as the idea. We can have a GUI screen on the investigator side which can be constructively provide rich enhanced view of the evidence. Also the encryption can be provided to data before uploading to the cloud. The stealth camera recording should be improved before video is turned on; it shouldn't provide any beep sound and also when device camera is turned on stealth video should stop recording and continue when device camera is turned off. The real time monitoring could be improved where the data should be sent based on the priority even when WiFi is not available. The data can be gathered is systematic and precise way and only data needed can be gathered. Also one major change is one can make this app to be controlled remotely. Even installing the app onto a device remotely without user knowledge is possible in future.

7

Appendix

7.1 `inotifywait` in Linux

This monitors events of /root/tmp directory and its the built in for linux as shown in Figure 7.1.

```
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

7.2 FileObserver log

Below shows the output of FileObserver logs done on the sample text file named FileObserver where it was created, open, written and other activities done with it.

```
I/FileObserver( 3596): FileObserver.txt:Create
I/FileObserver( 3596): FileObserver.txt:Open
I/FileObserver( 3596): FileObserver.txt:Modify
I/FileObserver( 3596): FileObserver.txt:CloseWrite
I/FileObserver( 3596): FileObserver.txt:Delete
```

```
I/FileObserver( 3596): FileObserver.txt:Create
I/FileObserver( 3596): FileObserver.txt:Open
I/FileObserver( 3596): FileObserver.txt:Modify
I/FileObserver( 3596): FileObserver.txt:CloseWrite
I/FileObserver( 3596): FileObserver.txt:Delete
I/FileObserver( 3596): FileObserver.txt:Create
I/FileObserver( 3596): FileObserver.txt:Open
I/FileObserver( 3596): FileObserver.txt:Modify
I/FileObserver( 3596): FileObserver.txt:CloseWrite
```

7.3 Sample output of Sensors

The sample screenshot of sensor activity is shown in Figure 7.2. The accelerometer sensor which shows X, Y, Z axis and rotation. Also below accelerometer it shows the values of gravity sensor. Below that there is temperature sensor as the device not supporting it shows error message. Light sensor shows the light on the device in lux units. Similarly proximity, gyroscope and ambient temperature values are shown on the screen.

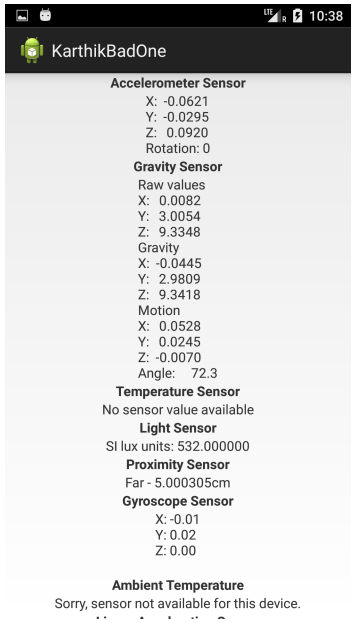

Figure 7.2: Screenshot of sensor data

7.4 Code to upload to Dropbox

The main objective of this section is to describe the Dropbox API in Android, by providing step by step guidance and easy to understand the implementation of Dropbox API.

Step 1 Dropbox SDK

To use the Core API, download the Dropbox Android SDK and also we need to add Json Simple jar file to handle Json type request and response in your android project library(app/libs) folder. To use features of Core API, we need to compile SDK in project. So, add `Dropbox SDK` and `json_simple jar` files as project dependencies. Open `build.gradle` file and add files as dependencies as shown below:

```
dependencies {
    ...
     compile files('libs/dropbox-android-sdk-1.6.3.jar')
     compile files('libs/json_simple-1.1.jar')
    ...
}
```

Step 2 Authentic Process

Go to Url and complete the authentication process as shown in Figure 7.3. See more at:

- https://www.dropbox.com/developers/apps

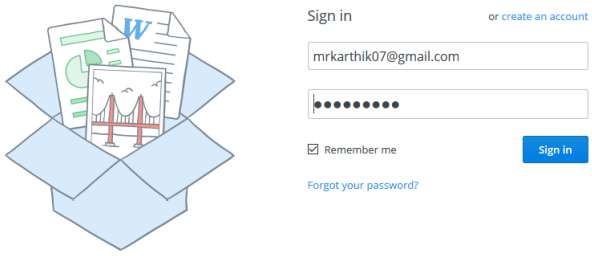

Figure 7.3: Home screen of Dropbox

Step 3 Register App on App console

Go to App Console and click on Create App button to create a new app as show in Figure 7.4.

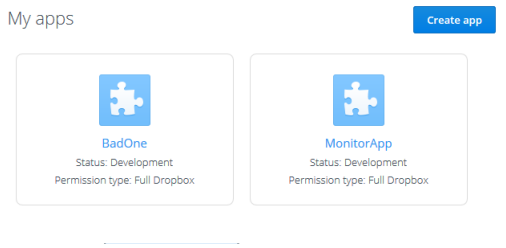

Figure 7.4: App Console of Dropbox

Step 4 Select The Dropbox API

Select the Dropbox API app as shown in Figure 7.5.

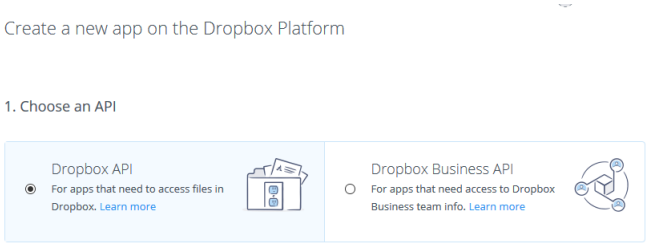

Figure 7.5: App Console of Dropbox

Step 5 Set Privacy Statement

Set the privacy statement access files or folders, here select App folder for accessing files which are already on Dropbox as shown in the Figure 7.6.

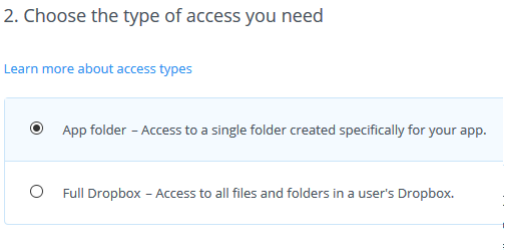

Figure 7.6: App Console of Dropbox

Step 6 Provide App Name

In the textbox type your application name, if that name is already registered then you cannot create with the same name. After inputting the desired name of your Dropbox app click on Create app button as shown in the Figure 7.7.

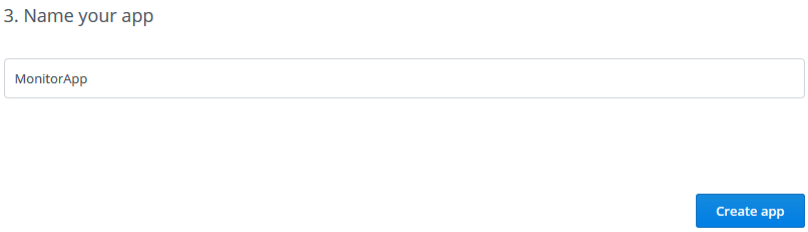

Figure 7.7: Dashboard of Dropbox

Step 7 DashBoard

We will be redirected to dashboard screen which contains all the useful parameters. We have to use that parameters in our app to access Dropbox account from our device. Here as shown in the figure below the App Key and App Secret provides unique key, which we have to pass in your application to access your Dropbox account form mobile devices as shown in the Figure 7.8.

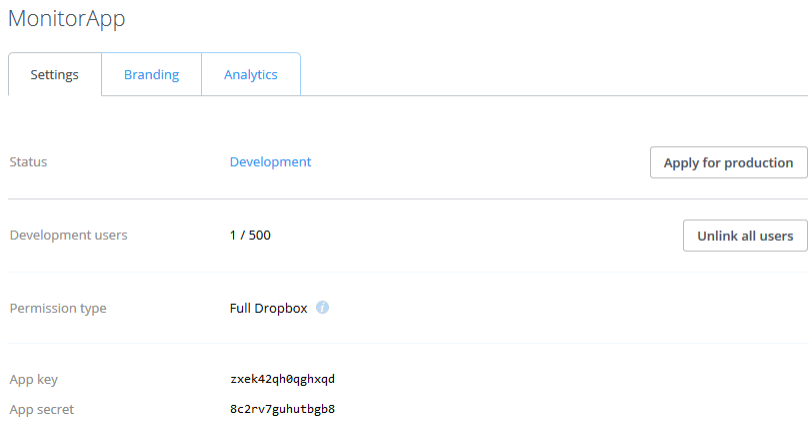

Figure 7.8: Dashboard of Dropbox

Step 9 Enable Additional Users

By default app is in development status and has all the features available to use as production status. Only limitation of development status is that app can be used only by test users. App can be used by up to 500 test users in development status. To make app accessible to test users, click on Enable Additional Users button. If we want to share app with the world, apply for production status.

Step 10 Dropbox Implementation

10.1 Apply Secret Key

Implementation of Dropbox in our project. Apply `DROPBOX_APP_KEY` and `DROPBOX_APP_SECRET` as a constants and copy the values of both the constants from the created application previously shown in STEP 7.

```
final static public String DROPBOX_APP_KEY = " Dropbox App Key here";
final static public String DROPBOX_APP_SECRET = " Dropbox App Secret";
final static public AccessType ACCESS_TYPE = AccessType.DROPBOX;
```

10.2 Create Session

Create Session which allows our application to authenticate to Dropbox API.

```
private DropboxAPI mApi;
```

Add the below lines in `onCreate()` method.

```
AndroidAuthSession session = buildSession();
mApi = new DropboxAPI(session);
- See more at: http://www.theappguruz.com/blog/10-steps-integrate-dropbox-api-andr
```

Method for creating session:

```
private AndroidAuthSession buildSession() {
     AppKeyPair appKeyPair = new AppKeyPair(Constants.DROPBOX_APP_KEY,
                                            Constants.DROPBOX_APP_SECRET);
     AndroidAuthSession session;
     String[] stored = getKeys();
     if (stored != null) {
          AccessTokenPair accessToken = new AccessTokenPair(stored[0],
                                                            stored[1]);
          session = new AndroidAuthSession(appKeyPair,
                                           Constants.ACCESS_TYPE,accessToken);
     } else {
          session = new AndroidAuthSession(appKeyPair, Constants.ACCESS_TYPE);
     }
     return session;
}
```

10.3 AndroidManifest.xml file

Changes in the `AndroidManifest.xml` file

```
<intent-filter>
     <!-- Change this to be db- followed by your app key -->
     <data android:scheme="db-n81vuqu3mfexf6i" />
     [...]
</intent-filter>
```

In the above `AndroidManifest.xml` file there is a line of `android:scheme`, which contains the key. We are required to change the key value of our application which we can find form Step 7.

10.4 startAuthentication

Apply the code below in `onActivityResult()` method to complete the authentication process.

mApi.getSession().startAuthentication(Main.this)

```
onResume = true;
@Override
protected void onResume() {
    AndroidAuthSession session = mApi.getSession();
    if (session.authenticationSuccessful()) {
        try {
            session.finishAuthentication();
            TokenPair tokens = session.getAccessTokenPair();
            storeKeys(tokens.key, tokens.secret);
            setLoggedIn(onResume);
        } catch (IllegalStateException e) {
            showToast("Couldn't authenticate with Dropbox:"
                                                + e.getLocalizedMessage());
    }
}
super.onResume();
}
private void storeKeys(String key, String secret) {
    SharedPreferences prefs = getSharedPreferences(
                                          Constants.ACCOUNT_PREFS_NAME, 0);
    Editor edit = prefs.edit();
    edit.putString(Constants.ACCESS_KEY_NAME, key);
```

```
        edit.putString(Constants.ACCESS_SECRET_NAME, secret);
        edit.commit();
    }
```

10.5 Upload file

Code to upload file in Dropbox after authentication:

```
public void setLoggedIn(boolean loggedIn) {
    mLoggedIn = loggedIn;
    if (loggedIn) {
        UploadFile upload = new UploadFile(Main.this, mApi, DIR, f);
        upload.execute();
        onResume = false;
    }
}
```

10.6 What is DIR?

DIR is the name of directory of Dropbox where we can upload our files and folders of specified directory. F: **f** parameter in `UploadFile()` is a file object which we want to upload in Dropbox.

10.7 Get All Files and Folders from Dropbox

To get all files and folders of Dropbox, use below code:

```
com.dropbox.client2.DropboxAPI.Entry dirent = mApi.metadata(
                                                DIR, 1000, null, true, null);
files = new ArrayList<com.dropbox.client2.DropboxAPI.Entry>();
dir = new ArrayList<String>();
for (com.dropbox.client2.DropboxAPI.Entry ent : dirent.contents) {
    files.add(ent);
    dir.add(new String(files.get(i++).path));
}
```

References

Aiyyappan, P. 2015. Android forensic support framework. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. Advisor: Prabhaker Mateti, `http://cecs.wright.edu/~pmateti/GradStudents/index.html`.

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.

Alharbi, S., Weber-Jahnke, J., and Traore, I. 2011. The proactive and reactive digital forensics investigation process: A systematic literature review. In *International Conference on Information Security and Assurance*. Springer, 87–100.

Android. 2015. The Android source code. `https://source.android.com/source/index.html`.

AppBrain. 2016. Number of Android applications. `http://www.appbrain.com/stats/number-of-android-apps`.

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/`.

Carrier, B. 2012. The sleuth kit, open source digital forensics. `http://www.sleuthkit.org/`.

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

Evgeni, I. 2012. Fileobserver says its recursive but it is not. `https://code.google.com/p/android/issues/detail?id=33659`.

Freiling, F., Spreitzenbarth, M., and Schmitt, S. 2011. Forensic analysis of smartphones: The Android data extractor lite (adel). In *Proceedings of the Conference on Digital Forensics, Security and Law*. 151–160.

George, N. 2015. Network Ombudsman for Android. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. Advisor: Prabhaker Mateti.

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.

Gudjonsson, K. 2012. log2timeline - plaso. `http://log2timeline.net`.

He, J., Liu, G., Zhao, B., Wan, X., and Huang, N. 2015. Ensuring the authenticity and non misuse of data evidence in digital forensics. *Journal of Harbin Institute of Technology (New Series) 22,* 1.

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

Inc., E. S. 2014. Uninstall it (free), An Android APK. `https://play.google.com/store/apps/details?id=installer.uninstaller&hl=en`.

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

Kamardeen, J. 2015. Auditing Android system for anomalous behavior. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tami Nadu 641112, India. Advisor: Prabhaker Mateti; `http://cecs.wright.edu/~pmateti/GradStudents/index.html`.

Linux Kernel Organization, I. 2016. The Linux Kernel Archives. `https://www.kernel.org/`.

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

Mednieks, Z., Dornin, L., Meike, G. B., and Nakamura, M. 2012. *Programming Android*. O'Reilly Media, Inc.

Mylonas, A. 2008. Smartphone spying tools. *Master's thesis, Royal Holloway, University of London*.

Mylonas, A., Meletiadis, V., Tsoumas, B., Mitrou, L., and Gritzalis, D. 2012. Smartphone forensics: A proactive investigation scheme for evidence acquisition. In *Information Security and Privacy Research*. Springer, 249–260.

PureTalkUSA. 2015. What is a burner phone? https://www.puretalkusa.com/blog/what-is-a-burner-phone/.

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

Shah, C. 2010. Zeus Crimeware Toolkit, McAfee Labs. https://blogs.mcafee.com/mcafee-labs/zeus-crimeware-toolkit/.

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.

Yang, S. J., Choi, J. H., Kim, K. B., and Chang, T. 2015. New acquisition method based on firmware update protocols for Android smartphones. *Digital Investigation 14*, S68–S76.

Zhang, Y., Chen, Z., Xue, H., and Wei, T. 2015. Fingerprints on mobile devices: Abusing and eaking. In *Black Hat Conference*.