Adding Proactive Forensic Support to Android

DISSERTATION

submitted by

AIYYAPPAN P S CB.EN.P2CYS13001

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 2015

Adding Proactive Forensic Support to Android

DISSERTATION

submitted by

AIYYAPPAN P S CB.EN.P2CYS13001

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 2015

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that this dissertation entitled **"ADDING PROACTIVE FORENSIC SUPPORT TO ANDROID"** submitted by **AIYYAPPAN P S, Reg. No. CB.EN.P2CYS13001** 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 EXAMINER

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, COIMBATORE -641 112

TIFAC CORE IN CYBER SECURITY

DECLARATION

I, AIYYAPPAN P S, (Reg. No: CB.EN.P2.CYS13001) hereby declare that this dissertation entitled **"ADDING PROACTIVE FORENSIC SUPPORT TO ANDROID"** 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

ACKNOWLEDGEMENTS

At the very outset, I would like to give the first honors to **Amma, Mata Amritanandamayi Devi** who gave me the wisdom and knowledge to complete this dissertation under her shelter in this esteemed institution.

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 would like to thank **Dr M. Sethumadhavan**, Professor and Head of the Centre for Cyber Security, for giving me useful suggestions and his constant encouragement and guidance throughout the progress of this dissertation.

I express my special thanks to my colleagues 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 and **Mrs. Jevitha K. P**, 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.

Contents

List of Figures

List of Tables

Abbreviations

ADB	**A**ndroid **D**ebug **B**ridge
NDK	**N**ative **D**evelopment **K**it
SDK	**S**oftware **D**evelopment **K**it

Abstract

Advanced features provided by smartphones catch criminal attention. Data from a smart phone is extremely useful in the course of a criminal investigation. By analysing an ordinary mobile device, investigators can collect contacts, call history and SMS. Smart phones contain even more useful information such as social network messages, e-mails, network connections, browser history etc. A smart criminal might modify or clear the traces of activities and the mobile data. This data is harder to acquire once the user deletes it. We propose forensic support to the Android Framework, which monitors all the user activities and stealthily stores it in the cloud. This thesis also proposes a client tool which runs on Android Linux ARM kernel layer that interacts with the adb tool, and also performs imaging, recovery and detailed analysis of device data.

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

Chapter 1

Introduction

The introduction of smartphones has changed the communication landscape drastically. Right from Whatsapp to online banking transactions, students to working professionals, smartphones has become an indispensable part of life. As of 2014, about 85 percent of the smartphone runs on Android. Also Android dominates in the number of application available for users across multiple app stores. In 2014 about 1.3 million apps are active on Google Play, this dominance is because of the popularity of the Andorid among people from all walks of life. Mobile phones are not just for calling and sending messages, they are as powerful as a desktop computer. A high end smart phone can do almost all computing which is done by a normal desktop computer and much more advanced features like NFC, Global Positioning etc. This increased availability of computing power for smartphone also catches the attention of criminals.

1.1 Motivation

Most operating systems are designed without a concern for forensics and Android is no exception. Let us imagine the following scenario. Given that criminals and terrorists use smart phones, what can an Android ROM do? What new components and stealth services could it include? This thesis is an exploration of these questions.

A basic mobile contains the information such as call history, contacts and text message data, but by analysing a smart phone like Android, an investigator can collect more information which includes data such as the information about the places

he travelled, networks he connected, his search history and even his Whatsapp chat history. What happens if he has deleted that information? Basic Forensic methodology fails to get those deleted data. We are including a proactive forensic service that runs on Android ROM such that it tracks the user activities, file system changes and stores it to a cloud server in stealth mode. We also propose client software that runs on Linux machine which connects device through the adb tool, which is responsible for advanced imaging, recovery and analysis of device data.

1.2 Problem Statement

As Android is capturing market, it is becoming favourite target platform of hackers. There are many forensic tools available, these are primarily targeted on applications database files and to extract the deleted data are difficult. Usually data which are used by messaging and mail application stores on encrypted form and to extract these data are hard with the usual forensic tools. Knowledgeable criminals can erase the forensic data from the mobile device. There are Mobile applications like Uninstall-It which can delete all the application data from device. We describe two services designed with the expectation that an Android device is going to be subject to forensic investigation. We leave the task of convincing a suspected criminal to use a forensics-proactive Android device to others.

1.3 Aim

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

1.4 Organization

The thesis is organised to six chapters and Appendix. The thesis discusses the design and implementation of our Forensic Support Module and also the working of user specific data collection tool for Android device. Chapter 2 provides all needed

background knowledge to understand Android forensics. It gives an overview about Android File system and Application data storage. Chapter 3 discusses the architecture, data collection methodology and implementation of our proactive forensic support module as well as the data collection tool. Chapter 4 discusses the related works of the project. Chapter 5 discusses the results and evaluation of the project. Chapter 6 discusses conclusion and future work.

Chapter 2

Background

This section summarizes the background for the forensic work.

2.1 Android File Volumes

Android is based on the Linux operating system. The file systems in Android are the same as in desktop Linux. From version 2.3 android has been using the Ext4 file system. There are kernel modules available for all major file volume designs e.g., vfat, and ntfs. Android devices have six volumes: /boot, /system, /data, /recovery, /cache and /misc.

2.1.1 /boot

The /boot enables the device to boot. This partition contains the kernel and ramdisk module.

2.1.2 /system

Operating System files are present in System partition. Stock applications that are pre installed are also present in this partition.

2.1.3 /data

This is the most important partition from the forensic point of view. All the user data such as user contact, SMS, application files etc. are stored at this location. Factory reset of the device actually wipes this partition.

2.1.4 /recovery

This is specially designed for backup, and considered as an alternative boot partition. This partition lets the device boot into a recovery console for performing advanced recovery and maintenance operations on it.

2.1.5 /cache

Application cache is stored at this partition. This partition is also important for forensic purpose.

2.1.6 /misc

This portion has setting files like USB and hardware configuration. Some features of the device do not work if the partition got corrupted.

There is a special directory, /storage, with mount points within it, to mount internal and external sdcards. Along with these partitions, Android supports pseudo file systems: procfs and sysfs that contain the kernel state and detailed information about hardware and running processes.

2.1.7 / proc

/proc contain the kernel state and some special files. The detailed information about processes running and hardware details can be found on this directory.

/proc contain the kernel state and some special files. The detailed information about processes running and hardware details can be found on this directory. Below are the contents of /proc directory

cpuinfo - Information about processor related parameters
crypto - Cryptographic ciphers which are used by the Linux Kernel
devices - Lists the devices configured in the system
filesystems - List of file system supported by the kernel
modules - List of all modules loaded into the kernel
mounts - a list of all mounts in use by the system
partitions - This file contains partition block allocation information
pid - Is a directory named by the process ID. That contains the information about that process

2.2 Stealth File Volumes

There are several methods to hide data partition Linux disk encryption [Torvalds and Others (2015)] and various file volume designs (e.g., Peters et al. (2015)) that are stealthy, i. e., 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. DEFY

2.3 Data Storage in Android

Android applications get a dedicated subdirectories to store its their private data. Androids sandboxing makes sure each application is assigned a difier- ent user-id and no other application to has access to that data. Files stored in storage can be accessed by any application.

2.3.1 SharedPreferences

If an APK needs to store just few items, they use Shared Preferences as name-value pairs stored in an XML file.

2.3.2 SQLite Database

Application specific data are usually stored on a relational database and Android supports SQLite database for the same.

Database File	**Location**
Phonebook Contact	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 Info	com.android.providers.calendar/databases/calendar.db
SMS	com.android.providers.telephony/databases/mmssms.db

TABLE 2.1: SQLite Databases (inside the **/data/data/** directory)

These are the path of major instant messenger databases.
Whatsapp: /data/data/com.whatsapp/databases/msgstore.db
Viber: /data/data/com.viber.voip/databases/viber messages.db
Facebook: /data/data/com.facebook.katana/databases/

2.4 Application Sand-boxing

Each application gets a dedicated part of the File-system in which they can write its private data. Since each application has different user id, no application can access another app data. Android Application uses SQLite database to store its data.

2.5 Noticing File and Directory Changes

In this section we discusses about those file monitoring tools supported in Linux as well as Android. Linux support many file monitoring tools; inotify, incorn, iwatch, lsyncd are some of them. But in android suppots notify command and FileObserver class.

2.5.1 FileObserver

Android provides an API for monitoring file events which is FileObserver. It is based on inotify file change notification system in Linux. It is an abstract class,

Events	**Constant Value**
ACCESS	1
ALL_EVENTS	4095
ATTRIB	4
CLOSE_NOWRITE	16
CLOSE_WRITE	8
CREATE	256
DELETE	512
DELETE_SELF	1024
MODIFY	2
MOVED_FROM	64
MOVED_TO	128
MOVE_SELF	2048

TABLE 2.2: File Observer Events and corresponding Event value

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 sub-folders are not.

2.5.2 notify command

This command uses the inotify system call an API to monitor directories or files for modifications. Similar to FileObserver, it is also not recursive.

2.5.3 inotify tool

Inotify is a ligh weight file monitoring system in the Linux. It supports the command inotifywait, which waits for a change to file or folder associated with it. Below command recursively check for create, modify and delete events on /home directory.

2.5.4 Other file monitoring tools

There are many tools available in Linux for file monitoring File Alteration Monitor, dnotify, incorn are some of them. File Alteration Monitor, also known as FAM subsystem allows applications to watch certain files and be notified when they are modified. It enables applications to work on a greater variety of platforms. During the creation of a large number of files it slows down the entire system, using many

CPU cycles. The dnotify is the preceder of inotify and was introduced in the 2.4 kernel series. Because of performance issue it has been obsolete by inotify. Incorn is cron-like daemon. Unlike the other cron daemons which execute an action on a specific time, incorn triggers an action on the output of inotify event. The setup is simple and straight forward. As a first step just put the name of the user who should be allowed to configure incron jobs to the file /etc/incron.allow. Invoking "incrontab -e" will pop up an editor and we can insert the rules.

Chapter 3

Proposed System

We propose adding a forensic support service to the Android ROM we are building [Mateti et al. (2015)]. This service proactively identifies, prioritizes, collects and stealthily stores securely forensic relevant data initially on a stealth file system within the device, that is opportunistically uploaded to a cloud server.

This section is an overview of the architecture (Figure 3.1). The next section describes a design and an initial implementation.

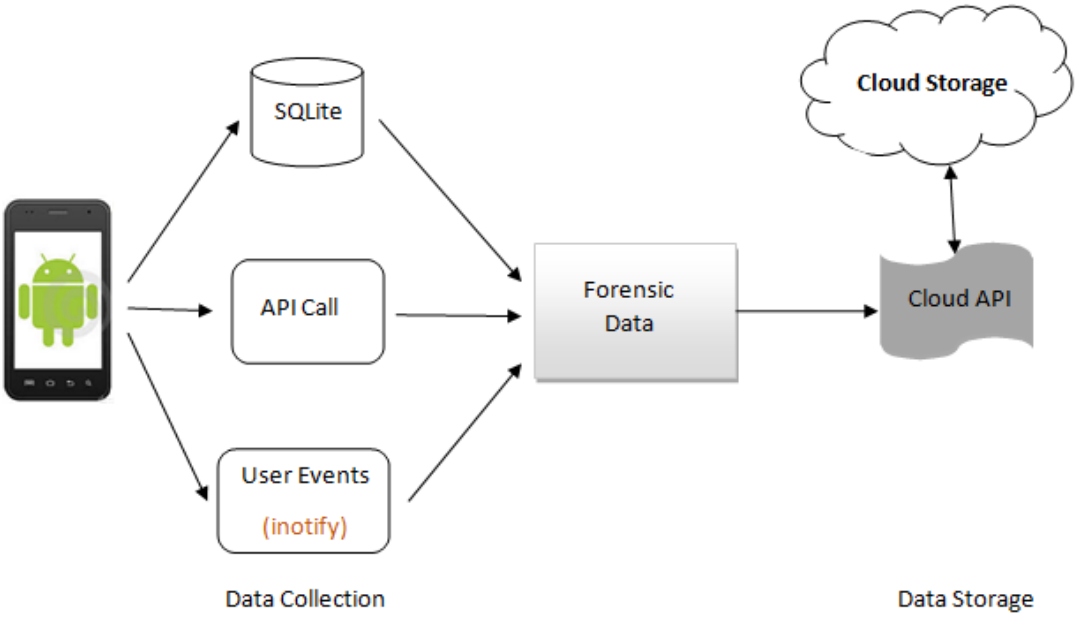

FIGURE 3.1: Proactive Forensic Support Architecture

The project has two phases; the forensic module resides in the device and the other is Linux tool for extracting device/cloud data. The details of forensic extractor are discussed in the (Section 3.5).

User Data	**API**
Phonebook Contact	android.provider.ContactsContract.Common DataKinds.Phone.CONTENT_URI
Call History	android.provider.CallLog.Calls.CONTENT_URI
Browsing History	android.provider.Browser.BOOKMARKS_URI
Browser Searches	android.provider.Browser.SEARCHES_URI
Pictures Added	android.provider.MediaStore.Images.Media. EXTERNAL_CONTENT_URI
GPS Location	android.permission.ACCESS_FINE_LOCATION

TABLE 3.1: Android APIs used by Forensic Module

3.1 Architecture of Forensic module

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

3.1.1 Data Collection Phase

Android application stores its private data using **SQLite database** and these files are frequently changing. For example, phone logs are stored in the `contacts2.db` database file and the data are more relevant in the case of forensic purpose. Forensic module collects these SQLite file in a scheduled manner so that no data should lose when investigating. User activities like camera usage, network usage and GPS data are collected by using Android provided **APIs**. Some of the Android APIs are in the below.

Filesystem events get captured by using a tweeked version of **inotify tool**. More details about inotify is discussed in the (Section 2.5.3).

3.1.2 Data Storage Phase

The collected data gets stored on the device partition until it gets connected to the Intrenet. 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.

3.2 Defining Forensic Relevant Data

This system service is capable of identifying and logging exhaustively the user interactions with the device. It includes the phone calls, SMS incoming and outgoing, camera events, browser history, GPS locations, Social media messages etc. This helps the investigator to find the information of the suspect like the places he travelled, his friends, networks he connected, his search history etc.

A setup dialog (Figure 3.2) during the initial flashing of the ROM defines what items are to be collected/ ignored, and stored. Care is taken that this settings file is hidden. This dialog is then destroyed, never to show up again until the device is reflashed.

The data are classified according to priority. There are only two priorities, Critical and Less. Critical data must be present while investing the device. So this data gets uploaded to Cloud first because of the risk that the partition may get full. If there is scarcity of space the module itself will delete Non Critical data first. Data like Phone contacts, Chat history, GPS location, Network log etc. are considered as Critical data, System updates, APK files etc. are less critical ones.

Data according to priority are stored in to an xml file and the opportunistic upload API takes data from this xml file to store. The data once uploaded is deleted from forensic partition in the device.

3.3 Ongoing/ 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.

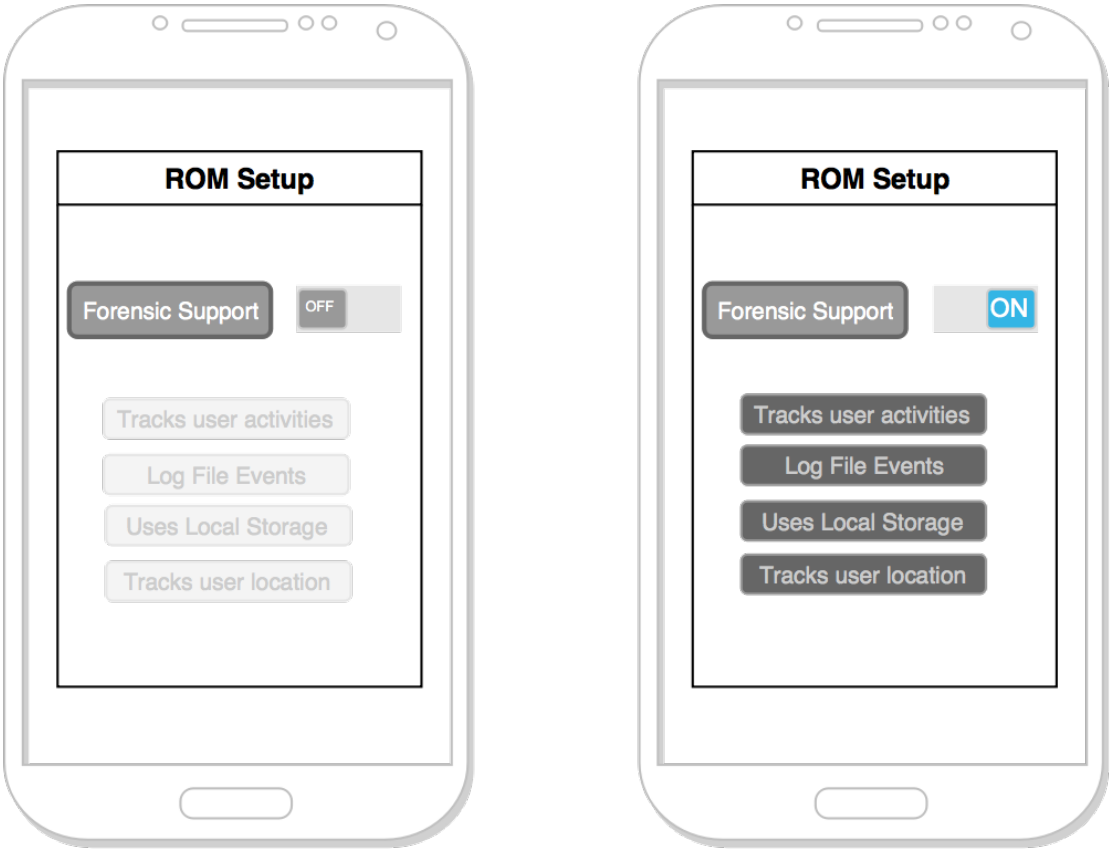

FIGURE 3.2: ROM Setup Dialog

3.4 Opportunistic Upload of "Big Data"

Forensic data can pile up to be very large (in 100s of GB). When the device gets connected to the Internet, the data is stealthily moved to a cloud server. Once the data is uploaded to the cloud, it is cleared from the device.

The data upload is only done if the device is fully charged or the device is on chaging mode. The device uppload is only done on WiFi session and if the forensic module uses the mobile data, the user will get concerned in that. The battery as well as the processor usage gets the user atension, so the cloud upload is only done when the device is idle.

3.5 Forensic Examiner on Desktop Linux

We also provide a client tool which runs on a Linux machine. The Forensic Examiner does advanced imaging of Android partitions, recovery and analysis of

forensic data. It interacts with the device through `adb`, the Android Debug Bridge (`http://developer.android.com/tools/help/adb.html`) tool. This Linux program not only images the device but also interacts with cloud data.

Imaging of data include the disk dump (`dd`) of the whole device and the data dump from the cloud storage. Device specific data from cloud is identified by the device IMEI id.

The extractor tool gets the image file and mount the partitioin on the Linux machine. This tool contain shell script and takes some code from the open source code of Android Free Forensic Toolkit.

3.5.1 Real Time Analysis

Most of the android application encrypts its data and keys. Forensic data can be collected only by analysing or decrypting those application data. No real time analysis is done on the Android device. The primary reason for this design choice is that the resulting lag and battery consumption will reveal the presence of "something."

The data are can be collected even the physical presence of the device is not available. Location of the device can also be tracked. The data uploaded are tracked with the date of uploading so the provenance of the device data can be found by the investigator.

3.6 Stealth Challenges

Since the forensic module causes additional device storage, internet connection and some amount of processor time. There should be a sting way to do these tasks to hide these activities from user. Main challenge is to hide the partition itself another is to hide the forensic process from process listing.

Cloud upload is also a challenging task and this has to be done in an oppurtunistic way so that the user must not concern on his/her mobile data,

Stealth That the device has forensic support enabled should not be obvious to the "owner" of the device. 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 dealing with `proc` and `sysfs` need to be modified, following the techniques of rootkits, to enable the hiding of service processes. This is done by using `hidepid` command.

Encryption Data collected by the forensic service is encrypted and is stored in a stealth forensic partition. (See Section 5.4.)

Opportunistic Uploads Users are more concerned about their mobile data usage rather than WiFi network usage. So, forensic data is uploaded on WiFi unless there is a risk of making device storage 100% full.

3.7 Hiding Process

There are many process listing commands as well as various APKs to list running processes. These tools uses procfs filesystem to get these data. The forensic process is hidden from user by using hidepid option. Figure(Figure 3.3) shows that the forensic process is hidden from OS Monitor APK.

The values are as follows:

`hidepid=0` - This is the default option and anybody may read all world-readable `/proc/PID/*` files.

`hidepid=1` - Users may not access any `/proc/PID/` directories, but their own.

`hidepid=2` - It means the same effect of `hidepid=1` with all `/proc/PID/` will be invisible to other users.

3.8 Android NDK Programming

The NDK is a toolset that allows implementing parts of the android applications use native-code languages such as C and C++. Typically, good use cases for the NDK are CPU-intensive applications such as game engines, signal processing, and physics simulation. By compiling native code by using NDK, a shared object code (.so) gets created and android application or android source can access this shared object library.

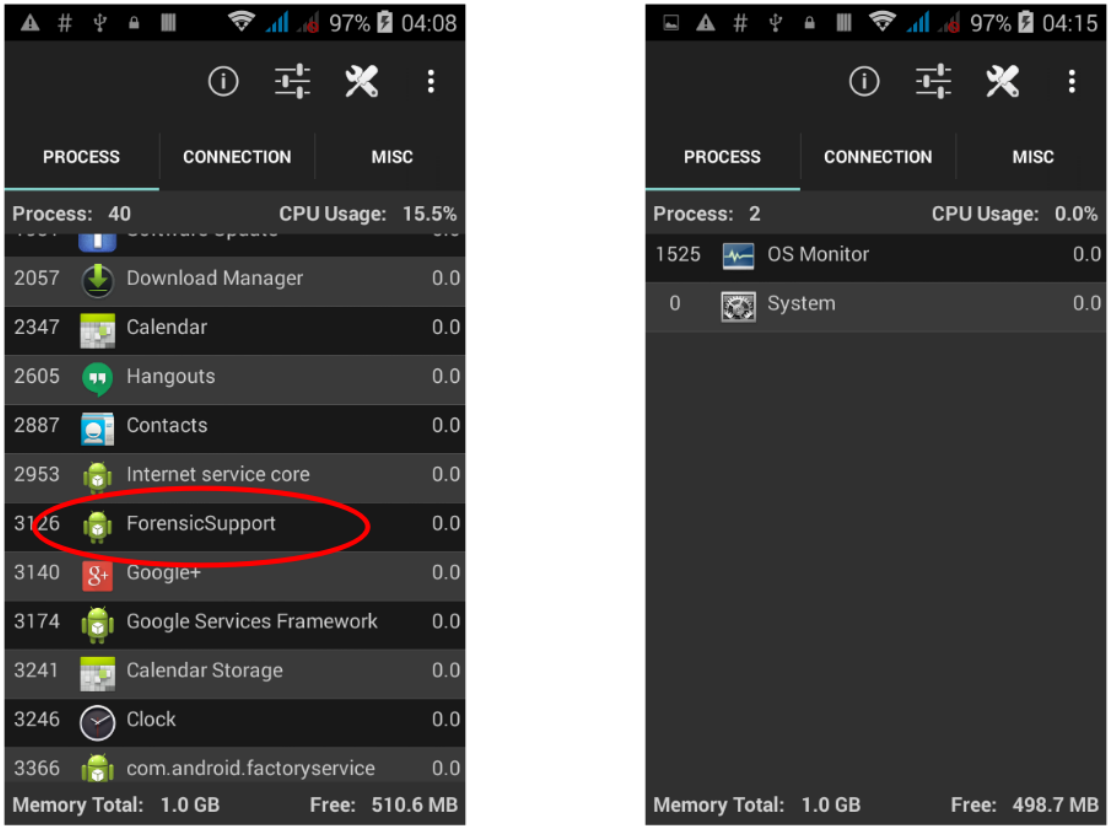

FIGURE 3.3: Forensic Process: OSMonitor Screenshot

Chapter 4

Related Work

This section describes two companion projects, and related Android Forensics projects.

4.1 Ethics

What is described is a sting operation. The ethics of deploying this technique on previously known criminals is well discussed in the book Encyclopedia of Criminal Justice Ethics [Arrigo (2014)]. If the forensic enabled device is used by a criminally-innocent but un-authorized individual it is surely a case of privacy issue and risk to the security of public. Tracking a GPS device without the user's knowledge need court-issued search warrant.

4.2 Our Companion Projects

We have two companion projects to make an Android ROM with some additional security features. One is Network Ombudsman [George (2015)] for Android, which is a network monitoring and filtering tool, it monitors and logs incoming and outgoing data traffic, provide real-time monitoring of all the process and network connections, provides context based dynamic filtering rules and facility for incident investigation by querying the uploaded logs in the cloud server. The other project is Auditing Android System for Anomalous Behaviour [Kamardeen (2015)], which is a Framework level audit service to Android. It logs all activities of system

and infers malicious activities from the log and identifies the process or app that triggered it. Both projects along with the forensic module connect to a Cloud server and stores the logs.

4.3 Android Forensics Elsewhere

The field of forensics of smart phones is active. We can classify them as being reactive [Mahajan, Dahiya, and Sanghvi (Mahajan et al.)] [CyberPunk (2015)] or proactive forensic methodologies [Freiling et al. (2011)] [Wright (2012)] [Grovera (2013)].

4.3.1 Forensic Analysis of Smartphones: ADEL

Forensic analysis of smartphones introduces a tool, ADEL, which extracts, analyse and report the SQLite data.

Software-based Approach

Two types of software based approaches are introduced in this paper, one is a software agent and other is using adb. Agent based approach has a software agent resides in Android device, that collect and analyse 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.

Using adb:

This method uses SDK and dump the data to investigators machine by using adb. Android devices in production mode deny the access to databases via the adb. Rooted devices has the option 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.

Hardware-based Approach:

A hardware based approach is also introduced which uses desoldering of chips to get the data back. By this method memory chips removed and is desoldered and analyse the chip using special hardwares.

In **Desoldering Memory Chips** method, chip is analysed using special hardware called PC-3000 Flash, there is no intermediate layer so forensic data is preserved. This method posses certain risk, since there is risk in removing and desoldering.

SQLite DB File format:

Database engine is an integral part of Android applications. All Android applications including stock appications such as contacts, GPS, call list, SMS, etc. store its data in SQLite database format.

- 100 bytes of DB header is there and stores general data about the database.
- Each table is internally represented as b-tree structure. First page contains sql-master table which stores the DB schema

ADEL (THE ANDROID DATA EXTRACTOR LITE):

ADEL automatically dump selected SQLite database files from Android devices and extract the contents stored within the dumped files.

- ADEL makes use of the Android Software Development Kit (Android SDK) to dump database files to the investigator's machine.
- ADEL reads the database header (first 100 bytes of the file) and extracts the values for each of the header fields. Complete b-tree structure is parsed for each table.

Reporting:

Telephone and SIM-card information (e. g. IMSI and serial number), telephone book and call lists, calendar entries, SMS messages are extracted. Data retrieved this way is written to an XML-File.

4.3.2 Analysis of WhatsApp Forensics in Android Smartphones

Analysis of WhatsApp Forensics in Android Smartphones focuses on conducting forensic data analysis by extracting useful information from WhatsApp.

- 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. With the starting version of WhatsApp 2.9 the messages exchanged was stored in 'msgstore.db' which is 'SQLite' databases.
- Now WhatsApp database encryption having custom AES encryption algorithm with above 192-bit encryption key mainly used for WhatsApp Android Platform.
- So now the previous file textbfmsgstore.db is converted to textbfmsgstore.db.crypt. (/sdcard/WhatsApp/Databases/msgstore.db.crypt)
- Major problem after having the file msgstore.db.crypt is its decryption.
- WhatsApp Xtract tool decrypt and organize SQLite database files in an organized HTML form. A python script uses this same key to decrypt the encrypted db file and presents the result in a well organised HTML page.

4.3.3 Symantec Smartphone Honey Stick Project

Experiment involving 50 lost smart phones, before the smartphones were intentionally lost. A collection of simulated corporate and personal data was placed on them and devices are remotely monitored. The data collected for apps on each device included: Device ID, App name and Time of app activation.

- A GPS tracking mechanism was also used to log each phone's position occasionally.
- Detailed finding was done on percent people accessed by the finders of the devices.
- Who accessed for personal related apps and information and accessed for both business and personal information

4.3.4 Android Forensics: Automated data collection and reporting from a mobile device

Automated data collection and reporting from a mobile device introduces DroidWatch an automated system prototype composed of an Android application and an enterprise server.

- DroidWatch continuously collects, stores, and transfers forensically valuable Android data to a remote Web server without root privileges.
- All collected data is stored temporarily in a local SQLite database on the phone and is configured to be accessible to the DroidWatch app only.
- A scheduled alarm periodically transfers the local SQLite database file over hypertext transfer protocol secure (HTTPS) POST to the enterprise server for processing.
- An Internet history event includes the action taken (e.g., browse or search), the search term or URL, the event time, and the attributed device ID.
- An audit for installed apps may reveal that a device has malware or other apps of concern. This would warrant additional concerns and security measures during an internal investigation.

4.3.5 Deniable, Encrypted File System for Log-Structured Storage

Peters et al. (2015) introduced a stealth file system, DEFY, a deniable, encrypted filesystem for log structured storage.

- It is a deniable file system that is specifically designed for flash memory.
- It is targeted for mobile file system.
- Implementation was done on Linux kernel module based on previous work in both YAFFS and WhisperYaffs.
- The physical properties of solid-state memory require wear leveling and disallow in-place updates, motivating our use of a log-structured file system.

- DEFY also supports other features useful in a mobile setting, including authenticated encryption and fine-grained secure deletion of data.
- Similar to WhisperYaffs, DEFY encrypts each chunk including the OOB areas resulting in a full disk encryption. DEFY has the ability to obfuscate the existance of data.

Some of these projects fail to collect encrypted data [Wright (2012)] and some fail to retrieve deleted data [CyberPunk (2015)]. Proactive forensic agents DroidWatch [Grovera (2013)], ADEL [Freiling et al. (2011)], Honey Stick Project [Wright (2012)] fail in tracking device activities in stealth mode. Our project can collect encrypted as well as deleted data from device and it stores them in a stealthy way.

Chapter 5

Results and Discussion

In this section, we discuss about the results and impact of the proactive forensics service.

5.1 Status of Proposed System

The forensic service obtains data in three ways. One is from Android APIs, which collects data such as Phone logs, SMS logs, Camera events and GPS data. Second is from inotify tool, which collects file system events and third is from SQLite database files. SQLite database is used to store all social media messages, browser history, Phone Contacts. The implementation and testing of the Android Forensic Service was done in two stages. First, we staged it on Desktop Linux. Second, we made an APK that can run the service on rooted devices.

5.1.1 Detecting File Changes

There are various file monitoring tools, e.g., Incron, File Alteration Monitor [FAM (2015)], Linux audit, inotify. 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/aiya` for the events of creation and modification, while logging those events to a file.

```
inotifywait -mr /home/aiya/ -e create -e modify -e close_write
```

```
-timefmt '%H:%M' -format '%f %w %T %e'
```

Our forensic service works as an Android system service at the Android Framework Layer. This uses `inotify NDK` library [Google.com (2015)] and tracks all the system changes. This tweaked inotify NDK code copies the files to a forensic partition. An XML hidden file stores the path names of all copied files along. The service checks for WiFi connection and once the device gets connected to the Internet, the files get uploaded.

5.1.1.1 Porting inotifwait to 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 (see Table 2.2).

5.1.1.2 Track user activities

The user events such as Camera events, GPS Location change, SMS and Call events are tracked by using broadcast reciever and all those logs are stored to a log file in Forensic partition.

5.1.2 Forensic Examiner Tool

We provide a client tool which runs on Linux machine to image, recover and collect device specific data from the Cloud. The extractor tool has shell scripts to dump the whole device data to the investigator's machine. The tool uses part of AFFT [CyberPunk (2015)] code. It has options to get data from cloud and retrieve all the collected data. Extractor tool is available on `https://github.com/psaiyappan/Forensic-Examiner-Tool.git`.

Cloud Upload

The forensic module along with the data from companion projects [Kamardeen (2015)] and [George (2015)] uses cloud storage to store its data. Device keeps the

```
aiyyappan@MainStage:~/Extractor$ ./tool.sh
Android Forensic - Extractor
Enter IMEI Number: 911361234567890
1) Retrieve whole system image
2) Obtain Cloud data
3) Collect forensic data
4) Exit
Enter your choice: 1
Imaging device . . .
Here are the list of partitions on the device
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
major minor  #blocks  name
   7        0      19312 loop0
 253        0     524288 zram0
 179        0    7609856 mmcblk0
 179        1          1 mmcblk0p1
 179        2      10240 mmcblk0p2
 179        3      10240 mmcblk0p3
 179        4    1048576 mmcblk0p4
 179        5     231424 mmcblk0p5
 179        6    1355776 mmcblk0p6
 179        7    4917504 mmcblk0p7
 179       64       4096 mmcblk0boot1
 179       32       4096 mmcblk0boot0


Imaging Done.
```

FIGURE 5.1: Forensic Examiner Tool

data in Stealth File Volume until it gets uploaded to the Cloud, data gets uploaded according to the priority assigned to it.

5.2 Stealth Enablers

The forensic service is done in a stealthy way. The forensic service process and cloud update are done on without catching user attention. Processes listing commands like `ps`, `top` etc get details from `/proc` virtual file system. By using `hidepid` command forensic processes and its information are hidden from other APKs to track. Upload to the cloud is done opportunistically. Since users pay attention to their mobile data, cloud upload is done during WiFi sessions. Data

```
Files can be found in /home/aiyyappan/Extractor/911361234567891
1) Retrieve whole system image
2) Obtain Cloud data
3) Collect forensic data
4) Exit
Enter your choice
2
Downloading data from Cloud . . .
Retrieved data
        Name            Uploaded_date   Size(KB)
        20150601_1      Jun 01 07:10            1430
        20150601_2      Jun 01 10:25            20
        20150601_3      Jun 01 18:19            175
        20150602_1      Jun 02 08:16            178
        20150602_2      Jun 02 21:44            114
        20150604_1      Jun 04 11:04            3565
        20150604_2      Jun 04 20:22            1097
```

FIGURE 5.2: Screenshot: Cloud Stored Data as Shown by Forensic Examiner Tool

upload uses considerable amount of battery charge. So this is done once the device is being charged or fully charged.

5.3 Hide Forensic Process

Process list command ps uses /proc file system to get processes details. Files status, stat and cmdline in /proc/PID/ contain the information about each process. Editing ps binary wont be a good solution to hide a processes, but hiding folder /proc/PID/ or denying access to it will be an ideal method. The command `hidepid` hides processes and its information to other users, it accepts 3 values. Default is hidepid=0, where all sttings 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 Borello (2014) provides a provision to load shared library before normal system libraries are loaded. The following source

code overrides libc's readdir function, every time the code see that the /proc/PID (PID for the specified process) just block that access.

5.4 Stealth File Volume

There is a stealth partition owned by root. Non root applications cannot detect that partition or the data it contains. On a future enhancement, we include an encrypted file system, so that even the APK with root privileges cannot detect it.

5.5 Impact on Battery Consumption

Connecting to internet affects the battery consumption due to process running in the background. Battery consumption will be unnoticeable as only WiFi connection is used for uploading forensic data and that too when device is charging.

5.6 Contribution to Lag

The file monitoring is done by the light weight native code library: inotify, so this activity does not cause any lag to the system. The lag caused by cloud upload is less visible to the user which is achieved by doing this task while the device is idle.

5.7 Local Storage

Forensic data is stored separately in the partition `'/forensic'` and it is hidden from the users. 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. It is done on priority basis. If the device does not connect to the internet the data with less priority gets deleted.

5.8 How Stealthy Is It?

Forensic service collects data, stores it locally and uploads it to cloud in a stealthy way. Forensic partition is hidden from non-root applications. Processes that do the respective tasks are hidden, so the process listing commands like `top`, `ps` and the APKs like OSMonitor cannot find them.

5.8.1 Hide from Antivirus

Antivirus software runs in the background on the device, checking every file in the system. Basically all antivirus software has a database of known viruses and signatures. They check the program first, comparing it to known viruses, worms, and other types of malware. It does a "heuristic" checking, analysing programs for types of bad behaviour that may indicate a new, unknown virus. Additionally it monitors the processes running on the system, ensuring that no malicious processes are running. Our forensic module runs as an android framework service and uploads collected data to the cloud in a stealthy way; we have stealth enablers to hide the processes running and to do opportunistic upload. Antivirus and anti-spywares scans for existing malware signatures in the device and all incoming network data. Since the forensic module and cloud API runs in the framework layer and runs as a stealth service existing antivirus software cannot detect the existence of them.

5.8.2 Differentiation with Spyware

Spyware by definition is software that aims to gather information about a person or organization without their knowledge and that may send such information to another entity or that asserts control over a device without the consumer's knowledge. Our proactive forensic support service is a spyware in that sense, but it is for investigative purpose. Section 4.1 discusses about the ethics to be followed to add such kind of tracking mechanism to a user device. Only authorised person or organisation should deploy the tracking mechanism to a suspect, if not it is a criminal offence.

Chapter 6

Conclusion and Future Work

Forensic analysis of smartphone is extremely useful during the course of criminal investigations. Mobile devices have a huge collection of personal data that can help the investigators to track the activities of a suspect. These data include the call details, location history, messages, browser history etc. The data in the device can be used to fingerprint the individual. Two services are introduced in this thesis, which does forensic data collection in a suspect‘s device. One is a data collection module that stealthily collects all the user activities and stores it to cloud storage. Other is a client tool that communicates with the device through Android debug bridge tool for carrying out image, recovery and detailed analysis of the forensic data. This data extractor tool can dump data from device but not from Cloud storage. As a future expansion extraction of data from the cloud according to the unique ID of the device (e.g. IMEI Number) has to be done. The Extractor tool also could create reports from the collected data. Data can be filtered according to the user activity such as the Phone logs, SMS or Social media messaged collected and reports can be generated to depict them. A real time monitoring without the device can also be done as a future enhancement, i.e. if the physical presence of the device is not available still the data can be collected from cloud server, which can be used to track activities of a suspect. Even though many forensic tools and techniques are available in the market, most of these tools fail to prove a user activity because of the lack of information collected. The deleted and encrypted data are hard to recover once the user forcefully does it. Our system collects and store all the user activities and stores stealthy to the central cloud server.

Chapter 7

Appendix A

7.1 Linux Implementation

This monitors events of /home/user/foo directory

Linux Implementation: Code

```
#!/bin/bash
 while read line
 do

 File=$(echo $line|cut -d ' ' -f1)
 Loc=$(echo $line|cut -d ' ' -f2)
 Time=$(echo $line|cut -d ' ' -f3)
 Task=$(echo $line|cut -d ' ' -f4)

 #Displays the File which is modified/Created
 echo File is $File Modified on Time at Location $Loc

 #Logs all the events modified/Created on /home/aiya/ on log.txt
 echo File is $File Modified on Time at Location $Loc >> log.txt
```

Linux Implementation: typescript

```
root@aiya-VirtualBox:/home/aiya/Desktop/Commands# ./MyBash
```

```
Setting up watches. Beware: since -r was given, this may take a while!
Watches established.
File Mydata.txt Modified on Time 05:10
File Mydata.txt Modified on Time 05:15
File Mydata.txt Modified on Time 05:17
```

7.2 Android Implementation

We implemented the same on an android device as well. This was done by using APIs provided by Android and NDK programming.

7.2.1 Using Android APIs

Android provides API support which used inotify using FileObserver and Notify Command.
FileObserver Syntax:
`FileObserver(String path, int mask)`

Parameters

`path` The file or directory to monitor
`mask` The event or events (added together) to watch

7.2.2 Using NDK

Implemented "inotifywait" by using Android NDK. This can monitor all the events like file access, create modify, delete inside all the subfolders.

Once a file or folder is accessed the event is captured and logged. Following are the screenshots of Logcat outputs.

Once a file or folder is accessed the event is captured and logged. Following are the screenshots of Logcat outputs.

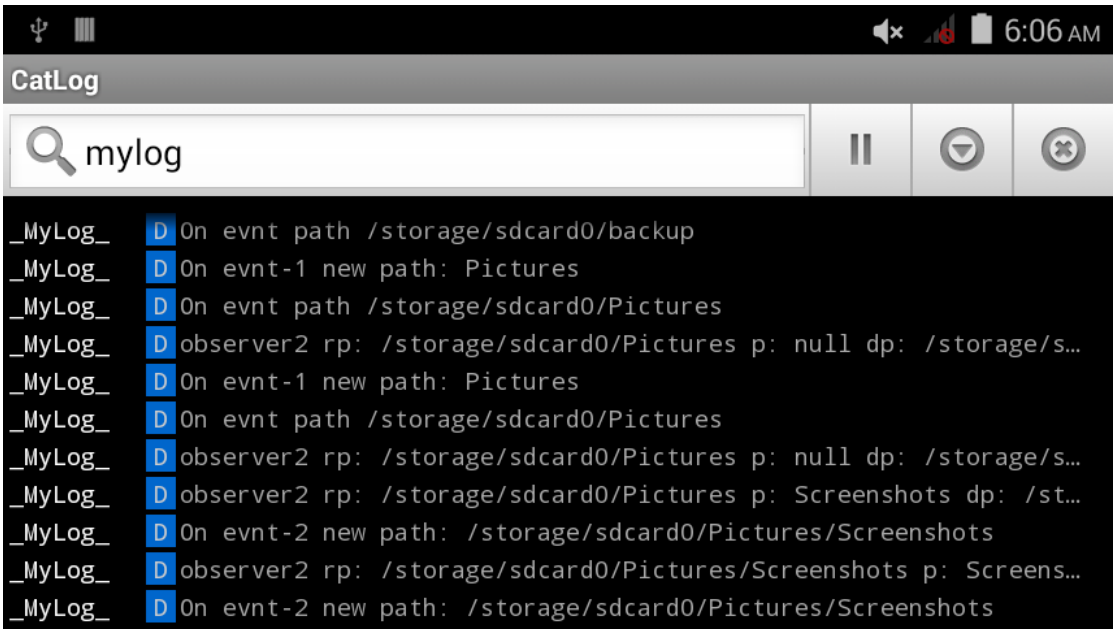

FIGURE 7.1: FileObserver Events

- `cat tmp.txt`

Events - Access, CloseOnWrite and Open

```
MyApp          AFSF: file event 32 @ /sdcard/Downloads/ of file tmp.txt
MyApp          AFSF: file event 1 @ /sdcard/Downloads/ of file tmp.txt
MyApp          AFSF: file event 16 @ /sdcard/Downloads/ of file tmp.txt
```

- `mv tmp.txt some-text.txt`

Events - Modify, CloseWrite, Create, CloseOnWrite and Open

```
MyApp          AFSF: file event 64 @ /sdcard/Downloads/ of file tmp.txt
MyApp          AFSF: file event 128 @ /sdcard/Downloads/ of file some-text.txt
```

- `cp a.txt hello.txt`

Events - Modify, CloseWrite, Create, CloseOnWrite and Open

```
MyApp          AFSF: file event 32 @ storage/sdcard0/fileXchg/sample/ of file a.tx ⏎
               t
MyApp          AFSF: file event 256 @ storage/sdcard0/fileXchg/sample/ of file hel ⏎
               lo.txt
MyApp          AFSF: file event 32 @ storage/sdcard0/fileXchg/sample/ of file hell ⏎
               o.txt
MyApp          AFSF: file event 2 @ storage/sdcard0/fileXchg/sample/ of file hello ⏎
               .txt
MyApp          AFSF: file event 16 @ storage/sdcard0/fileXchg/sample/ of file a.tx ⏎
               t
MyApp          AFSF: file event 8 @ storage/sdcard0/fileXchg/sample/ of file hello ⏎
```

- `rm hello.txt`

Events - Delete

```
MyApp          AFSF: file event 512 @ storage/sdcard0/fileXchg/sample/ of file hel ⏎
```

7.3 Sample Process List

Forensic processes are hidden from user and commands like ps, top will not list those processes. Below is a sample list of process.

```
aiyyappan@MainStage:~$ adb shell
shell@A102:/ $ ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
shell     159   1     1044   368   c00600e8 b6f23094 S
/system/bin/batterywarning
shell     354   1     2748   844   ffffffff b6ed0568 S /system/bin/mdlogger
shell     3691  188   1040   520   c025cee8 b6f354d8 S /system/bin/sh
shell     3736  188   1036   504   c025cee8 b6eff4d8 S /system/bin/sh
shell     3744  188   1036   504   c0011a3c b6e7d320 S /system/bin/sh
shell     3813  188   1036   504   c0011a3c b6efb320 S /system/bin/sh
shell@A102:/
```

7.4 PHP Code

Cloud uses php POST method to get data from device. Below is the php code for receving POST variable.

<?php

```
$file = $_POST["file"];
$name = $_POST["name"];
$IMEI = $_POST["IIMEI"];

authoriseDevice($IMEI);

$decodedFile = base64_decode("$file");
$todayh = getdate($WeekMon);

$d = $todayh[mday];
$m = $todayh[mon];
$y = $todayh[year];
$FolderName =  "$d-$m-$y"; //getdate converted day
file_put_contents($FolderName.".file",$decodedFile);

?>
```

7.5 Forensic Extractor Code

Forensic Extractor Code uses shell script code to extract data from device. Below is the script for imaging device.

```
#!/bin/bash

adb forward tcp:5555 tcp:5555
adb shell 'cat /proc/partitions' # gives the user a
 list of drives/partitions on the Android device
echo ""
echo "Here is a list of partitions on the Android device.
 You can image individual partitions or you can image
```

```
 the entire drive. To image the entire drive, provide the
 name of the device that has the largest size."
function pickdevice () {
echo "What device should be imaged?"
read device # gets the name of the desired device from the user
device=$(echo $device | sed 's/\/dev//')
}
pickdevice
adb shell "su -c \"/system/xbin/busybox nc -l -p 5555 -e
 /system/xbin/busybox dd if=/dev/block/$device\"" &
/opt/afft/imaging/step2.sh
 $1 # executes the imaging command on the device,
and goes to the next part on the PC

#!/bin/bash
echo ""
echo "Imaging command sent"
echo "Giving device 10 seconds to process command"
sleep 10 # ADB needs time to connect to the device and send
the command before we get things set up on the PC
currentDir=$(pwd)
adb forward tcp:5555 tcp:5555 # sets up connection from which
we recieve the data stream
cd $1
nc 127.0.0.1 5555 | pv -i 0.5 > image.dd # obtains the dd image
 from the device via aforementioned connection
mkdir ../mount
gdisk -l image.dd > partitioninfo # gets partition info from the image.
 We'll need this for mounting the files later
cat ./partitioninfo | awk 'f;/Number/{f=1}' > tmpdata # strips the header
from the data, makes parsing via script much easier.
cat ./tmpdata | awk '{print $2, $7}' > ../mount/mountinfo # alters the
patition info file to include info only useful to the mounting script
cd $currentDir
echo "Done! Is the user data partition called 'userdata'? If not,
type in the name"
#gets the name of the userdata partition, incase it isn't the default
```

```
read userdata
if [ -z $userdata ]; then
userdata="$1/mount/userdata"
fi
echo $userdata > $1/userdata-name.txt
```

7.6 NDK Code

Foensic module uses inotify NDK code to track file events.

NDK Make file:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
LOCAL_C_INCLUDES := \
./jni/regex \
./jni/libinotifytools

LOCAL_MODULE    := inotifywait
LOCAL_SRC_FILES := wrap.c common.c \
libinotifytools/inotifytools.c libinotifytools/redblack.c
LOCAL_LDLIBS := -llog
include $(BUILD_SHARED_LIBRARY)
```

NDK Native Library file:

```
#include <jni.h>
/* Header for class com_forensicsupport_NativeLib */

#ifndef _Included_com_forensicsupport_NativeLib
#define _Included_com_forensicsupport_NativeLib
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_forensicsupport_NativeLib
```

```
 * Method:    add
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_com_forensicsupport_NativeLib_add
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     com_forensicsupport_NativeLib
 * Method:    hello
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_forensicsupport_NativeLib_hello
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Bibliography

Arrigo, B. A. 2014. Encyclopaedia of criminal justice ethics. In *Encyclopedia of Criminal Justice Ethics*. SAGE Publications, 912–913.

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

CyberPunk. 2015. *Android Free Forensic Toolkit*. http://n0where.net/android-free-forensic-toolkit.

FAM. 2015. *File Alteration Monitor*. http://oss.sgi.com/projects/fam/.

Freiling, F., Spreitzenbarth, M., and Schmitt, S. 2011. Forensic analysis of smartphones: The Android data extractor lite (ADEL). 151–160.

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

Google.com. 2015. *Android NDK: Android Developers*. https://developer.android.com/tools/sdk/ndk/index.html.

Grovera, J. 2013. Android forensics: Automated data collection and reporting from a mobile device. *Digital Investigation*, S12–S20. The Proceedings of the Thirteenth Annual Digital Forensics Research (DFRWS) Conference.

Kamardeen, J. 2015. Auditing Android system for anomalous behavior. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tami Nadu 641112, India.

Mahajan, A., Dahiya, M., and Sanghvi, H. Forensic Analysis of Instant Messenger Applications on android devices. *International Journal of Computer Applications 68*, 8. http://research.ijcaonline.org/volume68/number8/pxc3886965.pdf.

Mateti, P., Aiyyappan, P., George, N., Kamardeen, J., Sahadevan, A. K., and Shetti, P. 2015. Design and construction of a new highly secure Android ROM. Tech. rep., Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India.

McGovern, R. 2012. *Inotifywait for Android*. https://github.com/mkttanabe/inotifywait-for-Android.

Peters, T. M., Gondree, M. A., and Peterson, Z. N. 2015. Defy: A deniable, encrypted file system for log-structured storage.

Torvalds, L. and Others, M. 2015. *The Linux Kernel*. http://www.kernel.org.

Wright, S. 2012. The symantec smartphone honey stick project. *Symantec Corporation, Mar*.