

MTech Thesis [Drafts] Stealth File Systems for Proactive Forensics Support in Custom Android ROMs

A [proposal of a] thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Technology in Cyber Security Systems and Networks

By

SUDIP HAZRA

Under The Guidance of

DR . PRABHAKER MATETI
WRIGHT STATE UNIVERSITY

2017
AMRITA VISHWA VIDYAPEETHAM
Amritapuri, Kerala 690525

AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM

July 25, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Sudip Hazra ENTITLED MTech Thesis [Drafts] Stealth File Systems for Proactive Forensics Support in Custom Android ROMs BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Technology in Cyber Security Systems and Networks.

Prabhaker Mateti, Ph. D.
Thesis Director

Krishna Sri Achuthan, Ph.D.
Department Head

Committee on
Final Examination

Prabhaker Mateti, Ph. D

Professor B, MTech or Ph. D

Professor B, MTech or Ph. D
Professor C, MTech or Ph. D

Krishna Sri Achuthan, Ph.D.
Dean, School of Graduate Studies

ABSTRACT

Hazra, Sudip M.Tech. Department of Cyber Security Systems and Networks, Amrita Vishwa Vidyapeetham, 2017. *Stealth File Systems for Proactive Forensics Support in Custom Android ROMs*

Smartphones nowadays are capable of doing a multitude of tasks which was not possible with conventional phones. The capabilities which makes smartphones different from conventional phones is giving nightmares to security organisations in tracing and extracting data from the smartphones. The criminals are now able to wipe out the traces of criminal activities performed using their phones with much ease than before. The criminal organisations are also using smartphones to communicate with others using encrypted messages which are very hard to trace and decrypt. Even if the phones are confiscated after the crime, It is becoming very hard for law-enforcement agencies to extract data from those devices as these devices are encrypted with advanced encryption features and any attempt to get access to the device memory using brute-force would potentially wipe the all the data hence a proactive forensic framework was necessary for continuous monitoring of the criminals using their smartphones. The Proactive Forensic Rom is capable of monitoring the online and offline activities of the suspected criminals with high degree of precision and stealth while discretely uploading the user data to the cloud for further monitoring and analysis purpose.

Keywords: Android, Forensics, Digital Evidence, Android Framework, adb tool, Pocket Spy.

Contents

1	Introduction	1
1.1	Smartphone OS Market Share	1
1.2	Android Architecture	2
1.2.1	Linux Kernel	3
1.2.2	Libraries	4
1.2.3	Android Runtime	4
1.2.4	Application Framework	5
1.2.5	Applications	5
1.2.6	Applications Used in Criminal Activites	6
2	Background	7
2.1	Fuse(File system in User Space)	8
2.2	Cloudfs	11
2.3	Stealthfs via Rootkits	12
2.3.1	User Level Rootkits	13
2.3.2	Kernel Level Rootkits	13
3	Problem Statement	15
3.1	Aim	16
4	Stealth File System with Cloud Support	18
4.1	Modules Description:	19
4.1.1	Forensic Service	19
4.1.2	Stealth File Systems	19
4.1.3	Cloud File System	20
5	Results and Discussion	22
5.1	Fuse File System in Linux	22
5.2	Mounting a Demo Fuse File System in Python	23
5.3	Mounting a Cloud Drive as Local File System	24
5.3.1	MegaFuse	24
5.4	Evidence Recovery using Sleuthkit Forensic Recovery Tool	27
5.4.1	Results	27
5.5	Hiding File Systems Using Rootkits	29

6	Related Work	30
7	Development Plan	34
7.1	Things To be Done	34

List of Tables

List of Figures

1.1	Gartner Survey Results	2
1.2	Android Architecture	3
2.1	Filesystem in Userspace	8
2.2	Making a Call in Fuse File System	10
4.1	Proposed System Architecture	18
5.1	Mounting a Fuse File System	23
5.2	Contents of the Mounted File System	24
5.3	Megafuse.h defining supported Fuse File Operations	25
5.4	Mounted Megafuse Cloud Drive	25
5.5	Running Megafuse Client	26
5.6	Megafuse File System as seen by df command	26
5.7	Device image Analysis using Sleuthkit	27
5.8	Recovered Images	28
5.9	Recovered Emails	28
5.10	Hiding Filesystem Using Rootkits	29

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 minor project 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 minor project. I would like to thank Dr Krishnashree Achuthan, Professor and Head of the Center for Cyber Security, for giving me useful suggestions and his constant encouragement and guidance throughout the progress of this minor project. I would like to thank my co-guides Mr. Vipin Pavithran, Assistant Professor, Cyber Security, Amrita Vishwa Vidyapeetham for giving me useful suggestions and guidance throughout the progress of this minor project. I express my special thanks to my colleagues who were with me from the starting of the minor project, 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 minor project.

Chapter 1

Introduction

Smartphones nowadays are capable of doing a multitude of tasks which was not possible with conventional phones, With the increase in complexity of smartphones, We can now send emails, click high-definition photos, Access satellite navigation and remain connected to the outside world 24 x 7. The capabilities which makes smartphones different from conventional phones is giving nightmares to security organisations in tracing and extracting data from the smartphones. The criminals are now able to wipe out the traces of criminal activities performed using their phones with much ease than before. The criminal organisations are also using smartphones to communicate with others using encrypted messages which are very hard to trace and decrypt. Even if the phones are confiscated after the crime, It is becoming very hard for law-enforcement agencies to extract data from those devices as these devices are encrypted with advanced encryption features and any attempt to get access to the device memory using brute-force would potentially wipe the all the data inside and overwrite it with 0. In this scenario, A radically new approach is necessary to proactively monitor the criminal activities with minimal interference and maximum stealth.

1.1 Smartphone OS Market Share

According to Gartner ,it's estimated that there are roughly 2 billion smartphone users in the market (1.91 billion to be exact), with that number expected to in-

crease another 12% in 2016 to top 2.16 billion people globally. The global share of smartphone OS is as follows:

Worldwide Smartphone Sales to End Users by Operating System in 1Q16 (Thousands of Units)

Operating System	1Q16 Units	1Q16 Market Share (%)	1Q15 Units	1Q15 Market Share (%)
Android	293,771.2	84.1	264,941.9	78.8
iOS	51,629.5	14.8	60,177.2	17.9
Windows	2,399.7	0.7	8,270.8	2.5
Blackberry	659.9	0.2	1,325.4	0.4
Others	791.1	0.2	1,582.5	0.5
Total	349,251.4	100.0	336,297.8	100.0

Source: Gartner (May2016)

Figure 1.1: Gartner Survey Results

Android dominated the smartphone market with a share of 84 % , IOS with 14.8 % and Windows with 0.7% of the Global smartphone market. As we can see Android OS is the most popular smartphone OS with almost 84 % market share and hence we are interested in examining the forensics capabilities in Android OS.

1.2 Android Architecture

Android is a mobile operating system developed by google based on the linux kernel and designed primarily for touchscreen mobile devices. Android source

code is open-source and it is one of the main reason of the immense popularity of the android OS.

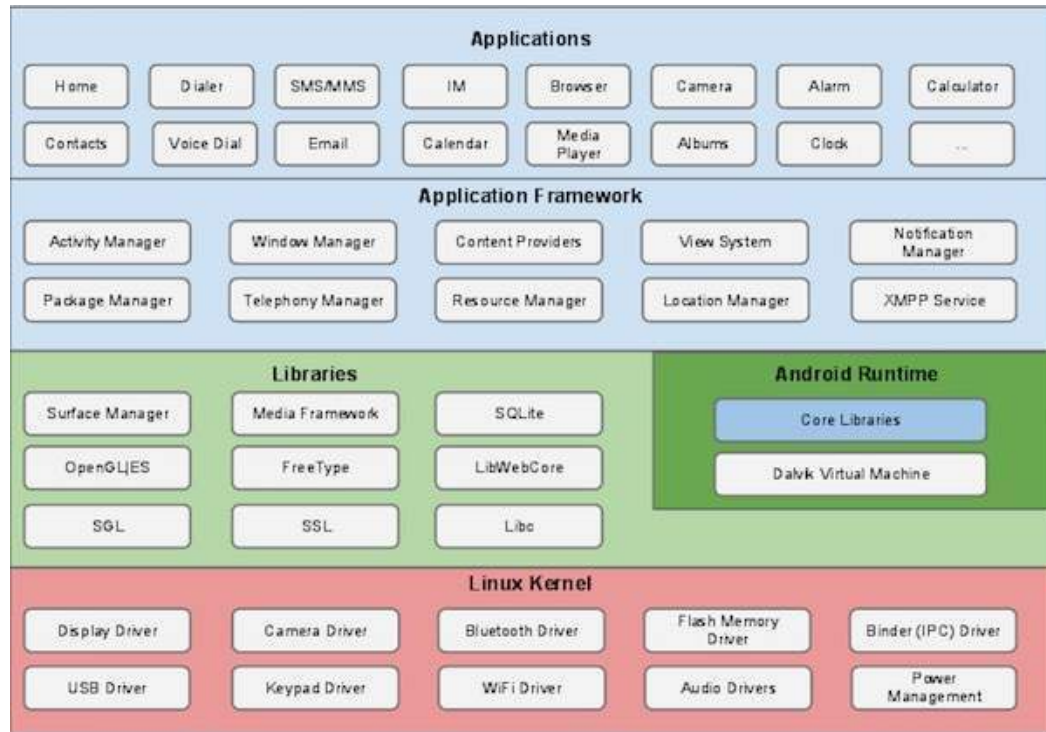


Figure 1.2: Android Architecture

1.2.1 Linux Kernel

The basic layer is the Linux kernel. The whole Android OS is built on top of the Linux 3.x Kernel with some further architectural changes made by Google. It is this Linux that interacts with the hardware and contains all the essential hardware drivers. Drivers are programs that control and communicate with the hardware. For example, consider the Bluetooth function. All devices has a Bluetooth hardware in it. Therefore the kernel must include a Bluetooth driver to communicate with the Bluetooth hardware. The Linux kernel also acts as an abstraction layer between the hardware and other software layers. Android uses the Linux for all its core functionality such as Memory management, process

management, networking, security settings etc. As the Android is built on a most popular and proven foundation, it made the porting of Android to variety of hardware, a relatively painless task.

1.2.2 Libraries

The next layer is the Android native libraries. It is this layer that enables the device to handle different types of data. These libraries are written in c or c++ language and are specific for a particular hardware. Some of the important native libraries include the following:

- **Surface Manager:** It is used for compositing window manager with off-screen buffering means you can't directly draw into the screen, but your drawings go to the off-screen buffer. There it is combined with other drawings and form the final screen the user will see. This off screen buffer is the reason behind the transparency of windows.
- **Media framework:** Media framework provides different media codecs allowing the recording and playback of different media formats.
- **SQLite:** SQLite is the database engine used in android for data storage purposes.
- **WebKit:** It is the browser engine used to display HTML content.
- **OpenGL:** Used to render 2D or 3D graphics content to the screen.

1.2.3 Android Runtime

Android Runtime consists of Dalvik Virtual machine and Core Java libraries.

- **Dalvik Virtual Machine:** It is a type of JVM used in android devices to run apps and is optimized for low processing power and low memory environments. Unlike the JVM, the Dalvik Virtual Machine does not run .class files, instead it runs .dex files. .dex files are built from .class file at the time of compilation and provides higher efficiency in low resource environments. The Dalvik VM allows multiple instance of Virtual machine to be created simultaneously providing security, isolation, memory

management and threading support. It is developed by Dan Bornstein of Google.

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.

- **Core Java Libraries:** These are different from Java SE and Java ME libraries. However these libraries provides most of the functionalities defined in the Java SE libraries.

1.2.4 Application Framework

These are the blocks that our applications directly interacts with. These programs manage the basic functions of phone like resource management, voice call management etc. Important blocks of Application framework are:

- **Activity Manager:**Manages the activity life cycle of applications.
- **Content Providers:** Manage the data sharing between applications.
- **Location Manager:** Location management, using GPS or cell tower .
- **Resource Manager:** Manage the various types of resources we use in our Application.

1.2.5 Applications

Applications are the top layer in the Android architecture and this is where our applications are going to fit. Several standard applications comes pre-installed with every device, such as:

- SMS client
- App Dialer
- Web browser
- Contact manager

1.2.6 Applications Used in Criminal Activities

The Criminals are increasingly becoming tech savvy and are using various apps which provide encryption facilities and as well as secure communication. This is becoming a problem for security agencies as they are not able to intercept the communication channel. Some of the apps used by Criminals Networks are as follows:

- **Mappr** An App that can change location data on photos, so they don't reveal where they actually are.
- **Cryptophone** An App that uses end-to-end encryption to for phone calls which effectively means that except the two communicating parties, no other person will be able to decrypt the data stream.
- **Telegram** An encrypted mobile messaging app that can host different channels where members can talk in a group setting.
- **Firechat** An App that connects to nearby devices which have firechat installed through wifi or bluetooth and build a "mesh network" that allows messages to be passed to other devices within vicinity without any usage of cell phone tower.

Chapter 2

Background

Filesystems are collections of methods and data structures that an operating system uses to keep track of data on disks or partitions. Different operating systems use different types of file systems like Windows uses NTFS and FAT32 file systems. Linux uses EXT4, EXT3, EXT2 etc. Earlier Android kernels used YAFFS2 file systems, However newer Android kernels use EXT4 file systems. File systems are implemented at kernel level. Implementing a new file system at the kernel level involves modifying the kernel source code and rebuilding it with the new file system support. This creates complexity which can be solved by implementing file system at the user level.

2.1 Fuse(File system in User Space)

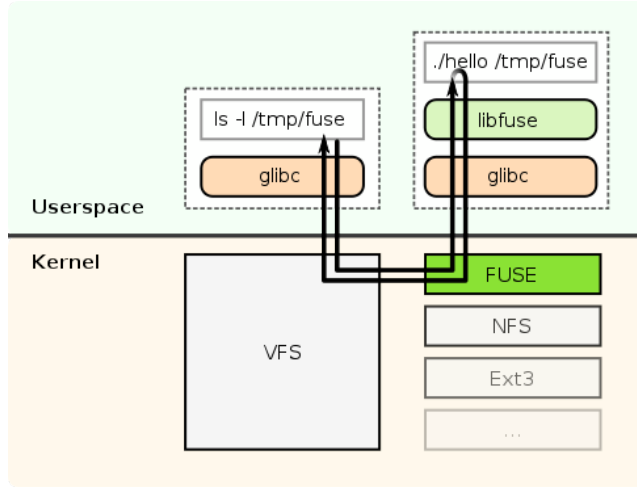


Figure 2.1: **Filesystem in Userspace**

FUSE (Filesystem in Userspace) is an interface for userspace programs to export a filesystem to the Linux kernel. The FUSE project consists of two components: the fuse kernel module and the libfuse userspace library. FUSE is particularly useful for writing virtual file systems. Unlike traditional file systems that essentially save data to, and retrieve data from, mass storage, virtual filesystems do not actually store data themselves. They act as a view or translation of an existing file system or storage device. FUSE has a small kernel module which plugs into the VFS layer in the kernel as a file system and then communicates with a user-level process that does all the work, using the FUSE library to communicate with the kernel module. IO requests from an application are redirected by the FUSE kernel module to this user level process for execution and the results are returned back to the requesting application. A fuse file system is a user-space program that listens on a socket for file operations to perform and perform them. The fuse library(libfuse) provides the communication with the socket and passes the requests to the code. The callbacks are a set of functions to implement the file operations and a struct fuse_operations contains pointers to them.

Necessary operations in fuse_operation structure [Pfeiffer \[2016\]](#)

```
struct fuse_operations {  
  
    • int (*getattr) (const char *, struct stat *);  
  
    • int (*readlink) (const char *, char *, size_t);  
  
    • int (*getdir) (const char *, fuse_dirh_t, fuse_dirfil_t);  
  
    • int (*mknod) (const char *, mode_t, dev_t);  
  
    • int (*mkdir) (const char *, mode_t);  
  
    • int (*unlink) (const char *);  
  
    • int (*rmdir) (const char *);  
  
    • int (*symlink) (const char *, const char *);  
  
    • int (*rename) (const char *, const char *);  
  
    • int (*link) (const char *, const char *);  
  
    • int (*chmod) (const char *, mode_t);  
  
    • int (*chown) (const char *, uid_t, gid_t);  
  
    • int (*truncate) (const char *, off_t);  
  
    • int (*utime) (const char *, struct utimbuf *);  
  
    • int (*open) (const char *, struct fuse_file_info *);  
  
    • int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);  
  
    • int (*write) (const char *, const char *, size_t, off_t, struct fuse_file_info *);  
  
    • int (*statfs) (const char *, struct statfs *);  
  
    • int (*flush) (const char *, struct fuse_file_info *);  
  
    • int (*release) (const char *, struct fuse_file_info *);  
  
    • int (*fsync) (const char *, int, struct fuse_file_info *);  
};
```

- `int (*setxattr) (const char *, const char *, const char *, size_t, int);`
 - `int (*getxattr) (const char *, const char *, char *, size_t);`
 - `int (*listxattr) (const char *, char *, size_t);`
 - `int (*removexattr) (const char *, const char *);`
- `};`

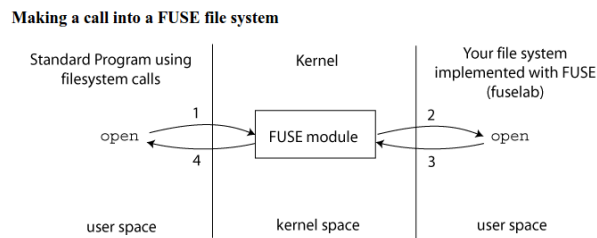


Figure 2.2: Making a Call in Fuse File System

Example of Fuse Filesystems: **Wikipedia** [2016]

- **CloudStore (formerly, Kosmos filesystem):** By mounting via FUSE, existing Linux utilities can interface with CloudStore
- **EncFS:** Encrypted virtual filesystem
- **ExpanDrive:** A commercial filesystem implementing SFTP/FTP/S3/Swift using FUSE
- **GDFS:** Filesystem which allows you to mount your Google Drive account on Linux.
- **GlusterFS:** Clustered Distributed Filesystem having ability to scale up to several petabytes.
- **GmailFS:** Filesystem which stores data as mail in Gmail.
- **GVfs:** The virtual filesystem for the GNOME desktop.
- **KBFS:** A distributed filesystem with end-to-end encryption and a global namespace that uses FUSE to create cryptographically secure file mounts.

- **MooseFS:** An open source distributed fault-tolerant file system available on every OS with FUSE implementation (Linux, FreeBSD, NetBSD, OpenSolaris, OS X), able to store petabytes of data spread over several servers visible as one resource.
- **NTFS-3G and Captive NTFS:** allowing access to NTFS filesystems.
- **Sector File System:** Sector is a distributed file system designed for large amount of commodity computers. Sector uses FUSE to provide a mountable local file system interface.
- **SSHFS:** Provides access to a remote filesystem through SSH.
- **Transmit:** A commercial FTP client that also adds the ability to mount WebDAV, SFTP, FTP and Amazon S3 servers as disks in Finder, via MacFUSE.
- **WebDrive:** A commercial filesystem implementing WebDAV, SFTP, FTP, FTPS and Amazon S3.
- **WikipediaFS:** View and edit Wikipedia articles as if they were real files.
- **Wuala:** A multi-platform, Java-based fully OS integrated distributed file system. Using FUSE, MacFUSE and Callback File System respectively for file system integration, in addition to a Java-based app accessible from any Java-enabled web browser.

2.2 Cloudfs

CloudFS will be developed using the file system in user-space (FUSE) toolkit. FUSE provides a framework for implementing a file system at user level. CloudFS will be implemented as user-level code that uses libfuse to communicate with applications and uses regular file system calls to access the storage. By default, FUSE is multi-threaded, so multiple system calls from user applications can be running at the same time in the user-level process, allowing higher parallelism and faster performance. Cloudfs can be used to mount cloud storage like dropbox, google drive and onedrive as local file system and any changes that are made in the local file system are synchronised with cloud storage. For example :

- WingFS, the universal cloud adapter, mounts the cloud service on the desktop. Storage services can be used like a hard drive without the need for apps, specific folders, browser plug-ins and login processes.
- Google Cloud Storage Fuse: It is an open source fuse adapter that allow us to mount Google cloud storage buckets as file systems on Linux or OS X systems. It also provides a way for applications to upload and download google cloud storage objects using standard file system semantics. It is written in Go language and hosted in github.
- The Storage Made Easy Linux App Suite is comprised of an integrated Cloud Drive (that uses FUSE), a graphical Cloud File Explorer and also Sync tools to sync cloud files to/from the Linux desktop.
- Drop-Fuse is a simple fuse based file system which is written in python and can mount the dropbox drive as local file system in Linux.

In our case We will be implementing the cloud fs below the android software stack in linux level and api's will be used synchronization and mounting a multitude of cloud storage services.

2.3 Stealthfs via Rootkits

When we install a rootkit in that system we will be able to get administrator privileges whenever you want. Good rootkits can hide in compromised system, so that they can't be found by administrator. There are many ways to hide in a system. For example there are rootkits that replace some most important programs in system (ls, ps, netstat etc.) with modified versions of them that won't let administrator see that something's wrong. Although, such a rootkit is quite easy to detect. Other rootkits work as linux kernel modules. They work in kernel mode, so they can do everything they want. They can hide themselves, files, processes etc.

Different types of Rootkits are explained below.

2.3.1 User Level Rootkits

User level rootkits are unprivileged and are stored outside of kernel memory space. They are user space code that patches or replaces existing applications to provide cover for malicious activities. User level rootkits replace system binaries, add malicious utilities, change configuration files, delete files, or launch malicious processes. For example, the Linux system program `ls` could be changed so as to not reveal the presence of a malicious file in a directory. These rootkits, while effective, are easily detected by file system integrity and signature checking tools, therefore, most modern IDS software prevent these rootkits from being installed or can detect an active intruder.

2.3.2 Kernel Level Rootkits

Kernel level rootkits defeat such tools by directly modifying the operating system kernel. These rootkits modify the execution flow of kernel code to run their own payload. However, modifying the kernel in this way can drastically affect the stability of the system causing a kernel panic.

The simplest way to introduce code into a running kernel is through a Loadable Kernel Module (LKM). LKMs add flexibility to an operating system by providing a means to add functionality without recompiling the entire kernel. Added functionality might include device drivers, filesystem drivers, system calls, network drivers, TTY line disciplines, and executable interpreters. Most modern UNIX-like systems, including Solaris, Linux, and FreeBSD, use or support LKMs. However, the kernel packaged with Android does not support LKMs by default. The kernel can be recompiled and installed on Android to add LKM support if physical access to the mobile is available. LKMs are very useful, but they also allow maliciously written kernel modules to subvert the entire operating system which can lead to a loss of control of the Linux kernel and consequently all the layers above the kernel. Kernel level rootkits typically subvert the kernel to hide processes, modules, connections and more to avoid detection. Particular techniques include hooking system calls, direct kernel object manipulation (DKOM), run-time kernel memory patching, interrupt descriptor table hooking, and intercepting calls handled by Virtual File System (VFS). These techniques are discussed at a high level in this section.

Hooking System Calls

The kernel provides a set of interfaces or system calls by which processes running in user space can interact with the system. The applications in user space send requests through this interface and the kernel fulfills requests or returns an error. Hooking is a technique that employs handler function, called hooks, to modify control flow . A new hook registers its address as the location for a specific function, so when that function is called the hook runs instead. Typically, a hook will call the original function at some point to preserve the original behavior. System calls can be hooked using a maliciously designed LKM to alter the structure of the system call table.

Direct Kernel Object Manipulation (DKOM)

Hooking the system write call allows a rootkit to hide from system binaries like ls, lsmod, and ps; even so, robust IDSs can still detect the existence by following the kernel structures. All operating systems store internal record-keeping keeping data in main memory . The Linux kernel is no exception and provides generic data structures and primitives to encourage code reuse by developers . These structures, that all programmers are familiar with, include linked lists, queues, maps, and binary trees. Altering the data in these structures to hide an attackers activity is called Direct Kernel Object Manipulation (DKOM).

Interrupt Descriptor Table (IDT) Hooking

An interrupt is an event that alters the sequence of instructions executed by the processor. When an interrupt occurs a system table called the Interrupt Descriptor Table (IDT) associates each interrupt or exception with the address of the corresponding handler . System calls use software interrupts to switch from user mode to kernel mode. The interrupt handler invokes the system call handler from the address stored in the system call table. A rootkit can hook the IDT by modifying the interrupt handler address in the IDT or by patching the first few instructions of the interrupt handler .These modifications would put the rootkit code in the flow of execution while still letting the system handle interrupts properly.

Chapter 3

Problem Statement

The project is a part of an ongoing project on Proactive Forensics. The Previous Work has been done by Karthik Et.al [Rao \[2016\]](#) and Aiyyappan Et.al [Aiyyappan \[2015\]](#). Aiyyappan ported the inotifywait to android. File events are tracked by inotify tool and the inotify source was complied using NDK programming and compiled to a native shared object library. The native function accepts a directory to track and all file events are tracked. He also created the forensic examiner toolkit which runs on linux machine to image, recover and collect device specific data from the cloud. Some amount of stealth was also enabled using hidepid =2 , where users can only see their own processes and process id's are also hidden from /proc also.

Karthik created the android apk which extensively tracks the user activities like GPS,Sensor data,WIFI Metadata,Sms,Call recording and keylogger was also included in the apk , which could be configured when the rom is flashed and used for the first time, after that there is no dialog whatsoever and the app runs in stealth mode and saves all the forensically relevant data in/forensic partition and opportunistically uploads it to the cloud.However the background process can be discovered if the user roots the phone and gets root access , Then he /she can easily see all the background process running and can also discover the /forensic partition , compromising the evidence , even though the partition is encrypted but it will still create suspicion in the suspects mind.

3.1 Aim

There are two possible solutions in this scenario:

- 1. The /forensic partition can either be encrypted and stored in the device, However it may lead to suspicion as the user will be able to read all data except that partition.
- 2. Another solution is to create A Fuse File System and store the forensically relevant data in that file system. The file system can then be hidden using rootkits which will subvert commands such as df and du and prevent any normal user from detecting the file system.

The Proposed Framework Will be Consisting of:

- An EXT4 file system which will store the forensically relevant data from the device. The forensic service designed by Karthik Rao [2016] and Aiyappan Aiyappan [2015] will store all the user data obtained from the device in this file system.
- An user space file system based on FUSE library which will mount the cloud storage as a local drive in the device accessible to the forensic service. The cloud drive will support major cloud storage service providers and will be configurable using an apk and a config file. The stealth file system will copy itself piece by piece to the cloud file system which will then opportunistically upload it to the cloud storage. This will give more storage for the forensic service to store greater amount of evidence in comparison to limited amount of device storage.
- A kernel level android rootkit which will hide both the EXT4 File system and the cloud file system from typical linux command like df, du and mount commands by hooking the sys_call_table and subverting the systems calls.

- Combining All the above we get a stealth file system with cloud file system support which will be mounted below the android software stack in the linux level.

Advantages of Mounting The cloud storage as local file system:

- We will be able to do all file system operations possible such as create, delete, modify files and directories.
- All applications will see the mounted cloud storage as local storage and will be able to perform all operations possible with a local file system.
- The storage capacity will increase manifold, typical modern android devices have storage capacity of maximum 64GB however if we are mounting the cloud drive as local storage , we are potentially having unlimited access to the storage . The storage capacity will increase manifold with terrabytes of storage space.
- The Android Cloud Storage Service(ACCS) will also be employed in our companion projects :
 - 1.Detecting Trojans in Android Devices by Manuel Antony.

Chapter 4

Stealth File System with Cloud Support

Here we propose a Stealth File system with cloud support Below the Android Software stack.

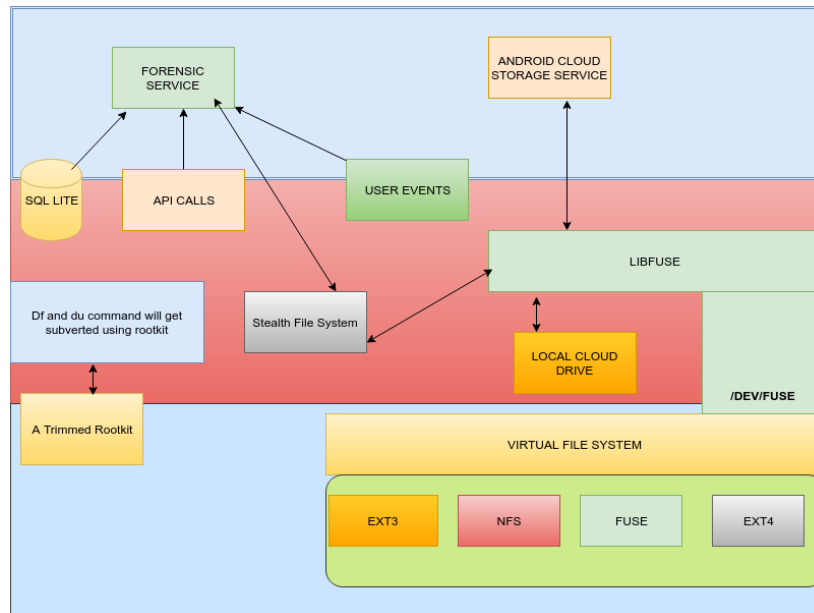


Figure 4.1: Proposed System Architecture

4.1 Modules Description:

4.1.1 Forensic Service

The forensic service designed by Karthik Rao [2016] Et.al and Aiyappan [2015] will collect the forensically relevant data from the device partitions and will send it to the /forensic file system. The data sent will include:

- 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

In addition to these Data Keylogger data ,call record data and Sensor data will also be sent to the forensic partition.

4.1.2 Stealth File Systems

The stealth file system will be a seperate file system which will be used by the forensic service to copy all the forensically relevant data from the relevant device partitions like /data to the stealth file system. The stealth file system will be based on ext4 file systems, rootkits will be used to hide the file volumes from commands like df and du. The rootkit will be responsible for subverting systems calls which can potentially expose the presence of the either the forensic

service running or the stealth file system and the cloud file system. The rootkit will also hide the forensic service running in the background by hooking on to the `sys_call_table` and filtering out the output. In future the stealth file system will be based on a separate emmc partition for added stealth.

4.1.3 Cloud File System

The cloud File system will be made using fuse, and we will be able to locally mount a cloud drive as the cloud file system. It will support various cloud providers using api's. The cloud drive will opportunistically upload the data to the cloud storage online. The cloud file system will also be made stealth using rootkits and seamlessly copy data from the stealth file systems and sync it. It will also act as a cache to prevent data loss.

Cloud Drives can be mounted as Local Drives using the various cloud storage provider api's like:

Cloud Storage API's Examples:

Dropbox Cloud Storage Api's:

- Create a Dropbox folder
`post('https://api.dropbox.com/1/fileops/create_folder', args)`
- Rename a Dropbox file/directory object.
`post('https://api.dropbox.com/1/fileops/move', args)`
- Delete a Dropbox file/directory object.
`post('https://api.dropbox.com/1/fileops/delete', args)`
- Get Dropbox metadata of path.
`get('https://api.dropbox.com/1/metadata/auto' + path, args)`

As per the official android documentation the external storage (SD cards) are accessed by the Android system using FUSE which implies that FUSE is supported by the kernel directly so we dont need to add fuse support in kernel , we need to develop the cloudfs which will be controlled by the Android Cloud Storage Service(ACCS) which will run as a background service and will be configurable using an Apk and .config file.The stealth file system with the cloud file system will work with forensic service developed by aiyyappan and karthik and will be incorporated into the forensic rom.

Chapter 5

Results and Discussion

The Following implementation have been done on Ubuntu Linux and our objective is to implement the framework in a Linux environment and then port it to Android.

5.1 Fuse File System in Linux

To develop a filesystem, first download the FUSE source code from [here](#) and unpack the package This creates a FUSE directory with the source code. The contents of the fuse directory are:

- ./doc contains FUSE-related documentation. At this point, there is only one file, how-fuse- works.
- ./kernel contains the FUSE kernel module source code .
- ./include contains the FUSE API headers, which needed to create a filesystem. The only one you need now is fuse.h.
- ./lib holds the source code to create the FUSE libraries to link binaries to create a filesystem.
- ./util has the source code for the FUSE utility library.

- ./example, of course, contains samples for your reference, like the fusexmp.null and hello filesystems.

Build and install FUSE:

1. Run the configure script from the fuse-2.2 directory: ./configure . This creates the required makefiles, etc.
2. Run ./make to build the libraries, binaries, and kernel module. Check the kernel directory for the file ./kernel/fuse.ko – this is the kernel module file. Also check the lib directory for fuse.o, mount.o, and helper.o.
3. Run ./make install to complete the installation of FUSE.

Fusepy

fusepy is a Python module that provides a simple interface to FUSE_ and MacFUSE_. It's just one file and is implemented using ctypes.

5.2 Mounting a Demo Fuse File System in Python

```
sudip@sudip:~$ cd /mnt/fusepy/
sudip@sudip:/mnt/fusepy$ ls -l
total 0
sudip@sudip:/mnt/fusepy$ cd /home/sudip/Desktop/android\ security\Thesis\Stealth
\ File\ Systems\fusepy/
sudip@sudip:~/Desktop/android security\Thesis\Stealth File Systems\fusepy$ pytho
n demo.py /home/sudip/Desktop/android\ security/ /mnt/fusepy/
```

Figure 5.1: Mounting a Fuse File System

```

sudip@sudip:/mnt$ cd /mnt/
sudip@sudip:/mnt$ cd fusepy/
sudip@sudip:/mnt/fusepy$ ls -l
total 0
-rw----- 1 sudip sudip 261287 Dec  5 13:06 746-s13-proj2.pdf
drwxrwxr-x 14 sudip sudip  4096 Dec  6 04:02 android apk and sourcecode
drwxrwxr-x  2 sudip sudip  4096 Sep 11 23:24 Android Development Tools
drwxrwxr-x  2 sudip sudip  4096 Dec  8 12:20 Android Videos
drwxrwxr-x  2 sudip sudip  4096 Sep 11 23:25 Assignments
-rw----- 1 sudip sudip 145858 Dec  8 19:29 CS135 FUSE Documentation.pdf
-rw----- 1 sudip sudip 467520 Nov 27 19:03 [GUIDE] Restore Moto E (2015) Stoc
k Firmware _ Moto E 2015.pdf
drwxrwxr-x  2 sudip sudip  4096 Oct 22 08:07 java
-rw----- 1 sudip sudip 860501 Dec  7 19:21 Julien Marchand • Dev blog _ Yet a
nother dev blog.pdf
-rw----- 1 sudip sudip 143955 Dec  8 19:41 l-fuse-pdf.pdf
-rw----- 1 sudip sudip 276603 Dec  8 23:50 Linux Rootkits 101 (1 of 3).pdf
-rw----- 1 sudip sudip 147177 Dec  7 19:18 MegaFS, a FUSE filesystem wrapper
for Mega. Part 1_ Listing files.pdf
-rw----- 1 sudip sudip 174061 Dec  7 19:19 MegaFS, a FUSE filesystem wrapper
for Mega. Part 2_ Reading and writing files.pdf
drwxrwxr-x  2 sudip sudip  4096 Sep 12 12:18 Roms
-rw----- 1 sudip sudip 165884 Dec  8 21:37 Rx.pdf
drwxrwxr-x  9 sudip sudip  4096 Dec  9 16:37 Thesis
-rw----- 1 sudip sudip 375138 Nov 28 02:24 (Tools) - unpack_repack boot.img_r
ecovery.pdf
-rw----- 1 sudip sudip 142922 Dec  7 19:19 Using the Mega API_ how to upload
a file anonymously (without logging in).pdf
-rw----- 1 sudip sudip 357286 Dec  7 18:35 Using the Mega API, with Python ex
amples! _ Julien Marchand • Dev blog.pdf
-rw----- 1 sudip sudip 264084 Dec  8 23:50 Writing Linux Rootkits 201 (2_3).p
df
-rw----- 1 sudip sudip 265301 Dec  8 23:51 Writing Linux Rootkits 301.pdf
sudip@sudip:/mnt/fusepy$

```

Figure 5.2: Contents of the Mounted File System

5.3 Mounting a Cloud Drive as Local File System

5.3.1 MegaFuse

This is a linux client for the MEGA cloud storage provider. It is based on FUSE and it allows to mount the remote cloud drive on the local filesystem. Once mounted, all linux program will see the cloud drive as a normal folder.

This software is based on FUSE.

Megafs.h header Information

```

int open(const char *path, struct fuse_file_info *fi);
int readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi);
int getattr(const char *path, struct stat *stbuf);
int release(const char *path, struct fuse_file_info *fi);
int releaseNoIhumb(const char *path, struct fuse_file_info *fi);
int mkdir(const char *path, mode_t mode);
int read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi);
int create(const char *path, mode_t mode, struct fuse_file_info *fi);
int write(const char *path, const void *buf, size_t size, off_t offset, struct fuse_file_info *fi);
int write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi);
int rename(const char *src, const char *dst);
int unlink(const char *);
int truncate(const char *a, off_t o);

int setattr(const char *path, const char *name, const char *value, size_t size, int flags);
int raw_setattr(const char *path, const char *name, const char *value, size_t size, int flags);
int getattr(const char *path, const char *name, char *value, size_t size);
int symlink(const char *path1, const char *path2);
int readlink(const char *path, char *buf, size_t bufsz);
int serializeXattr();
uint32_t addChunk(void *binbuff, uint32_t *buffsize, char *data, uint32_t data_size);
int unserializeXattr();
int serializeSymlink();
int unserializeSymlink();
int stringToFile(const char *path, std::string tnpbuf, size_t bufsz, fuse_file_info *fi);

```

Figure 5.3: Megafuse.h defining supported Fuse File Operations

```

sudip@sudip:/mnt$ sudo su
[sudo] password for sudip:
root@sudip:/mnt# ls -l
total 0
drwxrwxrwx 1 root root 4096 Aug 19 00:01 fusepy
root@sudip:/mnt# cd fusepy/
root@sudip:/mnt/fusepy# ls -l
total 0
drwxrwxrwx 1 root root 4096 Aug 19 11:29 bug_bounty
drwxrwxrwx 1 root root 4096 Sep 1 21:59 cloud_forensics
-rw-rw-rw- 1 root root 81305876 Oct 23 11:38 CSN_624-20161022T100152Z.zip
drwxrwxrwx 1 root root 4096 Sep 9 18:19 DbSecurity
-rw-rw-rw- 1 root root 957179 Sep 9 19:19 DbSecurityAssignment-Sudip Hazra-
2nd_Yr-P2CSN15027.tar.gz
-rw-rw-rw- 1 root root 246321519 Aug 19 01:41 EQGRP-Auction-Files.zip
drwxrwxrwx 1 root root 4096 Aug 19 11:29 forensic_methods_in_clouds
-rw-rw-rw- 1 root root 409711 Aug 19 11:30 Malware_Analysts.odt
-rw-rw-rw- 1 root root 208345 Aug 19 11:30 Malware_Analysis.pdf
drwxrwxrwx 1 root root 4096 Sep 9 19:18 php
drwxrwxrwx 1 root root 4096 Sep 5 10:10 python_books
drwxrwxrwx 1 root root 4096 Oct 19 22:47 reversing
drwxrwxrwx 1 root root 4096 Aug 19 11:29 rtf_books
-rw-rw-rw- 1 root root 32638187 Oct 23 11:31 Security-Introduction-20161022T100
226Z.zip

```

Figure 5.4: Mounted Megafuse Cloud Drive

```

root@sudip:/home/sudip/Desktop/android security/Thesis/MegaFuse# ./MegaFuse
caricata la variabile CACHEPATH con il valore /tmp
Username (email): hazrasudip9@gmail.com
Enter your password:
Specify a valid mountpoint (an empty directory): /mnt/fusepy
MegaFushe::MegaFuse. Constructor finished.
157 files and 28 folders added or updated
1 user received or updated
/usr/bin/notify-send
[17:45:47] MegaFuse is ready
flags:80000
searching node by path: /.megafuse_symlink
file .megafuse_symlink not found in MEGA
MegaFuse::unserializeSymlink. no symlink stored. Do nothing.
MegaFuse::getAttr Looking for /
/ not found in cache
flags:80000
searching node by path: /.megafuse_xattr
file .megafuse_xattr not found in MEGA
MegaFuse::unserializeXattr. No xattr stored. Do nothing.
MegaFuse::getxattr. looking for path '/', name 'security.selinux'
MegaFuse::getxattr. looking for path '10003140', name '2628e040'
getxattr. No path '/'(10003140) found on getxattr
getxattr. Warning. attribute not found: security.selinux on path /
MegaFuse::getxattr. looking for path '/', name 'system.posix_acl_access'
MegaFuse::getxattr. looking for path '8000990', name '2624f040'
getxattr. No path '/'(8000990) found on getxattr
getxattr. Warning. attribute not found: system.posix_acl_access on path /
MegaFuse::getxattr. looking for path '/', name 'system.posix_acl_default'
MegaFuse::getxattr. looking for path '10003030', name '2628e040'
getxattr. No path '/'(10003030) found on getxattr
getxattr. Warning. attribute not found: system.posix_acl_default on path /

```

Figure 5.5: Running Megafuse Client

```

root@sudip:/mnt/fusepy# df

```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
udev	2769912	4	2769908	1%	/dev
tmpfs	556488	1608	554880	1%	/run
/dev/sda2	474426520	326909736	123394212	73%	/
none	4	0	4	0%	/sys/fs/cgroup
none	5120	0	5120	0%	/run/lock
none	2782424	19232	2763192	1%	/run/shm
none	102400	100	102300	1%	/run/user
/dev/sda1	523248	3456	519792	1%	/boot/efi
megafuse	52428800	52428800	51380224	51%	/mnt/fusepy

Figure 5.6: Megafuse File System as seen by df command

5.4 Evidence Recovery using Sleuthkit Forensic Recovery Tool

Sleuthkit [Carrier et al. \[2015\]](#) is an open-source Forensic Recovery toolkit capable of recovering forensic artifacts from device memory image. We used an android device for 1 week and then made factory reset , and created an image of device using dd command and ran sleuthkit over it to recover artifacts.

5.4.1 Results

Running Sleuthkit Forensic Toolkit

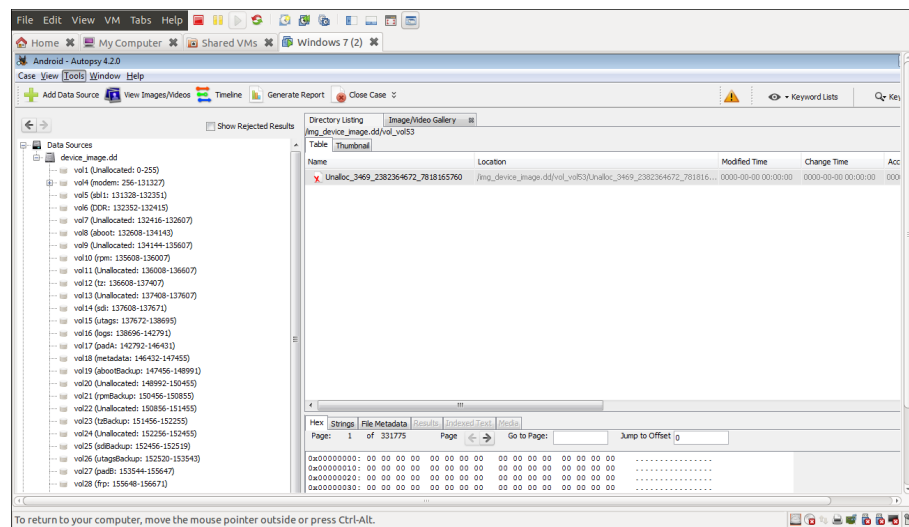
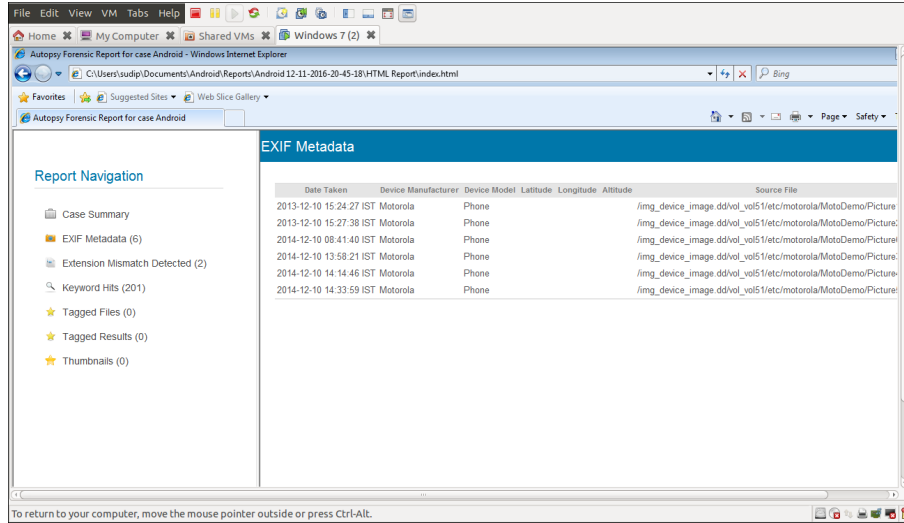


Figure 5.7: Device image Analysis using Sleuthkit

Recovered Images



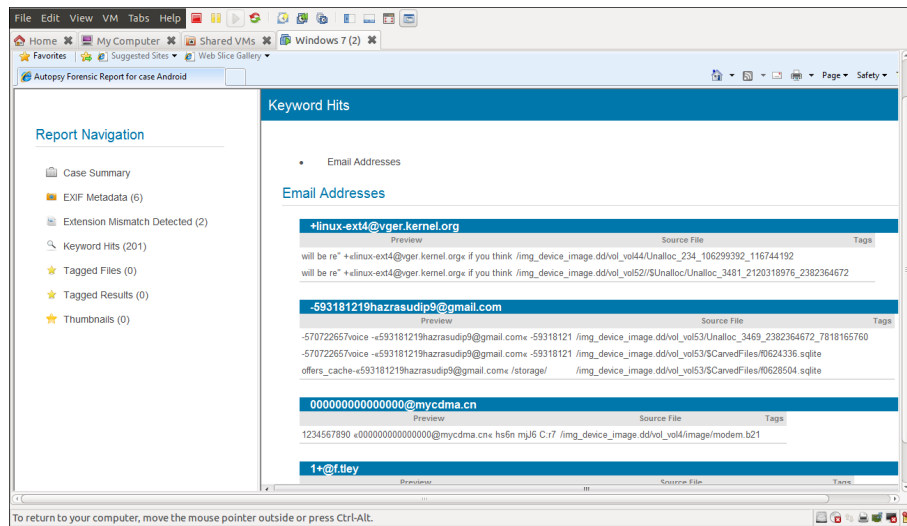
The screenshot shows the Autopsy Forensic Report interface for a case named 'Autopsy Forensic Report for case Android'. The left sidebar contains a 'Report Navigation' menu with options: Case Summary, EXIF Metadata (6), Extension Mismatch Detected (2), Keyword Hits (201), Tagged Files (0), Tagged Results (0), and Thumbnails (0). The main pane displays the 'EXIF Metadata' section, which contains a table with the following data:

Date Taken	Device Manufacturer	Device Model	Latitude	Longitude	Altitude	Source File
2013-12-10 15:24:27 IST	Motorola	Phone				/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture
2013-12-10 15:27:38 IST	Motorola	Phone				/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture
2014-12-10 08:41:40 IST	Motorola	Phone				/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture
2014-12-10 13:58:21 IST	Motorola	Phone				/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture
2014-12-10 14:14:46 IST	Motorola	Phone				/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture
2014-12-10 14:33:59 IST	Motorola	Phone				/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture

At the bottom of the window, a status bar reads: 'To return to your computer, move the mouse pointer outside or press Ctrl-Alt.'

Figure 5.8: Very Few Images Could be Recovered After Factory Reset of device

Recovered Email Id's



The screenshot shows the Autopsy Forensic Report interface for a case named 'Autopsy Forensic Report for case Android'. The left sidebar contains a 'Report Navigation' menu with options: Case Summary, EXIF Metadata (6), Extension Mismatch Detected (2), Keyword Hits (201), Tagged Files (0), Tagged Results (0), and Thumbnails (0). The main pane displays the 'Keyword Hits' section, which contains a list of email addresses under the heading 'Email Addresses'.

Email Address	Source File	Tags
+linux-ext4@vger.kernel.org	/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture	
-593181219hazrasudip9@gmail.com	/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture	
0000000000000000@mycdma.cn	/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture	
1+@t.ley	/img_device_image.ddvol_vo51/etc/motorola/MotoDemo/Picture	

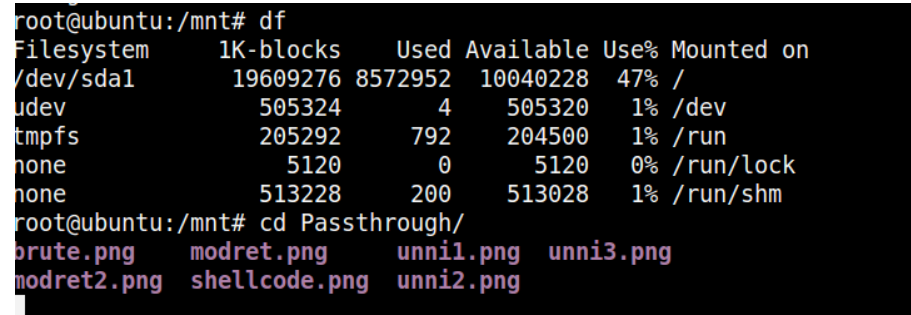
At the bottom of the window, a status bar reads: 'To return to your computer, move the mouse pointer outside or press Ctrl-Alt.'

Figure 5.9: Very Few Emails Could be recovered after Factory Reset of device

5.5 Hiding File Systems Using Rootkits

A Rootkit can be hidden by finding out the location of the `sys_call_table` and hijack write with our own write function. We'll first want to save a copy of the original write to pass data off to and restore when the module is unloaded. To do this we'll simply use the `xchg()` function and exchange the two pointers.

Hiding Filesystems using Rootkits



```
root@ubuntu:/mnt# df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda1       19609276 8572952  10040228  47% /
udev            505324      4    505320    1% /dev
tmpfs           205292      792    204500    1% /run
none            5120        0      5120     0% /run/lock
none            513228     200    513028    1% /run/shm
root@ubuntu:/mnt# cd Passthrough/
brute.png      modret.png     unni1.png     unni3.png
modret2.png    shellcode.png  unni2.png
```

Figure 5.10: The write system call is hijacked, However the filesystem is still mounted.

Chapter 6

Related Work

The Following is Collection of Literature Review on Works related to Android Forensics and other articles related to our area of interest.

Androphsy: Forensic Framework for Android

Akarawita et al. [2015] implemented a forensic framework which can acquire data both Physically and logically from the android smartphone. For physical acquisition they used DD program to clone the system image. For logical acquisition they use adbpull to clone the filesystem partition, other tools used were logcat,demsg ,dumpsys,scalpel and adb getprop to get device properties.Used Netcat to copy the system files to a remote server. It was better than other Opensource Forensic tools like Oxygen and ViaExtract CE tool. Rooting of the phone is necessary.

Smartphone Forensics : A proactive Investigative Approach

Mylonas et al. [2012] suggested a proactive forensic framework which is regulated by an independent authority. Two modes of forensics namely Usermode and Network mode. No information about Implementation , was the phone rooted or unrooted, functionality of the app not given.

Frost: Forensic Recovery of Scrambled Telephones

Müller and Spreitzenbarth [2013]devised an forensic image which was flashed

on to the phone and it was capable to bruteforcing Pin, direct recovery of encryption keys and decrypting user partition on phone itself. If the bootloader is locked not of much use, only option is to take memory dump but all smartphones now comes with default locked bootloader, secondly only if the handset is instantly available to the expert, he can freeze the ram to minimize data loss and recover encryption keys from ram.

Android Forensics : Automated Data Collection and Reporting Tool for Mobile Device

Justin Grover [Grovera \[2013\]](#) made the first of its kind android forensics tool which collected data with user consent and uploaded it to a remote server. The capabilities are limited and are susceptible to tampering. Droidwatch mainly uses content observers, Broadcast Receivers and Alarms for monitoring. Broadcast Receivers and Alarms can be tampered with. No data about social networking apps, no support was there for email.

Specifying a Realistic File System

Bilbyfs [Amani and Murray \[2015\]](#) is an asynchronous write flash file system whose formal verification is done using Isabelle/Hol. The file system needs a C-Wrapper which is placed between the VFS and the filesystem hence direct mapping is not possible. which can be a disadvantage. It follows strict ordering of updates and do not support concurrency. It was formally verified that the Bilbyfs indeed follows async writes.

A Fast Boot, Fast Shutdown Technique for Android OS Devices

The paper [Yang et al. \[2016\]](#) demonstrates a new technique of Fast Boot in android called FBFS. Here the snapshot is taken of the system only once and hence afterwards whenever the system boots, it makes a call to `RSS_thread` to sync the ram image with the latest files in emmc. significantly decreases the startup time to 7.8secs and shutdown to normal 10.3 secs, but when will the system know when to update the snapshot upon system update or a specified amount of time is not addressed.

Internet-Scale File Analysis

A [Hanif et al. \[2015\]](#) large scale malware analysis framework implemented in skald framework and has loosely coupled components like transporter ,planner and services, all inter-connected but highly failure resistant. TOTEM is the name of the tool they created using the framework, written in scala using AKKA. Planner designates tasks to various services which execute and the reports are forwarded using RabbitMQ messaging protocol . Highly concurrent and a very good technique for mass scale malware analysis but more details are needed how the service outputs are fed to humans or scripts for report generation.

Subverting Operating System Properties Through Evolutionary DKOM Attacks

New type [Graziano et al. \[2016\]](#) of DKOM attack ,does not change the kernel dynamic data structures at once but continues to do it over a period of time. Difficult to detect because there is no sudden change but continuous one. attacks the cfs tree of the scheduler but not the process tree hence the app being attacked shows in the ps but it is indefinitely delayed in the scheduler as the vruntime is set to max. Detected using a thin hypervisor debugger which uses defensive mimic technique to detect the attack using periodic monitor and task tracker. Can stop ids or any antivirus and do not modify kernel code. Various other variation possible like attacking memory management.

Forensic Analysis of Instant Messenger Applications on Android devices

[Mahajan et al.](#) have done forensic investigation on whatsapp and viber using Celebrite UEFD Classic device , they were able to extract whatsapp and viber data however on using the physical analyzer software of Celebrite, it succeeded incase of whatsapp but failed in case of viber. Manual Analysis of the viber folder were needed. Pretty much everything was extracted like chat messages, images videos with timestamp, however the data in internal memory of sdcard was encrypted however they did not test it after the deleting the data , was the tool able to extract data from the unallocated space is unknown.

Forensics Analysis of Whatsapp Messenger on Android Smartphones

Anglano Et.Al [Anglano \[2014\]](#) wrote a very good paper decoding the whatsapp artifacts and step by step linking to what is the greater picture . Artifacts were carefully correlated to infer all the required information and tracing events even if the messages were deleted using whatsapp logs but very little support what will happen if whatsapp is removed using Uninstall- it like app or the phone is formatted.

M.S Thesis on Forensic Analysis of whatsapp on Android Smartphone Neha Thakur

Whatsapp [Thakur \[2013\]](#) Forensics can be done in two ways : If whatsapp folder is in Sdcard can be decrypted using WhatsapXtract Tool which is now obsolete beacause the Key has changed andd we need to edit the code to add the new key. Memory Forensics of volatile memory can also be done , suing memfetch to extract selective portions of ram , taking the heap of the selected appliaction and running data extraction tool on the memory dump. Recently deleted messages can be easily discovered. Used a tool called WhatsappRamXtract.

Chapter 7

Development Plan

Our Development Plan is as follows:

7.1 Things To be Done

1. Creating a java/scala based cloud file system with support for multiple cloud storage providers.
2. Creating a new EXT4 partition in the android device (/forensic partition).
3. Creating a Android Rootkit based on ARM architecture to hijack system calls to subvert various linux commands which might compromise our file system.

I am now currently looking at various java based cloud file systems which can be incorporated into Android.

Exploring if google firebase can be used for this purpose.

Our Implementation will be done in Linux machine and then we will port it to Android. chapter Conclusion And Future Work Forensic Investigation is becoming increasingly complex day by day. The onset of mobile computing and end-to-end encryption implies that smartphones can now compute more

complex data and this has facilitated in development of applications which uses end-to-end encryption for message encryption, In such scenarios even Man-in-the-middle attacks are not successful. Smartphones memory can be erased with applications which erase and over-write the memory multiple times with 0, making the device unfit for forensic evidence collection. In such scenarios, A proactive forensic framework is necessary which will be able to monitor suspected criminals 24 x 7. This forensic framework will be able monitor the suspects with maximum accuracy and stealth. The evidence files will be directly copied to the cloud drive which will opportunistically upload it to the cloud server for analysis. The android cloud storage service will also be helpful in case of malware detection services which require to store large amounts of data for static and dynamic analysis.

Bibliography

- Aiyyappan, PS, 2015. Android Forensic Support Framework. Master's thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. Advisor: Prabhaker Mateti, <http://cecs.wright.edu/~pmateti/Students/index.html>.
- Akarawita, Indeewari U., Perera, Amila B., and Atukorale, Ajantha, 2015. ANDROPHSY -Forensic Framework for Android. 2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer). doi: 10.1109/ictcr.2015.7377696.
- Amani, Sidney and Murray, Toby, 2015. Specifying a realistic file system. arXiv preprint arXiv:1511.04169.
- Anglano, Cosimo, 2014. Forensic analysis of WhatsApp Messenger on Android smartphones. Digital Investigation, 11(3):201213. doi:10.1016/j.diin.2014.04.003.
- Carrier, Brian, Farmer, Dan, and Venema, Wietse, 2015. Sleuthkit: Open Source Digital Forensics. sleuthkit.org.
- Graziano, Mariano, Flore, Lorenzo, Lanzi, Andrea, and Balzarotti, Davide, 2016. Subverting Operating System Properties Through Evolutionary DKOM Attacks. In DIMVA, 3–24. Springer.
- Grovera, Justin, 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.

- Hanif, Zach, Lengyel, Tamas K, and Webster, George D, 2015. Internet-Scale File Analysis. Black Hat USA.
- Mahajan, Aditya, Dahiya, MS, and Sanghvi, HP, 2016. Forensic Analysis of Instant Messenger Applications on Android devices. International Journal of Computer Applications, 68(8). <http://research.ijcaonline.org/volume68/number8/pxc3886965.pdf>.
- Müller, Tilo and Spreitzenbarth, Michael, 2013. Frost. In Applied Cryptography and Network Security, 373–388. Springer. <https://www1.cs.fau.de/frost>.
- Mylonas, Alexios, Meletiadiis, Vasilis, Tsoumas, Bill, Mitrou, Lilian, and Gritzalis, Dimitris, 2012. Smartphone Forensics: A Proactive Investigation Scheme for Evidence Acquisition. IFIP Advances in Information and Communication Technology Information Security and Privacy Research, 249260. doi: 10.1007/978-3-642-30436-1_21.
- Pfeiffer, Joseph J., 2016. Writing a FUSE Filesystem: a Tutorial. <http://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/>.
- Rao, Karthik M., 2016. Proactive Forensic Support for Android Devices. Master's thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. Advisor: Prabhaker Mateti, <http://cecs.wright.edu/~pmateti/Students/index.html>.
- Thakur, Neha S, 2013. Forensic Analysis of WhatsApp on Android Smartphones. Master's thesis, University of New Orleans. <http://scholarworks.uno.edu/cgi/viewcontent.cgi?article=2736&context=td>.
- Wikipedia, 2016. Filesystem in Userspace — Wikipedia, The Free Encyclopedia. <http://en.wikipedia.org/w/index.php?title=Filesystem%20in%20Userspace&oldid=748163155>.
- Yang, Xia, Shi, Peng, Sun, Haiyong, Zheng, Wenxuan, and Alves-Foss, Jim, 2016. A Fast Boot, Fast Shutdown Technique for Android OS Devices. Computer, 49(7):62–68.