Amrita Vishwa Vidyapeetham
Mini Project Report
Stealth File Systems for Proactive Forensics Support in Custom Android ROM

By

SUDIP HAZRA
AM.EN.P2CSN15027

Under The Guidance of

DR. PRABHAKAR MATETI
WRIGHT STATE UNIVERSITY

Amrita Center For Cyber Security Systems and Networks
AMRITA VISHWA VIDYAPEETHAM

A minor thesis report submitted to Amrita Vishwa Vidyapeetham in accordance with the requirements of the degree of MTECH in Cyber Security Systems and Networks.

DECEMBER 2015

Amrita Vishwa Vidyapeetham
Amritapuri, Kollam

BONAFIDE CERTIFICATE

This is to certify that this mini project report entitled **Stealth File Systems for Proactive Forensics Support in Custom Android ROM** submitted by **SUDIP HAZRA (Reg.No : AM.EN.P2CSN15027)** in partial fulfillment of the requirements for the award of the Degree of Master of Technology in CYBER SECURITY SYSTEMS AND NETWORKS is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering.

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

Dr.Prabhakar Mateti
(Project Guide)

Dr.Krishnashree Achuthan
Professor and Head

Date:

DEDICATION AND ACKNOWLEDGEMENTS

At the very outset, I would like to give the first honors to **Amma, Mata Amritanan- damayi 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 Profes- sor, 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.

Author's declaration

I,**Sudip Hazra (Registered Number AM.EN.P2CSN15027)** hereby declare that this project report,entitled **Stealth File Systems for Proactive Forensics Support in Custom Android ROM**is a record of the original work done by me under the guidance of /textbf Dr.Prabhakar Mateti,Wright State University was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with,or with the assistance of, others, is indicated as such to the best of my knowledge.

Signature of the Student:
Date:

PLACE: AMRITAPURI.

CounterSigned

Dr.Krishnashree Achuthan
Professor and Head
Amrita Center for Cyber Security Systems and Networks

Abstract

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 satelite 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 access 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.

Table of Contents

List of Tables

Table **Page**

LIST OF FIGURES

CHAPTER 1

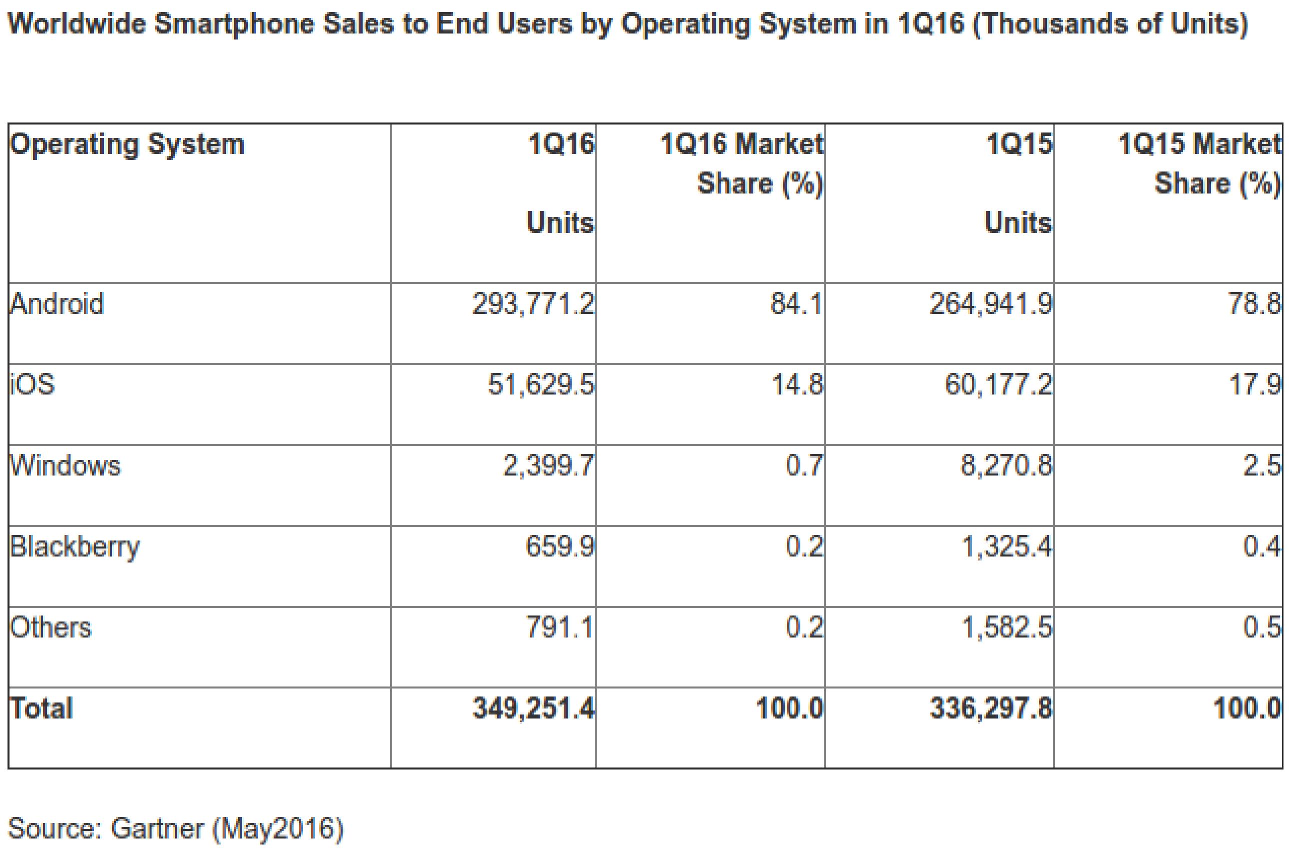
INTRODUCTION

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

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.



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

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.

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.

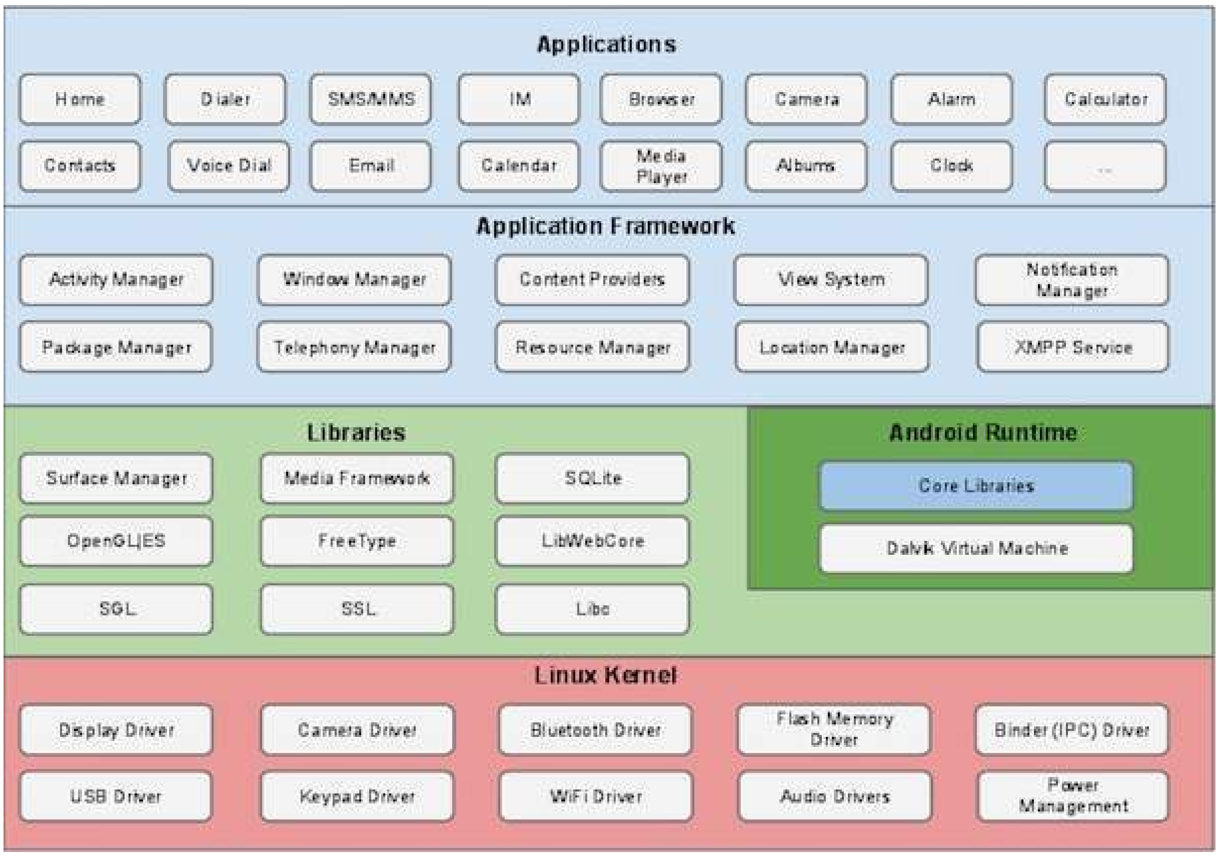


Figure 1.2: Android Architecture from [16]

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 cant 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 doesn‚Äôt run .class files, instead it runs .dex files. .dex files are built from .class file at the time of compilation and provides hifger 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 aheadof-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 gonna fit. Several standard applications comes pre-installed with every device, such as:

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

.

Chapter 2

Related Work

2.1 Androphsy: Forensic Framework for Android

Akarawita Et.Al[4] 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.

2.2 Smartphone Forensics : A proactive Investigative Approach

Mylonas Et.Al[14] 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.

2.3 Frost: Forensic Recovery of Scambled Telephones

Muller Et.Al[13] deviced 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.

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

Justin Grover[8] 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.

2.5 Specifying a Realistic File System

BilbyFs[5] is a 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 .

2.6 A Fast Boot, Fast Shutdown Technique for Android OS Devices

The paper [17] 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.

2.7 Internet-Scale File Analysis

A [9] 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 tey 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.

2.8 Subverting Operating System Properties Through Evolutionary DKOM Attacks

[7]New type 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.

2.9 Forensic Analysis of Instant Messenger Applications on Android devices

Adiya Mahajan Et.al [11] 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.

2.10 Forensics Analysis of Whatsapp Messenger on Android Smartphones

Anglano Et.Al [6] 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.

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

[15]Whatsapp 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 3

PROBLEM STATEMENT

The project is a part of an ongoing project. The Previous Work has been done by Karthik Et.al [10] and Aiyyappan Et.al [3]. Aiyyappan Et.al 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 excepts 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 speciofic data from the cloud. Some amount of stealth was alos enabled using hidepid =2 , where users can only see their own processes and process id's are also hidden from /proc also.

Karthik Et.al [10] 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 tealth mode and saves all the forensically relevant data in/forrensic partition and oppurtunistically 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 ruunning 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

The solution is to create A Fuse File System and enable Stealth Features and Copy all Forensically Relevant Data in that File System. There will be two fuse file system one will directly mount the cloud storage in the linux level and the other file system will be used to store all the forensically relevant data and it will copy the data to the cloud storage file system which will oppurtunistically sync with the cloud storage online. Both file system will be hidden using kernel Rootkits.

CHAPTER 4

BACKGROUND

4.1 Fuse(File system in User Space)

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,

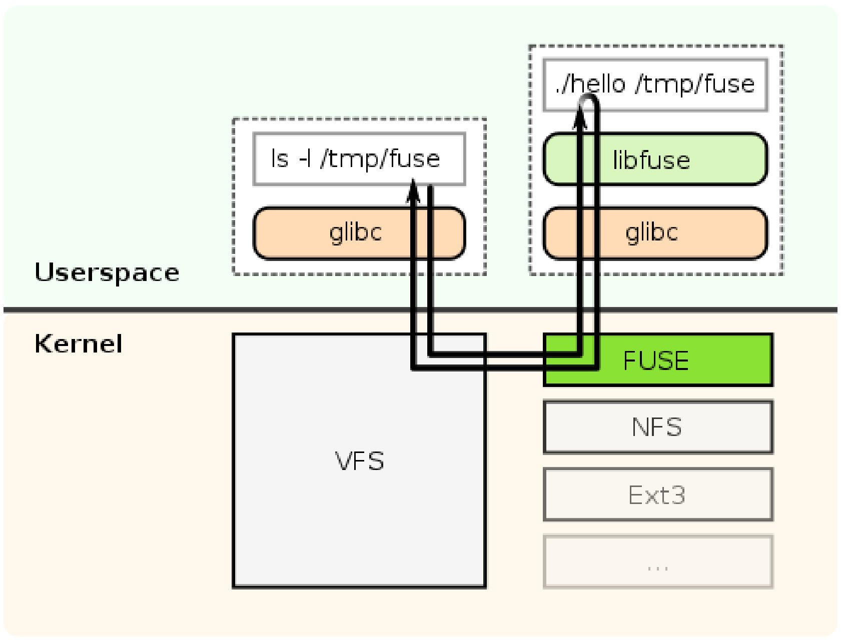


Figure 4.1: Filesystem in Userspace [2]

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 opertions in fuse_operation structure

```
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 *);
};
```

Some of the api are explained below:

- getattr: int (*getattr) (const char *, struct stat *);

  This is similar to stat() . The st_dev and st_blksize fields are ignored. The st_ino field is ignored unless the use_ino mount option is given.

- readlink: int (*readlink) (const char *, char *, size_t);

  This reads the target of a symbolic link. The buffer should be filled with a null-terminated string. The buffer size argument includes the space for the terminating null character. If the
  linkname is too long to fit in the buffer, it should be truncated. The return value should be "0"
  for success.

- getdir: int (*getdir) (const char *, fuse_dirh_t, fuse_dirfil_t);

  This reads the contents of a directory. This operation is the opendir() , readdir() , closedir() sequence in one call. For each directory entry, the filldir() function should be called.

- mknod: int (*mknod) (const char *, mode_t, dev_t);

  This creates a file node. There is no create() operation; mknod() will be called for creation of all non-directory, non-symlink nodes.

- mkdir: int (*mkdir) (const char *, mode_t);

- rmdir: int (*rmdir) (const char *);

  These create and remove a directory, respectively.

- unlink: int (*unlink) (const char *);

- rename: int (*rename) (const char *, const char *);

  These remove and rename a file, respectively.

- symlink: int (*symlink) (const char *, const char *);

  This creates a symbolic link.

- link: int (*link) (const char *, const char *);

  This creates a hard link to a file.

- chmod: int (*chmod) (const char *, mode_t);

- chown: int (*chown) (const char *, uid_t, gid_t);

- truncate: int (*truncate) (const char *, off_t);

- utime: int (*utime) (const char *, struct utimbuf *);

  These change the permission bits, owner and group, size, and access/modification times of a file, respectively.

- open: int (*open) (const char *, struct fuse_file_info *);

  This is the file open operation. No creation or truncation flags (O_CREAT , O_EXCL , O_TRUNC)

  will be passed to open() . This should check if the operation is permitted for the given flags. Optionally, open() may also return an arbitrary filehandle in the fuse_file_info structure, which will be passed to all file operations.

- read: int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);

  This reads data from an open file. read() should return exactly the number of bytes requested, except on EOF or error; otherwise, the rest of the data will be substituted with zeroes. An exception to this is when the direct_io mount option is specified, in which case the return value of the read() system call will reflect the return value of this operation.

Example of Fuse Filesystems:[2]

- **CloudStore (formerly, Kosmos filesystem):** By mounting via FUSE, existing Linux utilities can interface with CloudStore

- **EncFS:** Encrypted virtual filesystem

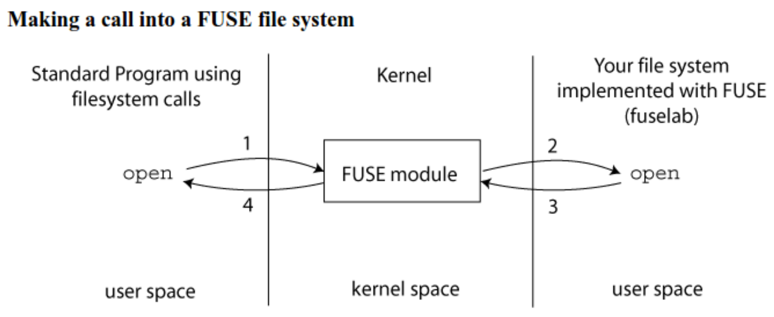

Making a call into a FUSE file system
Standard Program using filesystem calls
Kernel
Your file system implemented with FUSE (fuselab)
1
2
open
FUSE module
open
4
3
user space
kernel space
user space

Figure 4.2: Making a Call in Fuse File System [1]

- **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.

4.2 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 syncronised with cloud storage. For example :

- WingFS, the universal coud 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 llowas 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 synchronisation and mounting a multitude of cloud storage services.

4.3 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.**

4.3.1 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 ‚Äòls‚Äô 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.

4.3.2 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.

4.3.2.1 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 tha t 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.

4.3.2.2 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 attacker‚Äôs activity is called Direct Kernel Object Manipulation (DKOM).

4.3.2.3 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 5
PROPOSED SYSTEM

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

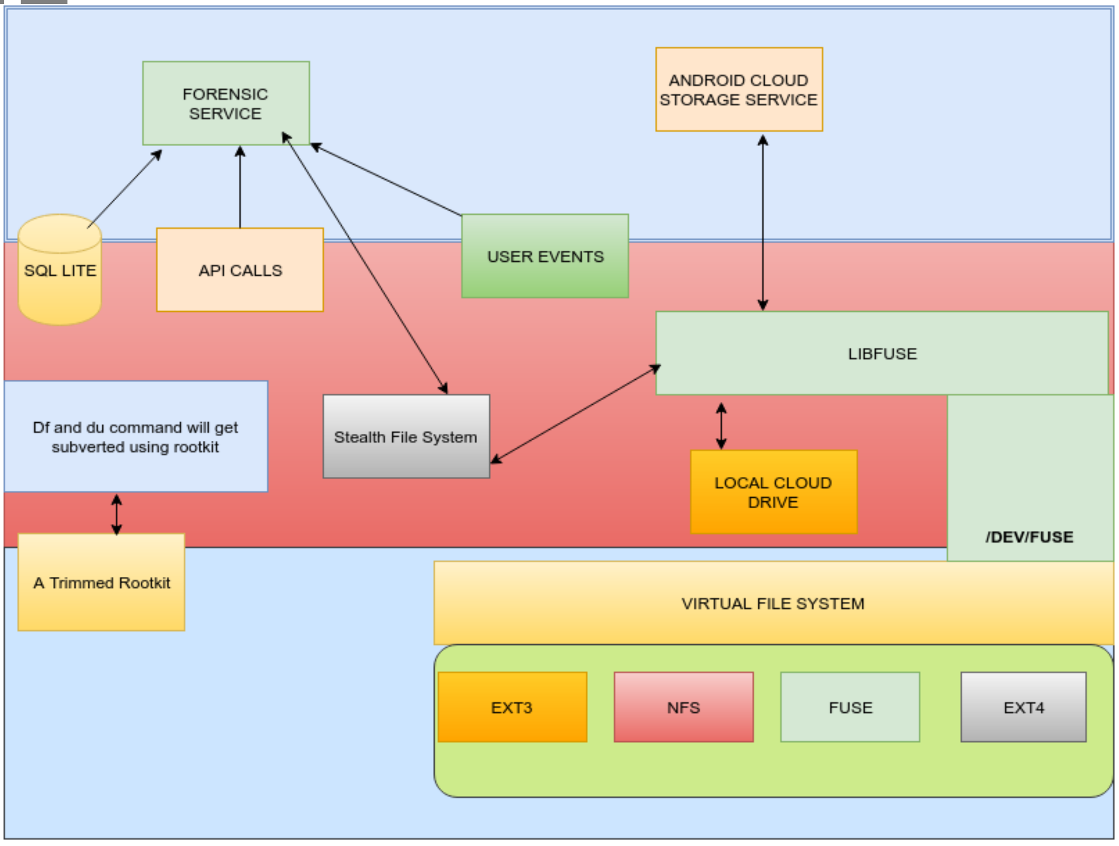


Figure 5.1: Proposed System Architecture

5.1 Modules Description:

5.1.1 Forensic Service

The forensic service designed by Karthik [10] et.al and aiyyappan Et.al[3] 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.

5.1.2 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 seperate emmc partition for added stealth.

5.1.3 Cloud File System

The cloud File system will made using fuse , and we will be able to localy mount a cloud drive as the cloud file system. It will support various cloud providers using api's .The cloud drive will oppurtunistically 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 a cache to prevent data loss.

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 6

Results and Discussion

6.1 Fuse File System in Linux

To develop a filesystem, first download the FUSE source code https://github.com/libfuse/libfuse/releases and unpack the package This creates a FUSE directory with the source code. The contents of the fuse directory are:

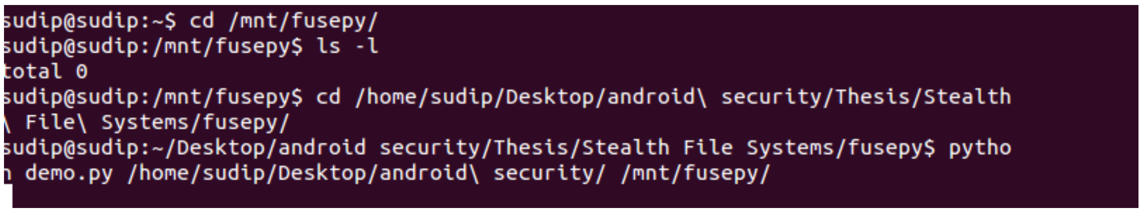
- ./doc contains FUSE-related documentation. At this point, there is only one file, how-fuse-works.
- ./kernel contains the FUSE kernel module source code .
- ./include contains the FUSE API headers, which needed to create a filesystem. The only one you need now is fuse.h.
- ./lib holds the source code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

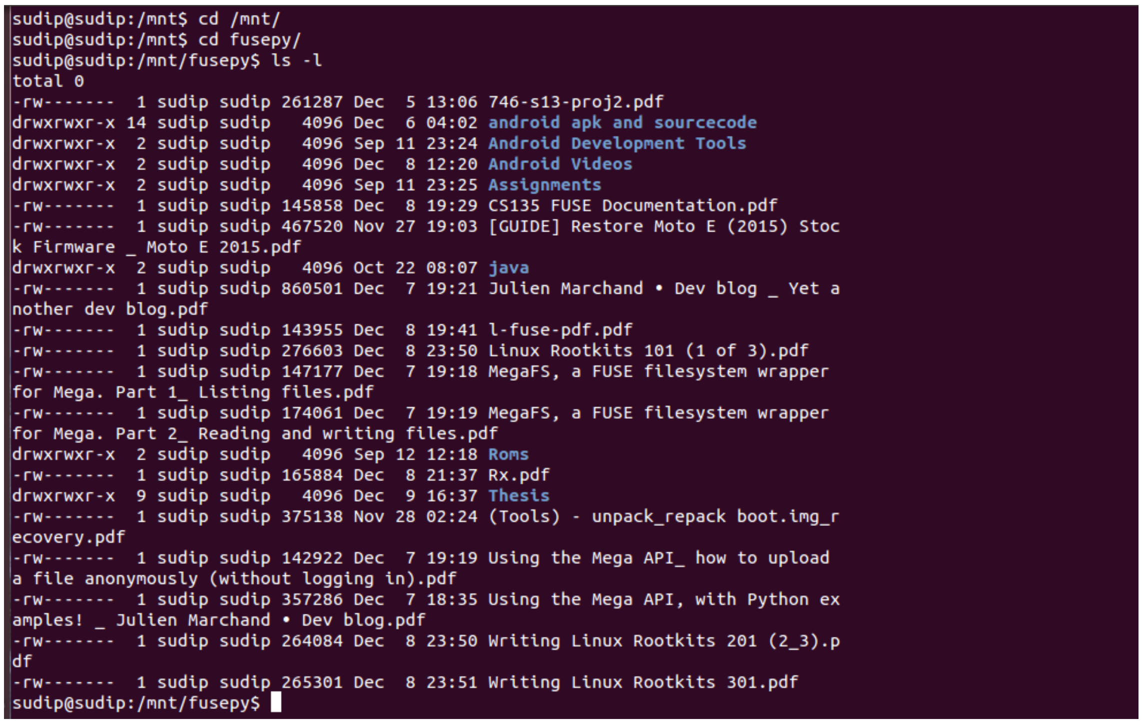
Fusepy
fusepy is a Python module that provides a simple interface to FUSE_ and MacFUSE_. It's just one file and is implemented using ctypes.

6.2 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 6.1: 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 6.2: Contents of the Mounted File System

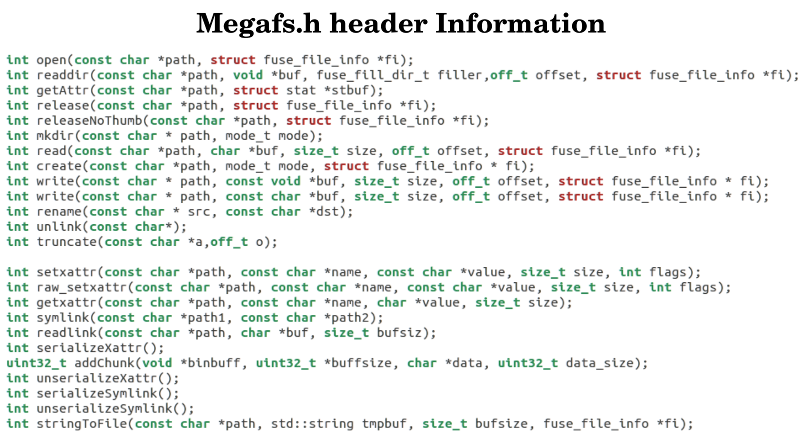
6.3 Mounting a Cloud drive as Local File System

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

6.3.1 MegaFuse [12]

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.

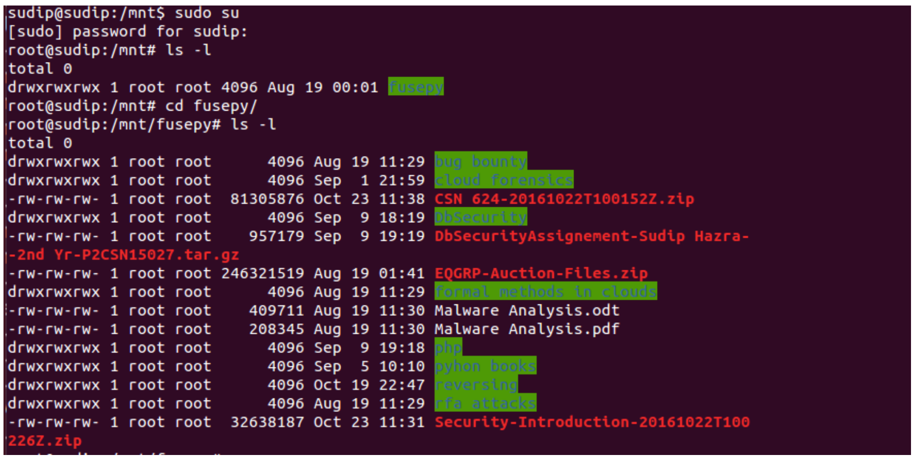
This software is based on FUSE.



Megafs.h header Information

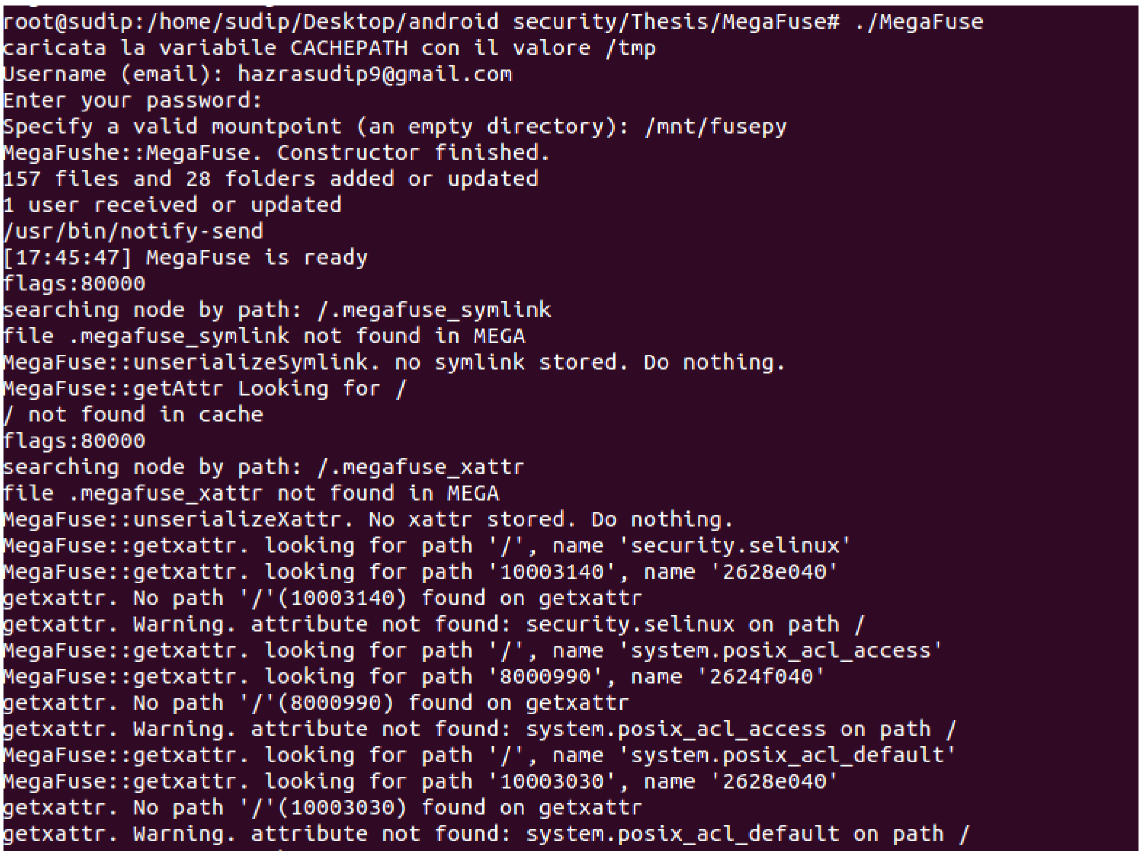
```
int open(const char *path, struct fuse_file_info *fi);
int readdir(const char *path, void *buf, fuse_fill_dir_t filler,off_t offset, struct fuse_file_info *fi);
int getAttr(const char *path, struct stat *stbuf);
int release(const char *path, struct fuse_file_info *fi);
int releaseNoThumb(const char *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 setxattr(const char *path, const char *name, const char *value, size_t size, int flags);
int raw_setxattr(const char *path, const char *name, const char *value, size_t size, int flags);
int getxattr(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 bufsiz);
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 tmpbuf, size_t bufsize, fuse_file_info *fi);
```

Figure 6.3: Megafuse.h defining supported Fuse File Operations



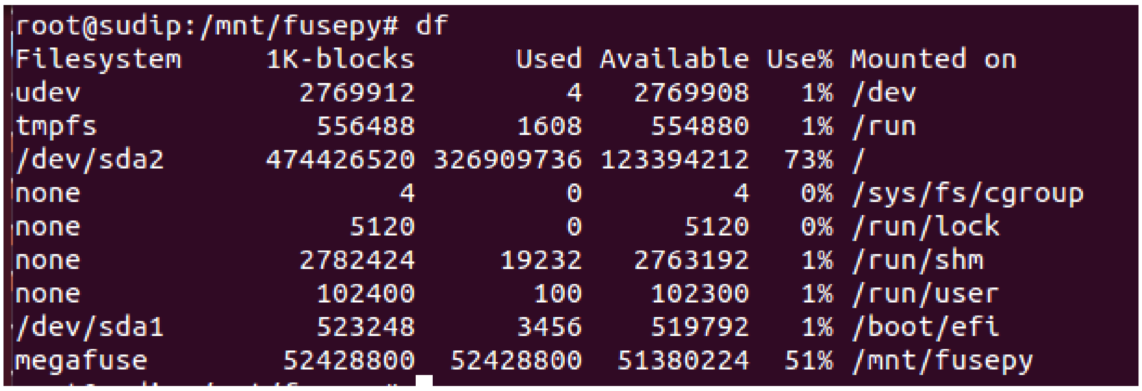
```
sudip@sudip:/mnt$ sudo su
[sudo] password for sudip:
root@sudip:/mnt# ls -l
total 0
drwxrwxrwx 1 root root 4096 Aug 19 00:01 fusepy
root@sudip:/mnt# cd fusepy/
root@sudip:/mnt/fusepy# ls -l
total 0
drwxrwxrwx 1 root root      4096 Aug 19 11:29 bug bounty
drwxrwxrwx 1 root root      4096 Sep  1 21:59 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 DbSecurityAssignement-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 formal methods in clouds
-rw-rw-rw- 1 root root    409711 Aug 19 11:30 Malware Analysis.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 pyhon books
drwxrwxrwx 1 root root      4096 Oct 19 22:47 reversing
drwxrwxrwx 1 root root      4096 Aug 19 11:29 rfa attacks
-rw-rw-rw- 1 root root  32638187 Oct 23 11:31 Security-Introduction-20161022T100
226Z.zip
```

Figure 6.4: 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 6.5: 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 6.6: Megafuse File System as seen by df command

CHAPTER 7

CONCLUSION

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.

Appendix A

Begins an appendix

BIBLIOGRAPHY

[1] *File systems and fuse*.

[2] *Filesystem in userspace*.

[3] AIYYAPPAN AND PRABHAKAR, *Android forensic support framework*, Master's thesis, 2015.

[4] I. U. AKARAWITA, A. B. PERERA, AND A. ATUKORALE, *Androphsy - forensic framework for android*, 2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer), (2015).

[5] S. AMANI AND T. MURRAY, *Specifying a realistic file system*, Electron. Proc. Theor. Comput. Sci. Electronic Proceedings in Theoretical Computer Science EPTCS, 196 (2015), p. 1‚Äì9.

[6] C. ANGLANO, *Forensic analysis of whatsapp messenger on android smartphones*, Digital Investigation, 11 (2014), p. 201‚Äì213.

[7] M. GRAZIANO, L. FLORE, A. LANZI, AND D. BALZAROTTI, *Subverting operating system properties through evolutionary dkom attacks*, Detection of Intrusions and Malware, and Vulnerability Assessment Lecture Notes in Computer Science, (2016), p. 3‚Äì24.

[8] J. GROVER, *Android forensics: Automated data collection and reporting from a mobile device*, Digital Investigation, 10 (2013).

[9] Z. HANIF, T. K. LENGYEL, AND G. D. WEBSTER, *Internet-scale file analysis*, Black Hat Usa, (2015).

[10] KARTHIK AND PRABHAKAR, *Proactive forensic support for android devices*, Master's thesis, 2016.

[11] A. MAHAJAN, M. S. DAHIYA, AND H. P. SANGHVI, *Forensic analysis of instant messenger applications on android devices*, International Journal of Computer Applications, 68 (2013), p. 38‚Äì44.

[12] MATTEOSERVA, *matteoserva/megafuse*, May 2016.

[13] T. MÜLLER AND M. SPREITZENBARTH, *Frost: Forensic recovery of scrambled telephones*, (2013), pp. 373–388.

[14] A. MYLONAS, V. MELETIADIS, B. TSOUMAS, L. MITROU, AND D. GRITZALIS, *Smartphone forensics: A proactive investigation scheme for evidence acquisition*, IFIP Advances in Information and Communication Technology Information Security and Privacy Research, (2012), p. 249‚Äì260.

[15] N. THAKUR, *Forensic Analysis of whatsapp on Android Smartphone*, PhD thesis, 2013.

[16] TUTORIALSPOINT.COM, *Android tutorial*.

[17] X. YANG, P. SHI, H. SUN, W. ZHENG, AND J. ALVES-FOSS, *A fast boot, fast shutdown technique for android os devices*, Computer, 49 (2016), p. 62‚Äì68.