12-4-2013

Procedures And Tools For Acquisition And Analysis Of Volatile Memory On Android Smartphones

Andri P. Heriyanto
Edith Cowan University, aheriyan@our.ecu.edu.au

DOI: 10.4225/75/57b3c9bafb86f
11th Australian Digital Forensics Conference. Held on the 2nd-4th December, 2013 at Edith Cowan University, Perth, Western Australia

PROCEDURES AND TOOLS FOR ACQUISITION AND ANALYSIS OF VOLATILE MEMORY ON ANDROID SMARTPHONES

Andri P Heriyanto
School of Computer and Security Science
Edith Cowan University, Perth, Australia
aheriyan@our.ecu.edu.au

Abstract
Mobile phone forensics have become more prominent since mobile phones have become ubiquitous both for personal and business practice. Android smartphones show tremendous growth in the global market share. Many researchers and works show the procedures and techniques for the acquisition andanalysisthe non-volatile memory inmobile phones. On the other hand, the physical memory (RAM) on the smartphone might retain incriminating evidence that could be acquired and analysed by the examiner. This study reveals the proper procedure for acquiring the volatile memory inthe Android smartphone and discusses the use of Linux Memory Extraction (LiME) for dumping the volatile memory. The study also discusses the analysis process of the memory image with Volatility 2.3, especially how the application shows its capability analysis. Despite its advancement there are two major concerns for both applications. First, the examiners have to gain root privileges before executing LiME. Second, both applications have no generic solution or approach. On the other hand, currently there is no other tool or option that might give the same result as LiME and Volatility 2.3.

Keywords
Mobile Phone Forensics, Android Smartphone, Volatile Memory, Memory Forensics, LiME, Volatility

INTRODUCTION

According to Serber (2013), there are nine trends in the future that will highlight the significant increase in demand of Mobile Phone Forensics. Those trends are implementation of BYOD, enormous growth in mobile applications, tougher encryption in smartphones, difference platforms of smartphones, the upcoming of Windows 8 OS, the advancement of Mobile Devices as witnesses, the uncertainty of the regulatory and legislative landscape, the tremendous rise of mobile malware incidents and the increasing risk of data breaches via mobile device. Apparently, the trends will leverage the Mobile Phone Forensicsinto the same arena as the Computer Forensics in the future.

Currently, most research and analysis of Mobile Phone Forensics are focused on static data or non-volatile memory. Static data that resides on the Subscriber Identity Module (SIM), memory cards such as external SD Cards or emulated MultiMediaCard (eMMC) and the internal flash memory such as NAND flash memory(Alghafli, Jones, & Martin, 2011; Garfinkel, 2011; Hoog, 2011; Thackray, 2010). Hence, due to limited capacity of the non-volatile memory storage media, the volatile data or other information such as the application data, internet browsing data and instant messaging conversation histories are often not stored in the non-volatile storage media.

The issue will become more evident when dealing with the encryption keys, temporary file system and advanced mobile malware. This malware may not store their footprints on the non-volatile memory. Hence, the only option for the examiner is by conducting the investigation on the volatile memory. There are two techniques that available for the examiner to perform such investigation: traditional live response and memory forensics. Whilst according with Aljaedi et al.(2011), there is a disadvantage impact on traditional live response technique that might create a significant risk in losing of evidence. Therefore, this paper is proposing the use of memory forensics in incident response cases.

The Android OS has become more predominant in the global smartphones markets both in current and future time. The usage and adoption of Android OS is not bound only to smartphones but has been utilized into Netbooks, Ultra Mobile PCs, Printers, Gaming Devices, Home Appliances, GPS

receivers, E-readers, Home Audio, Media Players, TVs, Vehicles etc. As usual, if one Operating System becomes more prevalent, then it will lure more cyber-criminals to target it as an object to attack.

It became evident that the growth in usage and market share on Android OS is linear with the increasing of malicious software that is targeting the Android devices. Moreover, the advanced malware such as Mobile Trojan banking malware might only reside its data on the volatile memory storage, specifically on Android's smartphone. Therefore, the only viable option for a mobile phone examiner to acquire and analyse the incriminating evidence is by performing live memory forensic on volatile memory on Android smartphones.

The purpose of this work is to answer the question: what is the proper procedure and what are the free tools that are available for mobile phone forensic examiners when acquiring and analysing the memory forensics in Android smartphones? Additionally, the procedures and free tools should be applied with a forensically sound rule of thumb. Therefore, the evidence that has been acquired and analysed should fulfil the legal consideration. The work reveals the Linux Memory Extraction (LiME) application to acquire the memory from the Android devices and Volatility 2.3 for the analysis process. Furthermore, the work uses the image evidence from DFRWS Rodeo 2012 for the analysis process in order to show the capability of Volatility 2.3 for analysis purposes.

RELATED WORKS

There is research, works and presentations that reveal the procedures, techniques and tools for conducting mobile phone forensics in Android platform, but limited information about how to perform the acquisition and analysis of the volatile memory in it. Thing et al. (2010), Case (2012b), Sylve et al. (2012), Ligh (2013), and Macht (2013) are several researchers who revealed the memory forensics specifically in Android platform. The procedure for handling memory has become critical since the memory is volatile. Unproper procedures for acquiringthe memory from an Android smartphone would cause the loss of vital evidence forever. Brezinski and Killalea (2002) show that the every examiner should consider the order volatility in collecting evidence. Therefore, collecting the volatile memory should be a priority in any evidence acquisition procedure.

Sylve (2012) shows the importance of memory forensics in mobile phone forensics on his paper for several reasons. First, RAM dump provides both structured and unstructured information. Second, the use of strings on memory dump reveals on crucial evidence such as application data, fragments of communication and encryption keys. Third, it shows the kernel and application structures. Fourth, it reveals the previous processes, open files, network structure etc. According with Case (2011), memory forensics is vital to orderly recovery of runtime information.

Furthermore, works from Labs (2013) demonstrate the advancement of memory forensic in Android platform on Chuli Malware incident. This malware after infecting the Android's smartphone, could steal all the credential data such as a phone book and messages. The main goal of the work is investigating the malware with LiME to capture the physical RAM and analyse the image with Dalvik Inspector. This process found the evidence such as the phone number and IP Address of Command and Control (C&C) Server that belong to alleged cyber-criminal and sets of possible C&C commands.

Digital Forensic Research Workshop (DFRWS) has held the annual forensic conference and challenge since 2001. Both events have created much attention and new progression in the digital forensic community, especially in Memory Forensics. DFRWS 2012 has the challenge regarding with Memory Forensic in Android smartphone. The challenge is to answer seven questions that relate with botnet infection in a HTC EVO 4G Android device. This challenge will eventually support the development of LiME and Volatility version 2.3.

LiME (formerly known as DMD) is a Loadable Kernel Module (LKM), which allows the acquisition of volatile memory from Linux and Linux-based devices, such as those powered by Android. The tool supports acquiring memory either to the file system of the device or over the network. LiME is unique in that it is the first tool that allows full memory captures from Android devices. It also minimizes its interaction between *the userland* and *the kernelland* processes during acquisition, which allows it to produce memory captures that are more forensically sound than those of other tools designed for Linux memory acquisition(Forensics, 2012).

The Volatility Framework is a open collection of tools, implemented in Python under the GNU General Public License, for the extraction of digital artifacts from volatile memory (RAM) samples. The extraction techniques are entirely independent of the investigated system, but offer unprecedented visibility into the runtime state of the system. The purpose of the framework is introducing the techniques and complexities associated with extraction of digital artifacts from volatile memory samples and to provide a platform for further work into this exciting area of research.

WHERE AND WHAT IS THE DATA ON ANDROID DEVICES

Hoog (2011) divides the data on Android Devices for locating the data into two groups: Data at Rest and Data in Transit. First, there are five locations that could contain the data at rest: NAND-Flash Memory (non-volatile memory), memory card such as SD card and Embedded MultiMediaCard (EMMC). Removable media such as Universal Integrated Circuit Card (UICC) or known as SIM Card or SD Card are other locaters for Data at Rest. The last location of Data at Rest is on the Data Backups for the Android. Second, there are three locations that might be contained the data in transit: Network Service Provider, Physical RAM (volatile memory) and the Cloud.

There is potential for data evidence to be stored on NAND-Flash Memory and SD Card/eMMC. The examples of data such as SMS/MMS, call logs, voice mail, financial applications, personal email, web history, Google search history, YouTube, pictures and videos, geo-location, game history and interactions, corporate email and attachments, voice mail and faxes sent via email, user names, passwords and domain information, Wi-Fi Access Point with its information and passwords, calendar items, instant messenger and corporate files that are stored on the devices.

Potential data evidence found on UICC of SIM Card in forms of personal data such as SMS, EMS and address book/contact lists, secure key identification IMSI, ICCID, own number (dependent on service provider), service provider, LAI (local area identity – cell site information), allowed network information and PIN key encryption(Thackray, 2010).

Network service provider might give the data with legal consent such as call, SMS and MMS logs. Voice mail might be retained by the provider for several days depends on the regulation along with IMSI, ICC, IMEI, ESN and MSN. Internet traffic, emails, web activity, subscriber information, and PUK code are examples of data from provider. The cell site analysis and triangulation to identity the movement and location or users/devices are examples of data that might be collected from the provider(Thackray, 2010).

There could be various obtained data on physical memory (RAM) such as passwords, two-factor authentication, password reset security responses and data displayed in the application, but saved or cached to non-volatile storage (e.g. account numbers and balances). Examiners can recover process information on RAM such as process listing, memory maps, open files and networking information such as network interface information, open and listening sockets, and ARP tables.

TYPE OF TECHNIQUE FOR DATA ACQUISITION

According to Brothers (2007) and Hoog (2011), there are five techniques that available for acquiring the data from a mobile phone: manual extraction, logical extraction, physical extraction including JTAG technique, chip-off and macro read. All the techniques might be applied on the Android smartphone. There are several tools available for supporting such techniques with hardware and software-based tools. Commercial, freeware and open-source applicationsare available for data acquisition from mobile phones.

A presentation by viaForensics (2011) proposed various acquisition techniques such as SD card analysis using physical and simulated techniques, examining the backups, using Android Debug Bridge (adb), AFLogical App and viaExtract App. There are several compelling features in viaExtract such as SD Card imaging and Android Physical. The application images the SD card without removeing it from the device. This feature is essential since many Android devices still locates its SD card under the battery. This condition will create an obstacle if the examiner wants to dump the memory from physical (RAM) into the SD card. The examiner should image the SD card before the they want to dump the memory in the first place without removing the battery to access the SD card, otherwise the data on the memory will be lost.

Two methods are available for data extraction of running processes in Android smartphone. First, the examiner uses the Android Debug Bridge (ADB) to access a shell on the device and execute specific commands to dump the data, such as running processes, network connections and other device logs. Second, the examiner uses the Dalvik Debugging Monitor Server (DDMS), a tool that comes together with the Android SDK for dumping the contents of memory of a running process in the Android smartphone. Nevertheless, none of the methods described above has proven to dump the 'full contents' or all of the pages on the memory (Valenzuela, 2013).

Joe Sylve, et al. (2012) creates new kernel module for dumping the memory on Android devices, named Droid Memory Dumpstr (DMD) or known as Linux Extraction Memory (LiME). This technique could be categorised as physical extraction, because the the process performed by dumping the data on the memory. The main problem in this technique is an examiner needs to write the SD Card for dumping the memory. It seems that the technique is violating the common forensic rule of thumb, but the only non-volatile removable storage that could store the memory dump is the SD Card. Another issue is whether the SD Card is underneath the battery. Removing the battery could diminish the data on the physical memory, hence the solution is to tether Android Smartphone on USB mode, image the SD Card and finally dump the memory into SD Card(Joseph Sylve, 2012).

PROPER PROCEDURE FOR ACQUISITION

Procedure for Acquisition

L.Simao et al., 2011 proposed the procedure to acquire the data on Android Smartphone with regards to five conditions: turn on/off condition, availability of removable card, locked/unlocked condition, lockable/unlockable feature and super user feature. These conditions will affect the procedures, steps to be taken and tools to be used for acquiring the data by the examiner. The paper does not discuss the proper procedure for securing and handling the Android devices prior to data acquisition. The workflow of the procedure is shown on Figure 1.

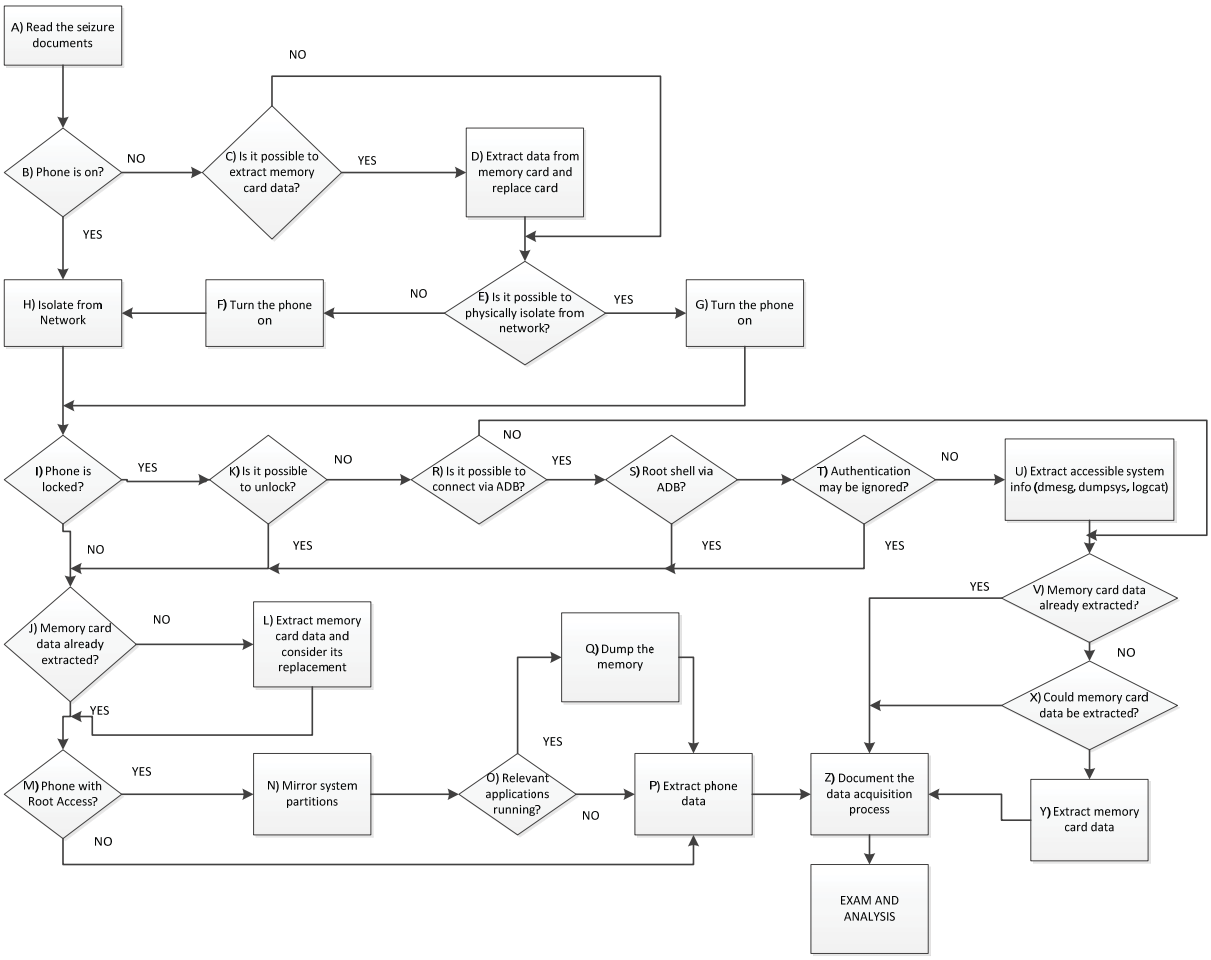

Figure 1: Workflow with the process of acquiring data with the Android operating system(de L, et al., 2011)

The work supports the procedure since its proceeding the extraction of memory card (SD Card/eMMC) before dumping the memory. Memory dumping process needs the media storage to store the image and the only option available is the memory card. Care should be taken that the process of dumping the memory should not turnoff the smartphone to access the memory card. This process turned out to be the only possible method if the smartphone uses a non-removable memory card such as eMMC. Therefore, the examiner should perform physical extraction upon the memory card before dumping the memory. In order to improve the procedure, the work directly suggest to dump the memory without asking whether the relevant application is running on the Android smartphone.

Linux Memory Extraction (LiME)

LiME was created to overcome the limitation that exist on the previous tools. Traditionally, memory captures on Linux is acquired by accessing the */dev/mem* device, which contained a map of the first gigabyte of RAM. This allowed acquisition of 896 MB of physical memory without the need to load code into the kernel. This approach did not work for machines with more than 896 MB of RAM. The */dev/mem* device has recently been disabled, due to security concerns on all major Linux distributions as it allowed for reading and writing of kernel memory.

In order to capture all physical memory, regardless of size, and to work around the loss of the */dev/mem* device, Ivor Kollar created fmem. According to Joe Sylve et al. (2012) the fmem module does not work on Android devices. Hence, they create the LiME specifically to overcome the issue. On the other hand, LiME still uses the loadable kernel module like fmem. The important feature of LiME is the capability of dumping the memory directly to SD card or over the network (network dump over adb). The target of the feature is minimizing an interaction between userand kernelland.

Moreover, it allows examiners to produce acquired memory that is more forensically sound than other tools designed for Linux memory acquisition.

Forensically Sound Process on LiME

According with Joe Sylve, et al., (2012) there are three reasons why LiME meets the basic forensic soundness standard. First, it attempts to minimize the impact on the target device when transferring data to and from it. Second, only a USB connection with the phone is needed for interaction. Once connected, only a single binary (the kernel module) needs to be transferred and executed to perform the acquisition. Third, loading of the module requires a minimal footprint as the LiME module is small (~70 KB) and requires few kernel functions to acquire memory.

Joseph Sylve (2012), has performed a test for the soundness of DMD/LiME by comparing the RAM Snapshot using the emulator with RAM image that have acquired with DMD-both by TCP and direct SD Card and fmem. The result on Table 1 shows the percentage of identical pages. This test shows the effectiveness of LiME in capturing the pages in physical memory (RAM) and also supporting the claim of forensic soundness on LiME.

Table 1: Percentage of Identical Pages on DMD and Fmem(Joseph Sylve, 2012)

Method	***Total Number of Pages***	***Number of Identical Pages***	***Percentage of Identical Pages***
DMD (TCP)	131072	130365	99.46%
DMD (SD Card)	131072	129953	99.15%
fmem (SD Card)	131072	105080	80.17%

504ENSICS (2013) reveals two types of acquisition technique on LiME. First is the acquisition of memory over a TCP connection. On this technique, the examiner should copy the kernel module to the phone's SD card using ADB. All network data is transferred via USB. After that, the target device should listen to specified TCP Port (port 4444) and then the examiner should connect to the device from the host computer. Whilst the socket is connected, the kernel module will automatically send the acquired RAM image to the host device. Second is the acquisition a memory dump via the phone's SD card. This option will be taken if the examiner wants to make sure no network buffer is overwritten. In order to execute the process, the examiner should image the SD card to save unallocated space. After that, the examiner should tethering the device to a Linux machine and activating USB storage exposes a */dev/sd?. The* device can be imaged using traditional means (eg. using dd on the Linux machine). If the SD can be removed, then the examiner could remove the SD card to acquire the memory dump, but if could not be removed, then the examiner could use ADB to transfer the memory dump to investigator's machine.

Gain Root Privileges

Every examiner before they decided to use the tools such as ADB and LiME should be aware that they have to gain root privileges on the Android devices. The root privileges in Android device is not enabled by default due to a security mechanism. Therefore, the first task for every examiner before they execute the ADB and LiME or any software-based physical technique is to gain root privileges. On the other hand, gaining root privileges sometimes is not an easy task. The techniques for root privileges will vary not only for each manufacturer and device but for each version of Android and the Linux kernel in use(Hoog, 2011).

ANALYSIS PROCESS OF THE MEMORY IMAGE WITH VOLATILITY 2.3

According with Case (2012a) and Volatility (2013), there are at least nine analysis capabilities on Volatility release 2.3. The analysis capabilities include iomem and limeinfo, processes, kernel objects, memory caches, networking, mounted filesystems, files in memory, rootkit detection and specific analysis. There is new features on the latest realease of Volatility such as new ARM address space to support memory dumps from Linux and Android devices on ARM, added plugins to scan linux

process and kernel memory with yara signatures, dump LKMs to disk, and check TTY devices for rootkit hooks and added plugins to check the ARM system call and exception vector tables for hooks. The Volatility release 2.3 also supports the Address Space with *LimeAddressSpac*e to be able to analyse the acquired memory with LiME(Volatility, 2013). Appendice 1 shows the detail information regarding detailed analysis capability.

In order to show the analysis capability of Volatility, the work documented several commands for analysing the image evidence from DFRWS Rodeo 2012 name "Evo4GRodeo.lime."

Iomem and limeinfo

The plugins will provide output similar to */proc/iomem* whichshows the physical addresses currently reserved for IO devices like PCI and video card memory.

```
aph4nc@aph4nc:~/Volatility 2.3$ python vol.py -f
/media/IMATION/DFRWSRodeo2012/Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_iomem
Volatile Systems Volatility Framework 2.3_alpha
PCI mem                                0x0                  0xFFFFFFFF
msm_hdmi.0                             0x2B00000            0x2EFFFFF
kgsl_phys_memory                       0x3700000            0x39FFFFF
kgsl                                   0x3700000            0x39FFFFF
  ram_console                          0x3A00000            0x3A3FFFF
msm_panel.1                            0x3B00000            0x3DFFFFF
System RAM                             0x20000000           0x2E7FFFFF
   Kernel text                         0x20038000           0x204D5FFF
   Kernel data                         0x204D6000           0x2062F617
System RAM                             0x30000000           0x33FFFFFF
System RAM                             0x34000000           0x3B5FFFFF
  kgsl_reg_memory                      0xA0000000           0xA001FFFF
kgsl                                   0xA0000000           0xA001FFFF
  msm_serial_hs_bcm.0                  0xA0200000           0xA0200FFF
msm_sdcc.1                             0xA0300000           0xA0300FFF
  msm_sdcc.2                           0xA0400000           0xA0400FFF
msm_sdcc.3                             0xA0500000           0xA0500FFF
  msm_hsusb                            0xA0800000           0xA0801000
spi_base                               0xA1200000           0xA1200FFF
  msm_i2c.0                            0xA9900000           0xA9900FFF
msm_i2c                                0xA9900000           0xA9900FFF
mdp                                    0xAA200000           0xAA2EFFFF
msm_mddi.0                             0xAA600000           0xAA600FFF
```

Processes Shows

The plugins show per-processing listings, child/parent process relationship, opened files and memory maps.

```
C:\Volatility 2.3>python vol.py -f c:\Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_psaux -p 1873
Volatile Systems Volatility Framework 2.3_alpha
Pid    Uid    Arguments
1873   0      insmod /sdcard/lime-evo.ko path=tcp:4444 format=lime          Sun, 05 Aug 2012 10:33:09 +0000
```

```
C:\Volatility 2.3>python vol.py -f c:\Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_proc_maps -p 1
Volatile Systems Volatility Framework 2.3_alpha
 0x8000- 0x20000 r-x                0     0: 1                    34 /init
 0x20000- 0x21000 rw-               98304 0: 1                    34 /init
 0x21000- 0x34000 rw-               0     0: 0                    0 [heap]
0x40000000-0x40001000 r--           0     0: 0                    0
0x40001000-0x40011000 rw-           0     0:10                    315 /dev/__properties__
0xbe982000-0xbe9a4000 rw-           0     0: 0                    0 [stack]
```

Kernel Objects

The plugins show the list of kernel modules, debug buffer and memory caches.

```
aph4nc@aph4nc:~/Volatility 2.3$ python vol.py -f /media/IMATION/DFRWSRodeo2012/Evo4GRodeo.lime --
profile=LinuxEvo4Gx86 linux_lsmod
Volatile Systems Volatility Framework 2.3_alpha
lime 8014
bcm4320 228359
```

Memory Caches

The plugin will mimics */proc/slabinfo* on a running machine, gather files from the dentry cache, gather tasks from the *kme_cache*, recover the routing cache from memory, recovers packets from the *sk_buff kemem_cache* and gather VMAs from the *vm_area_struct cache*

Networking

The plugin will recover the ARP table, replicate ifconfig output, recover the routing cache, replicate netstat output and recover per-socket packet queues.

```
aph4nc@aph4nc:~/Volatility 2.3$ python vol.py -f
/media/IMATION/DFRWSRodeo2012/Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_ifconfig
Volatile Systems Volatility Framework 2.3_alpha
Interface              IP Address                MAC Address               Promiscous Mode
---------------- -------------------- ------------------ ---------------
lo                     127.0.0.1                 00:00:00:00:00:00         False
dummy0                 0.0.0.0                   00:00:00:00:00:00         False
ifb0                   0.0.0.0                   00:00:00:00:00:00         False
ifb1                   0.0.0.0                   00:00:00:00:00:00         False
rmnet0                 0.0.0.0                   00:00:00:00:00:00         False
rmnet1                 0.0.0.0                   00:00:00:00:00:00         False
rmnet2                 0.0.0.0                   00:00:00:00:00:00         False
usb0                   0.0.0.0                   00:00:00:00:00:00         False
sit0                   0.0.0.0                   00:00:00:00:00:00         False
ip6tnl0                0.0.0.0                   00:00:00:00:00:00         False
eth0              50.94.125.176                  00:00:00:00:00:00         False
```

```
aph4nc@aph4nc:~/Volatility 2.3$ python vol.py -f
/media/IMATION/DFRWSRodeo2012/Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_arp
Volatile Systems Volatility Framework 2.3_alpha
[::                                          ] at 00:00:00:00:00:00          on lo
[50.94.125.1                                 ] at 00:90:fb:34:af:ca          on eth0
```

Mounted Filesystem

The plugin will lists mounted file systems including the mount flags and gather mounted *fs/devices* from *kmem_cache*

```
aph4nc@aph4nc:~/Volatility 2.3$ python vol.py -f /media/IMATION/DFRWSRodeo2012/Evo4GRodeo.lime --
profile=LinuxEvo4Gx86 linux_mount
Volatile Systems Volatility Framework 2.3_alpha
tmpfs                                                    /app-cache
tmpfs              rw,relatime
tmpfs                /mnt/obb
tmpfs               rw,relatime
tmpfs               /mnt/asec
tmpfs               rw,relatime
/dev/block/vold/179:1              /mnt/secure/asec/.android_secure   vfat
rw,relatime,nosuid,nodev,noexec
/dev/block/mtdblock5               /cache
yaffs2             rw,relatime,nosuid,nodev
/dev/block/mtdblock6               /data
yaffs2             rw,relatime,nosuid,nodev
none                               /acct
cgroup             rw,relatime
tmpfs                              /mnt/sdcard/.android_secure      tmpfs
ro,relatime
htcfs                              /data/htcfs
fuse               rw,relatime,nosuid,nodev
/dev/block/vold/179:1              /mnt/sdcard                      vfat
rw,relatime,nosuid,nodev,noexec
none                                        /dev/cpuctl
cgrou              rw,relatime
tmpfs                              /dev
tmpfs              rw,relatime
```

```
devpts                                                  /dev/pts
devpts          rw,relatime
/dev/block/mtdblock4            /system                 yaffs2
ro,relatime
sysfs                                   /sys
sysfs           rw,relatime
/sys/kernel/debug               /sys/kernel/debug               debugfs
rw,relatime
proc                                    /proc
proc            rw,relatime
```

Files in Memory

The plugin shows all files and entire file systems that can be recovered directly from memory and shows unencrypted contents of encrypted files.

```
C:\Volatility 2.3>python vol.py -f c:\Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_tmpfs -L
Volatile Systems Volatility Framework 2.3_alpha
1 -> /app-cache
2 -> /mnt/obb
3 -> /mnt/asec
4 -> /mnt/sdcard/.android_secure
5 -> /dev
```

Specific Analysis

The plugin basically switch from the kernel land on the previous analysis capability into userland to analyse the Android applications. The aim is to analyse the specific application to focus on the data structure of one process.

```
aph4nc@aph4nc:~/Volatility 2.3$ python vol.py -f
/media/IMATION/DFRWSRodeo2012/Evo4GRodeo.lime --profile=LinuxEvo4Gx86 linux_check_creds
Volatile Systems Volatility Framework 2.3_alpha
PIDs
--------
139..97
```

CONSIDERATIONS ON LiME AND VOLATILITY 2.3

Customized Loadable Kernel Module on LiME

The Linux kernel uses a security mechanism called module verification(Joe Sylve, et al., 2012). It is intended to prevent the kernel from accepting incompatible or possibly malicious code to being inserted into the operating system. This feature makesit impossible to load a general module for every type of Android smartphone. There is no option available to load module in a kernel-agnostic way for generic solution approach. Apparently, the only approach is available is to create a pool of precompiled modules. Every module in the pool is compiled against a specific kernel; basically there is one module for each device and Android version. Therefore, every examiner should build the customized Loadable Kernel Module (LKM) for each smartphone and Android version in order to acquire physical memory with LiME(Macht, 2013; Valenzuela, 2013).

Creating the Profile on Volatility 2.3

According with Macht (2013), a profile needs to be created which can be passed to Volatility on the command line, before an examiner wants to analyse the memory image. A Volatility profile is a set of *vtype definitions* and optional symbol addresses. The Volatility 2.3 (Alpha version) only has one profile for Android devices: *LinuxEvo4Gx86*. On the other hand, there are several profiles for Microsoft Operating System. There are two main reasons for that. First, Volatility development has started from a Microsoft Windows point of view and Android support is quite new, consequently there are less corresponding profiles are available by default. Second, there are many different flavours of different kernels available in the Android market. Moreover, every single vendor such as HTC, Samsung or LG usually has multiple devices running a different kernel. The problem with that is

each kernel needs its own Volatility profile. The situation is identical with usage of LiME: every examiner needs to create an own profile for every Android smartphone for forensic investigation.

CONCLUSION AND FUTURE WORK

The work supports the procedure for acquiring the data evidence in Android smartphones with regard to the order of volatility. First of all, the procedure should image the SD card in order to create space to store the memory dump. Every examiner could consider using LiME as the tool for conducting the acquisition process. As the tool, LiME deliberates the forensically sound process and could capture almost all of the pages on the physical memory.

Volatility 2.3 shows the analysis capability for investigating the acquired image evidence with LiME. Several plugins and commands have been developed to gather the specific purposes. The work has demonstrated the analysis upon the evidence image from DFRWS Rodeo 2012. The analysis result shows how the application could answer the questions on the forensic challenge.

Despite the benefit that would be created from LiME and Volatility 2.3, there are two concerns regarding with its usage. First, the examiner should gain the root privileges before executing LiME. Sometimes this process is quite difficult task and will depend on the manufacturer, version ofthe Android OS and Linux Kernel. Second, there is no generic solution or approach for every type of Android smartphone. Examiners should create customized LKM for each Android device in order to use LiME. The same problem is persists on Volatility 2.3, the examiner should create a specific profile for specific Android smartphones to be able to analyse the acquired image.

Nevertheless, both applications are offering the best solution that no other options could offer for the digital forensics community. Especially in acquiring and analyzing the volatile memory inthe Android smartphone. The development of both applications is still growing and needs support from community in order to create more 'user-friendly' options for both applications.

Future works could be applied to investigate the Mobile Trojan Banking Malware Incidents, smartphone encryption keys and temporary file systems in Android smartphones. Hence, it would support the development of LiME and Volatility as a contribution to the digital forensic community.

REFERENCES

504ENSICS. (2013) LiME-Linux Memory Extractor: Instructions v1.2. 504ENSICS, LLC.

Alghafli, K. A., Jones, A., & Martin, T. A. (2011). Guidelines for the digital forensic processing of smartphones.

Aljaedi, A., Lindskog, D., Zavarsky, P., Ruhl, R., & Almari, F. (2011). *Comparative Analysis of Volatile Memory Forensics: Live Response vs. Memory Imaging.* Paper presented at the Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom).

Brezinski, D., & Killalea, T. (2002). *Guidelines for Evidence Collection and Archiving (RFC 3227)*. Retrieved from http://www.hjp.at/doc/rfc/rfc3227.html.

Brothers, S. (2007). *iPhone Tool Classification*. Retrieved from http://www.appleexaminer.com/files/iPhone_Levels.pdf

Case, A. (2011). Memory Analysis of the Dalvik (Android) Virtual Machine. Retrieved from

Case, A. (2012a). Android Forensics with Volatility and LiME: YouTube.

Case, A. (2012b). Android Memory Forensics: DFIR Online.

de L, S., Morum, A., Sicoli, F. C., de Melo, L. P., & de Sousa Junior, R. T. (2011). Acquisition of digital evidence in android smartphones.

Forensics, L. (2012). LiME Forensics, from https://code.google.com/p/lime-forensics/)

Garfinkel, S. L. (2011). Android Forensics. Retrieved from http://simson.net/ref/2011/2011-07-12%20Android%20Forensics.pdf

Hoog, A. (2011). *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*: Syngress.

Labs, E. (2013). Android Application (Dalvik) Memory Analysis & The Chuli Malware, 2013, from http://www.504ensics.com/android-application-dalvik-memory-analysis-the-chuli-malware/

Ligh, M. H. (2013). Android Memory Forensic, from https://code.google.com/p/volatility/wiki/AndroidMemoryForensics

Macht, H. (2013). *Live Memory Forensics on Android with Volatility.* Diploma, Friedrich-Alexander Universitat, Nurnberg.

Serber, R. (2013). Industry Experts Identify Mobile Forensics Trends, 2013, from http://www.s-ox.com/dsp_getnewsDetails.cfm?CID=2969

Sylve, J. (2012). Android Mind Reading: Memory Acquisition and Analysis with DMD and Volatility. Retrieved from

Sylve, J., Case, A., Marziale, L., & Richard, G. G. (2012). Acquisition and analysis of volatile memory from android devices. *digital investigation, 8*(3–4), 175-184. doi: http://dx.doi.org/10.1016/j.diin.2011.10.003

Thackray, J. (2010). *Mobile Phone Forensic Investigation: Recognition, Recovery, Preservation and Analysis Techniques*: Thackray Forensics Ltd.

Thing, V. L. L., Ng, K.-Y., & Chang, E.-C. (2010). Live memory forensics of mobile phones. *digital investigation, 7, Supplement*(0), S74-S82. doi: http://dx.doi.org/10.1016/j.diin.2010.05.010

Valenzuela, I. (2013). Acquiring volatile memory from Android based devices with LiME Forensics, Part I, 2013, from http://blog.opensecurityresearch.com/2012/04/acquiring-volatile-memory-from-android.html

viaForensics. (2011). Android Forensics: Background, techniques and analysis tools Retrieved from

Volatility. (2013). Volatility Release 2.3, from https://code.google.com/p/volatility/wiki/Release23

Volatility. (-). Volatility Framework.

APPENDICES 01: Analysis Capability, Command and the Purpose

No	Analysis Capability	Command and the Purpose
1	IOMEM and Limeinfo	• linux_iomem : Provides output similar to /proc/iomem • linux_cpuinfo : Prints info about each active processor
2	Processes	• linux_pslist : Gather active tasks by walking the task_struct->task list • linux_pstree : Shows the parent/child relationship between processes • linux_psaux : Gathers processes along with full command line and start time • linux_lsof : Lists open files • linux_memmap : Dumps the memory map for linux tasks • linux_proc_maps : Gathers process maps for linux • linux_dump_map : Writes selected memory mappings to disk • linux_pidhashtable : Enumerates processes through the PID hash table
3	Kernel Objects	• linux_dmesg : Gather dmesg buffer • linux_lsmod : Gather loaded kernel modules
4	Memory Caches	• linux_slabinfo : Mimics /proc/slabinfo on a running machine • linux_dentry_cache : Gather files from the dentry cache • linux_pslist_cache : Gather tasks from the kmem_cache • linux_route_cache : Recovers the routing cache from memory • linux_sk_buff_cache : Recovers packets from the sk_buff kmem_cache • linux_vma_cache : Gather VMAs from the vm_area_struct cache
5	Networking	• linux_ifconfig : Gathers active interfaces • linux_netstat : Lists open sockets • linux_arp : Print the ARP table • linux_pkt_queues : Writes per-process packet queues out to disk
6	Mounted Filesystem	• linux_mount : Gather mounted fs/devices • linux_mount_cache : Gather mounted fs/devices from kmem_cache
7	Files in Memory	• linux_tmpfs : Recovers tmpfs filesystems from memory • linux_find_file : Recovers tmpfs filesystems from memory
8	Rootkit Detection	• linux_psxview : Find hidden processes with various process listings • linux_check_afinfo : Verifies the operation function pointers of network protocols (finds hooks in structure that control displaying of network connections (netstat)) • linux_check_fop : Check file operation structures for rootkit modifications (finds hooks in structures that deal with file opening, reading and writing) • inux_check_modules : Compares module list to sysfs info, if available (finds hidden kernel modules by xrefing with sysfs)
9	Specific Analysis	• zygote : linux_pslist | grep zygote • atoms : Print session and window station atom tables • atomscan : Pool scanner for _RTL_ATOM_TABLE • clipboard : Extract the contents of the windows clipboard • eventhooks : Print details on windows event hooks • gahti : Dump the USER handle type information • gditimers : Print installed GDI timers and callbacks • linux_bash : Recover bash history from bash process memory • linux_check_creds : Checks if any processes are sharing credential structures • linux_check_idt : Checks if the IDT has been altered • linux_check_syscall : Checks if the system call table has been altered • messagehooks : List desktop and thread window message hooks • patcher : Patches memory based on page scans • sessions : List details on _MM_SESSION_SPACE (user logon sessions) • userhandles : Dump the USER handle tables • windows : Print Desktop Windows (verbose details) • wintree : Print Z-Order Desktop Windows Tree