

2015

## Comparison of Live Response, Linux Memory Extractor (LiME) and Mem tool for acquiring android's volatile memory in the malware incident

Andri Heriyanto  
*Security Research Institute, Edith Cowan University*

Craig Valli  
*Security Research Institute, Edith Cowan University*

Peter Hannay  
*Security Research Institute, Edith Cowan University*

Follow this and additional works at: <https://ro.ecu.edu.au/adf>



Part of the [Information Security Commons](#)

---

DOI: [10.4225/75/57b3f143fb884](https://doi.org/10.4225/75/57b3f143fb884)

13th Australian Digital Forensics Conference, held from the 30 November – 2 December, 2015 (pp. 5-14), Edith Cowan University Joondalup Campus, Perth, Western Australia.

This Conference Proceeding is posted at Research Online.

<https://ro.ecu.edu.au/adf/144>

# COMPARISON OF LIVE RESPONSE, LINUX MEMORY EXTRACTOR (LiME) AND MEM TOOL FOR ACQUIRING ANDROID'S VOLATILE MEMORY IN THE MALWARE INCIDENT

Andri Heriyanto<sup>1,2</sup>, Craig Valli<sup>2</sup>, Peter Hannay<sup>1,2</sup>

<sup>1</sup>School of Computing and Security Science, <sup>2</sup>Security Research Institute  
Edith Cowan University, Perth, Australia  
aheriyanto@our.ecu.edu.au, c.valli@ecu.edu.au, p.hannay@ecu.edu.au

## Abstract

*The increasing use of encryption and obfuscation within the malware development arena has necessitated the use of volatile memory acquisition on smartphone platforms. Current smartphone forensics research lacks a well-formulated process for the acquisition of volatile memory. This research evaluates and contrasts three differing tools for acquisition of volatile memory from the Android platform: Live Response, Linux Memory Extractor (LiME) and Mem Tool. Evaluation is conducted through practical examination during the analysis of an infected device. The results demonstrate a combination of LiME and the Volatility Framework provides the most robust findings. Complexities due to the nature of LiME prevent it from being a feasible tool for real-world use. In contrast, Live Response is found to be reliable and applicable to real-world scenarios. In all evaluations, it was found that the forensic practitioner must take care to understand and be aware of the impact to data stored within volatile memory caused by the acquisition process.*

## Keywords

Android, Smartphone Forensics, Live Response, Linux Memory Extractor (LiME), Mem tool, Volatility Framework, Volatile memory, Backdoor.AndroidOS.Obad

## INTRODUCTION

According to Hoog (2011), there are two classifications for storing the data on an Android device: data at rest and data in transit. Data at rest is stored data in several storage media such as internal flash memory (NAND-flash or NOR-flash Memory), memory card (SD card), embedded multimedia card (eMMC), Universal Integrated Circuit Card (UICC), and data backups. Furthermore, data on Android devices can be classified as data in transit if it is stored in a Network Service Provider (NSP) and as Random Access Memory (RAM) as a store of volatile memory.

Many well-known computer forensic guidelines such as Justice (2008), Group (2002) and Cichonski, Millar, TimGrance, and Scarfone (2012) propose the importance of prioritising the acquisition of memory based on its volatility. In contrast, a few well-known of the mobile phone forensics guidance such as (SWGDE) (2013) and Ayers, Brothers, and Jansen (2014) do not state the same approach. As a consequence, many smartphone forensics workflows or approaches are still focusing on non-volatile memory acquisition.

As the Android smartphone become ubiquitous, there is an immense growth of the malware that mainly targeted the Android devices. According to Cisco (2014), ninety-nine percent of all mobile malware in 2013 targeted Android devices. Android users also have the highest encounter rate (71%) with all form of web-delivered malware. On the other hand, there is another perspective for the forensic community in regards with advanced malware. Nasim, Aslam, Ahmed, and Naeem (2015) reveals one code obfuscation technique called packing that enables a malware author to mangle an executable is such a way that it becomes difficult for an analyst to reverse engineer. In addition, Burdach (2006) reveals rootkits and worms that store their code only in volatile memory. This obfuscation technique has rendered the traditional forensics approach of offline analysis insufficient. Thus, volatile memory acquisition in Android devices suspected to be infected with malware is highly recommended.

The paper aims to address the following research questions:

- What are the proper forensics approaches for acquisition and analysis of volatile memory on Android devices in exposed with Malware incidents?
- What is the advantages and disadvantages of three different approaches: Live Response, Mem Tool, and LiME/Volatility for memory forensics on Android devices?
- What is the final result from comparison of the analysis' output from the three different approaches?

- How is the feasibility evaluation for each approach in real case scenario?

## RELATED WORKS

Aljaedi et al. (2011) have performed comparative analysis in Live Response and memory imaging on Linux desktop environment. The research reveals the advancement of memory imaging in comparison with a Live Response. It stated that Live Response approach does not include hidden and terminated processes. Moreover, the average percentage of changed pages during Live Response is higher than memory imaging as shown in Figure 1. This data suggests memory imaging had less of an impact on volatile data.

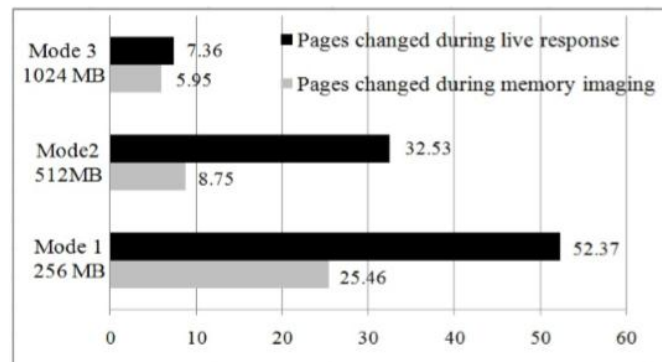


Figure 1: Comparison of pages changed during Live Response and memory imaging (Aljaedi et al., 2011)

Sylve, Case, Marziale, and Richard (2012) presented Linux Memory Extractor (LiME) as a means to obtain complete captures of volatile memory along with a subsequent analysis of that data in both userland and the kernel. The tool offers two significant advancements: forensic soundness of acquisition approach and the completeness of pages acquisition in comparison with other tools. Wächter (2015) concluded that memory forensics with LiME is not feasible for law enforcement purposes. He found seven factors that might deter the usage of LiME: identifying the model, identifying Operating System (OS), root exploit, lock screen, availability or sources, kernel configuration and evidence erosion.

## LIVE RESPONSE, LINUX MEMORY EXTRACTOR (LiME) AND MEM TOOL

### Live Response

Live Response is a method of acquiring volatile data, whereby the suspect operating system is acquired for potential evidence such as the status of open network connections, the status of open files, the system's conception of the current date and time, and any other forensically relevant volatile data. However, it is well understood that acquiring volatile data is inherently problematic because it relies on a potentially compromised, and thus untrusted OS and that therefore the acquired evidence could be contaminated during this information gathering process, and because it is not a repeatable process (Nagy et al., 2014).

According to Case (2012a), malware can trivially default live analysis, because it is running tools built into the OS to gather volatile data. Hence, it is possible that malware can hide its presence to any and all userland tools and even in-kernel monitors. Moreover, many advanced malware only operates in the volatile memory; they might never touch the non-volatile memory, and all their network traffic has been encrypted. As a conclusion, the memory acquisition becomes the need and critical for handling the malware incident.

### Linux Memory Extractor (LiME)

Some researchers have proposed tools and procedures to acquire volatile memory to overcome the limitations on the Live Response. Leppert (2012) proposes using Dalvik Debug Monitor Service (DDMS) in the Android Software Development Kit (SDK) for acquiring the heap dump as a dumping file of the volatile memory in Android devices. However, according to Macht (2013), Leppert's approach contains some flaws. At first, the approach only feasible for the debugging application process. Hence, it only feasible for an application that has been prepared for debugging. Secondly, the acquisition process cannot obtain comprehensive artefacts such as complete class objects or binary data. As a conclusion, the following analysis will be limited since it came from the limited data acquisition.

Thing, Ng, and Chang (2010) develop the acquisition tool called **memgrab**. It aims to dump the process' memory in the Android devices. A workflow of the tools consists of two consecutive steps. First, the tool will trace the process memory and acquire it based on the */proc/pid/maps* and */proc/pid/mem* files. Second, the tool deploys the Process Trace (ptrace) system call, which enable the locating of the process by monitoring its execution, as well as gaining access to its address space. Furthermore, the researchers investigated the cached data and the volatile memory persistence in Android smartphone.

Sylve et al. (2012) reveal several issues with Thing's research. First, significant information such as in-kernel structures, networking information, and others are not analysed. Second, the memgrab requires many interactions with the live system that potentially change the data. This last issue creates a significant concern regarding with forensically sound process requirement. Hence, they propose Android memory acquisition module called Droid Mobile Dumpster (DMD) or known as Linux Memory Extractor (LiME) to overcome all the limitations. The research claimed that LiME can dump the address memory over TCP and to an Android device's SD card. Moreover, it offers thoughts on the forensic soundness of the approach. LiME works in three consecutive steps. First is parsing the kernel's *iomem\_resource* structure to learn the physical memory address ranges of system RAM. Second is performing physical to virtual address translation for each page of memory. Third is reading all pages in each range and writing them to either a file (typically on device's SD card) or a TCP socket.

There is a significant challenge on LiME especially for loading the module into the kernel. There is a security mechanism for the kernel in the Android OS that preventing incompatible module or suspected malicious code to be loaded into the OS. Although there were attempts to bypass the security mechanism, none of the attempts is perfect. Also, there is no research for creating a module in a kernel-agnostic feature (Sylve et al., 2012).

## Mem Tool

Tamma and Tindall (2015) propose the use of Mem Tool as an alternative tool to dump the process files on the RAM. Mem tool is an executable binary that needs to push to the device and with netcat tool-the mem is used to dump the selected processes based on its PID. Moreover, it claimed has the capability to read the entire RAM or target specific. There is no current research or discussion that shown the risk and benefit that might occur from the Mem tool. Based on the experiment, the Mem tool is relied on the current processes similar with the Live Response and has many interactions with the live data. Therefore, it depends on the running OS and potentially can change the data on the RAM. However, the tool offers a repeatability function by dumping the running processes.

## RESUME OF ADVANTAGES AND DISADVANTAGES

Based on the related research as described above, the research resumes the requirement including the advantages and disadvantages of each approach. Every examiner should understand this conditions to justify their selected procedure and finally can minimise the risk that might occur from a selected approach.

*Table 1: Advantages and Disadvantages of Live Response, LiME and Mem tool*

No	Variables	Live Response	LiME	Mem Tool
1	<b>USB debugging requirement</b>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>
2	<b>Root access requirement</b>	<i>No</i>	<i>Yes</i>	<i>Yes</i>
3	<b>Kernel agnostic</b>	<i>Yes</i>	<i>No</i>	<i>Yes</i>
4	<b>Forensically soundness</b>	<i>Low</i>	<i>High</i>	<i>Medium</i>
5	<b>Detection ability for cached and terminated process</b>	<i>No</i>	<i>Yes</i>	<i>No</i>
6	<b>Impact on current Live Memory</b>	<i>High</i>	<i>Low</i>	<i>Medium</i>
7	<b>Implementation complexity</b>	<i>Low</i>	<i>High</i>	<i>Medium</i>
8	<b>Completeness (Pages Acquired)</b>	<i>~</i>	<i>High</i>	<i>~</i>
9	<b>Analysis result</b>	<i>Limited</i>	<i>Robust</i>	<i>Limited on Strings Output</i>
10	<b>Repeatability</b>	<i>No</i>	<i>Yes</i>	<i>Yes</i>

## RESEARCH METHOD

The research conducted an experiment for supporting the comparison analysis of the three approaches. The research method for the experiment shown in Figure 2. There are two scenarios for the experiment. The first scenario uses LiME as the first tool to acquire the RAM and followed by Live Response to acquire the volatile data based on the running OS. The second scenario uses Mem tool as the initial tool to acquire the running processes and followed by Live Response. There is an additional procedure for comparing two output of the Live Response: before and after the malware infection. The purpose of this comparison is acquiring the detail information of all changes that have been made on the device by the malware.

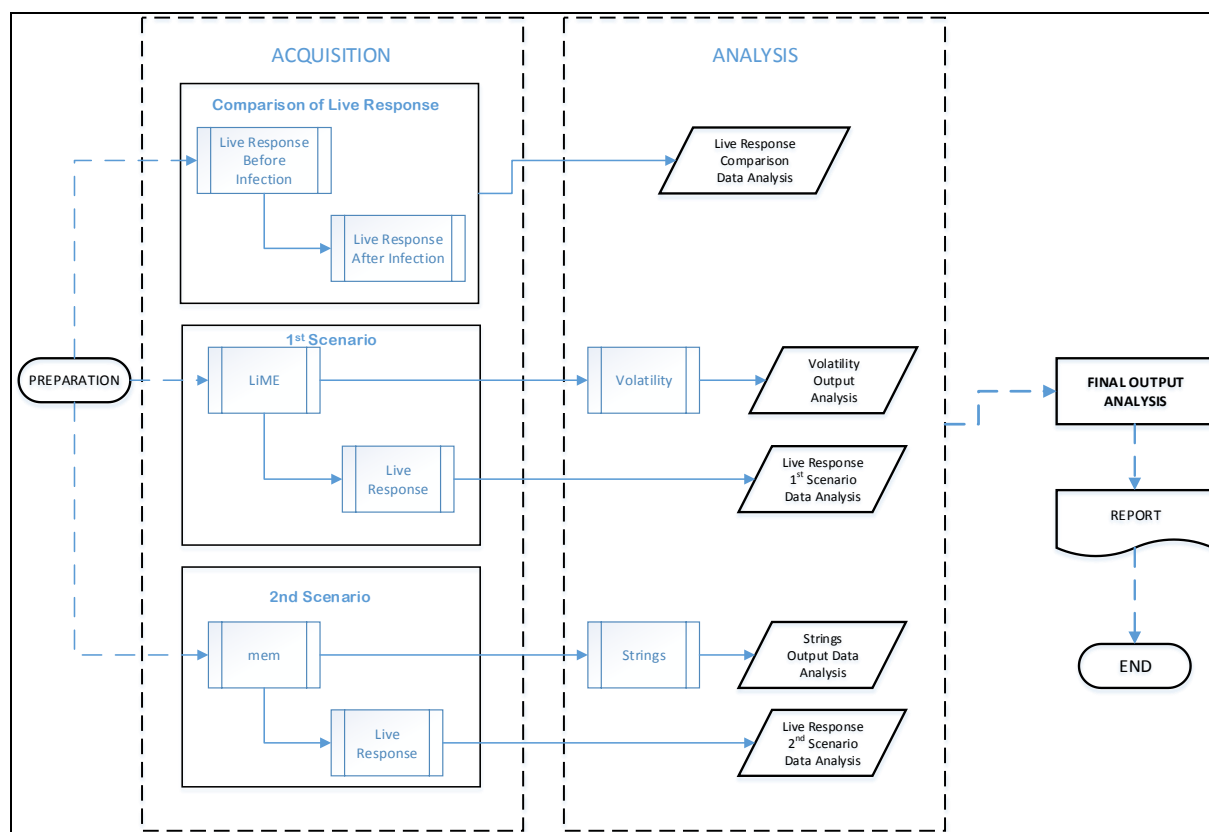


Figure 2: Research Method

Forensic Workstation: Linux Ubuntu 14.04 LTS 64-bit, RAM: 8 GB, CPU: Intel Core i7 CPU Q720 @ 1.60GHz x 8. Testbeds: Android Virtual Device/AVD with the configuration: CPU:armeabi-v7a, Target: Android 5.0.1(API level 2), Device Name: Google Nexus S with skin 480x800, SDCard: 2GB, RAM:1024MB, VM heap size: 64, Data Partition size: 512MB. The research used the AVD with the same configuration for all examination's scenario.

Forensic tools:

- a. Android Debug Bridge (ADB) for the Live Response acquisition;
- b. Linux Memory Extractor (LiME) for memory acquisition and Volatility for analysing the acquired memory;
- c. Mem tool for dumping the processes and Autopsy 3.1.0 for analysing the dumped files;
- d. Kdiff3 for comparing Live Response acquisition before and after infection.

## Malware Sample

### Overview

The technical name for the malware is Backdoor.AndroidOS.Obad. The malware author used a previously unknown vulnerability in Android that allowed the malware to gain extended device administrator privileges. All strings in the executable file of the malware were encrypted, and the code was obfuscated. It has multi-functional capability such as sending SMS to premium-rate numbers, downloading other malware programs, install them on the infected device and/or sending them further via Bluetooth. This malware sample does not have a graphical user interface and operates as a background service (Sims, 2013; Unuchek, 2013).

### *Anti-Analysis Techniques and Behaviour*

OBAD is an emulator-aware malware, which increases the complexity of analysis. The malware looks for the “Android.os.build.MODEL” value throughout the code and exits if it matches with the emulator's model. The malware can only be run in an emulator after patching several of checks. The malware sends information including the victims IMEI number, operator name, MAC address of the Bluetooth device and the prepaid card account balance of the user in an encrypted JSON form. In response, the malware author sends another JSON object that contains configuration information alongside the commands to be executed.

## PROCEDURES IN LIVE RESPONSE

The research following the guidance from Hubbard (2013), Kercher (2013) and Nagy et al. (2014) that includes specific ADB commands for gathering the Live Response information. Since Live Response can overwrite the unallocated data in the RAM, the examiner should be aware of their actions to minimise the impact on the system.

## PROCEDURES IN LIME AND VOLATILITY

The procedure for LiME follows the guidance from 504ensicsLabs (2014), Case (2012b), Valenzuela (2013) and Wächter (2015). The research found that the thesis from Wächter is the most feasible guideline in comparison with the previous guidelines available. For analysing the LiME file, the research uses the instruction and guidelines from Volatility (2013) and Ligh et al. (2014).

## PROCEDURES IN MEM TOOL

First of all, the Mem tool and NC tool should be installed on the */dev/* directory. It is *tmpfs* file directory that means the directory only persisted as long as the device has not been rebooted. The main purpose of this procedure is to avoid both tools to overwrite potential evidence on non-temporary file system's directory.

According to with the documentation file of the tool, if the examiner set O for the PID then it can dump the entirety of memory. Unfortunately this did not appear to be the case regarding implementation, as such processes were dumped on an individual basis.

## RESULTS AND DISCUSSION

### Analysis Output

#### *a. List of processes*

The Live Response can obtain detail information regarding all processes that related with the malware sample. Especially the **ps -t** command that can list whole related process including the processes that will diminish afterward. Figure 3 shows the detail processes that related with the malware. Examiner can see the similarity of all process based on its User ID (u0\_a53) and PPID (1585).

USER	PID	PPID	VSIZE	RSS	WCHAN	PC	NAME
u0_a53	1585	87	524300	40328	ffffffff	b6f71228	S com.android.system.admin
u0_a53	1592	1585	524300	40328	c002a59c	b6f71f88	S Signal Catcher
u0_a53	1593	1585	524300	40328	c030bec4	b6f70e5c	S JDWP
u0_a53	1594	1585	524300	40328	c0050f4c	b6f4ae70	S ReferenceQueueD
u0_a53	1595	1585	524300	40328	c0050f4c	b6f4ae70	S FinalizerDaemon
u0_a53	1596	1585	524300	40328	c0050f4c	b6f4ae70	S FinalizerWatchd
u0_a53	1597	1585	524300	40328	c0050f4c	b6f4ae70	S HeapTrimmerDaem
u0_a53	1598	1585	524300	40328	c0050f4c	b6f4ae70	S GCDaemon
u0_a53	1599	1585	524300	40328	c0266068	b6f71adc	S Binder_1
u0_a53	1600	1585	524300	40328	c0266068	b6f71adc	S Binder_2
u0_a53	1602	1585	524300	40328	c001ca7c	b6f71afc	S .ProcessManager
u0_a53	1604	1585	524300	40328	c0266068	b6f71adc	S Binder_3
u0_a53	1608	1585	524300	40328	c0050f4c	b6f4ae70	S Thread-208
u0_a53	1614	1585	524300	40328	c00bd870	b6f71d18	S Thread-212
u0_a53	1713	1585	524300	40328	c0050f4c	b6f4ae70	S Timer-1
u0_a53	1615	1585	3064	616	c0274b2c	b6f62f4c	S logcat

Figure 3: Live Response: ps -t command

LiME/Volatility (plugin: pid hash table) gives the similar output: name, PID and UID of the related processes as shown in Figure 4. Alongside with the detail of memory address of the process (offset and Directory Table Base/DTB). Not disclosed by Live Response is the start time. This information is significant for the examiner to know when the malware and its related process are starting and identified all the process that related with the malware based on the sequence of the start time.

Offset	Name	Pid	Uid	Gid	DTB	Start Time
0xcc6fc000	Signal Catcher	1592	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xc9046000	HeapTrimmerDaem	1597	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xcc741c00	.ProcessManager	1602	10053	10053	0x0c7d8000	2015-10-12 06:16:40
0xc90cb400	logcat	1615	10053	10053	0x09108000	2015-10-12 06:16:43
0xc9046c00	ReferenceQueueD	1594	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xc904b800	Binder_1	1599	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xcc741400	Binder_3	1604	10053	10053	0x0c7d8000	2015-10-12 06:16:40
0xc9046400	FinalizerWatchd	1596	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xc90cb800	Thread-212	1614	10053	10053	0x0c7d8000	2015-10-12 06:16:43
0xd3cec800	JDWP	1593	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xd30a7800	id.system.admin	1585	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xc904bc00	GCDaemon	1598	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xd3147800	Timer-1	1713	10053	10053	0x0c7d8000	2015-10-12 07:16:41
0xc9046800	FinalizerDaemon	1595	10053	10053	0x0c7d8000	2015-10-12 06:16:38
0xc90cbc00	Thread-208	1608	10053	10053	0x0c7d8000	2015-10-12 06:16:40
0xc904b400	Binder_2	1600	10053	10053	0x0c7d8000	2015-10-12 06:16:38

Figure 4: Volatility Plugin: PID Hash Table

On the Autopsy, the whole bin files from Mem tool's acquisition have been processed as a single case. After searching based on the keyword "system.admin," there are seven processes that might relate with the malware as shown in Table 2 below:

Table 2: Mem Tool: Using keyword for searching "system.admin"

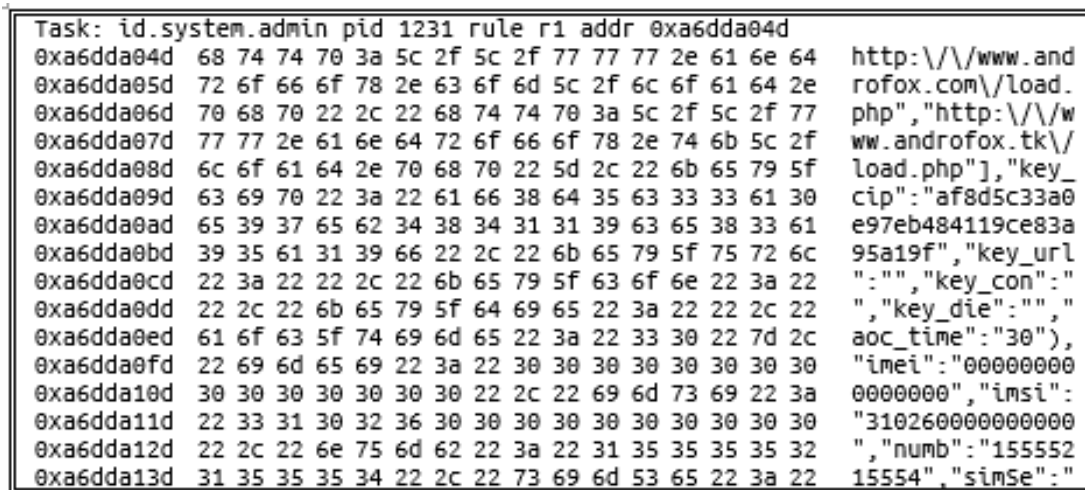
Keyword	Hits	PID	Process Name
system.admin	104	1541	com.android.system.admin
		1578	com.android.settings
		341	system.server
		664	com.android.input.method.latin
		69	zygote
		708	com.android.phone
		916	com.android.systemui

b. APK (Package) Detail

Live Response uses adb command: “dumpsys package ” and obtained the Android package file of the malware. It includes detail information such as the time stamp, first installed date, the last update time, etc. Moreover, the information such as the permissions that have been granted and the Android component such as intent and action are disclosed on the output file. The similar information could be gathered by examiner through static malware analysis.

c. C&C Server

This information is significant to reveal who is might responsible or involved with such incidents. Most of Trojans require C&C server to receive and to send data or files. However, the Live Response cannot reveal the information of C&C server. In contrast, LiME and Mem tool reveal how the process (com.android.system.admin) send the key cipher, the balance of prepaid SMS/MMS and other victim’s credential to the C&C server ([www.androfox.com](http://www.androfox.com) and [www.androfox.tk](http://www.androfox.tk)). Through Volatility’s plugin: yara\_scan -Y “http”, the examiner can obtain three findings based on the keyword search “androfox.” Figure 6 shows the example of the finding.



Task:	id.system.admin	pid	1231	rule	r1	addr	0xa6dda04d										
0xa6dda04d	68	74	74	70	3a	5c	2f	5c	2f	77	77	77	2e	61	6e	64	http://www.and
0xa6dda05d	72	6f	66	6f	78	2e	63	6f	6d	5c	2f	6c	6f	61	64	2e	rofox.com/load.
0xa6dda06d	70	68	70	22	2c	22	68	74	74	70	3a	5c	2f	5c	2f	77	php", "http://w
0xa6dda07d	77	77	2e	61	6e	64	72	6f	66	6f	78	2e	74	6b	5c	2f	ww.androfox.tk/\
0xa6dda08d	6c	6f	61	64	2e	70	68	70	22	5d	2c	22	6b	65	79	5f	load.php]", "key_
0xa6dda09d	63	69	70	22	3a	22	61	66	38	64	35	63	33	33	61	30	cip": "af8d5c33a0
0xa6dda0ad	65	39	37	65	62	34	38	34	31	31	39	63	65	38	33	61	e97eb484119ce83a
0xa6dda0bd	39	35	61	31	39	66	22	2c	22	6b	65	79	5f	75	72	6c	95a19f", "key_url
0xa6dda0cd	22	3a	22	22	2c	22	6b	65	79	5f	63	6f	6e	22	3a	22	": "", "key_con": "
0xa6dda0dd	22	2c	22	6b	65	79	5f	64	69	65	22	3a	22	22	2c	22	", "key_die": "", "
0xa6dda0ed	61	6f	63	5f	74	69	6d	65	22	3a	22	33	30	22	7d	2c	aoc_time": "30"),
0xa6dda0fd	22	69	6d	65	69	22	3a	22	30	30	30	30	30	30	30	30	"imei": "00000000
0xa6dda10d	30	30	30	30	30	30	30	22	2c	22	69	6d	73	69	22	3a	0000000", "imsi":
0xa6dda11d	22	33	31	30	32	36	30	30	30	30	30	30	30	30	30	30	"310260000000000
0xa6dda12d	22	2c	22	6e	75	6d	62	22	3a	22	31	35	35	35	35	32	", "numb": "15552
0xa6dda13d	31	35	35	35	34	22	2c	22	73	69	6d	53	65	22	3a	22	15554", "simSe": "

Figure 5: Volatility’s plugin: yara\_scan -Y “http”

The Mem tool provides the similar findings as LiME/Volatility. After the first launch, the malware sample collects information such as MAC address of the Bluetooth device, the name of the operator, telephone number, IMEI, phone users account balance, whether or not Device Administrator has been obtained, and local time. All this information is sent to the C&C server ([www.androfox.tk](http://www.androfox.tk) or [www.androfox.com](http://www.androfox.com)) in the form on an encrypted JSON object. After sending the information, the malware receives an instruction from the C&C server and records them in the database. Each instruction recorded in the database contains the instructions sequence number: the time when it must execute as ordered by C&C server and parameters (Unuchek, 2013). However, during the examination process, there is no instruction have been received by the malware.

d. Network Information

Live response with netstat commands only reveals the IP Address of the devices and Internet Service Provider without any information about third party IP Address that might relate with the incident.

In contrast, with plugin route cache in Volatility, the examiner found a particular IP Address: **195.20.46.245** that might relate with the incident. After checking those IP Address, apparently it has been already listed on the Blacklist Check.

e. Packets Sent through Sockets

The detail processes can be gathered from **lsolf** command on the Live Response. It shows how the malware attempts to connect its C&C server through the socket with file descriptor. However, from the output file it is shown that not all sockets are active. Hence, there may be no packets transmitted via those sockets at the time of



analysis (Unuchek, 2013). This feature shows the advancement of Live Response from LiMe/Volatility and Mem Tool.

LiME/Volatility with the plugin **lsof** reveals the similar output. Although there are two differences from Live Response’s output: First, there is no information whether the process is active or not (on/off); second, there is information about the size of the related process. However, it is not clear the cause of the differences: whether the plugin lsof did not parsing the information or the LiME did not dump the information.

With the Autopsy and based on the keywords search of “**androfox**” and “**system.admin**,” there are several hits in related processes that obtained from the bin files. Moreover, with regular expression search of “**URL**”, there is one hit in the related process (as shown in Table 3 below).

*Table 3: Mem Tool-Keyword/Regex Search*

<b>Keyword</b>	<b>Hits</b>	<b>PID</b>	<b>Process Name</b>
androfox	20	1541	com.android.system.admin
system.admin	104	1541	com.android.system.admin
		1578	com.android.settings
		341	system.server
		664	com.android.input.method.latin
		69	zygote
		708	com.android.phone
		916	com.android.systemui
<b>Regex: URL</b>			

*f. Time Events*

LiME/Volatility is the only tool that can reveal the time events of processes and services that related with the malware. Therefore, the examiner can reconstruct the incidents and easily identify the related processes or services based on the time events.

**CONCLUSION**

LiME and Volatility show the most robust findings for answering the investigative questions. Moreover, LiME has a repeatability feature that enable examiner or another examiner to do further analysis or re-analyse the memory dumps. However, LiME is not feasible for real world use for many reasons. Thus, the research suggests deploying LiME and Volatility Framework as a part of dynamic malware analysis only.

Live Response shows the less effective findings than LiME. Although providing less output overall, Live Response did provide some findings not found by LiME/Volatility. Significant advantages from Live Response are the implementation is far less complex and does not need root access. Mem Tool shows the least robust findings comparing with the two previous approaches. Mem Tool Allows for repeatability much like LiME. If root access is available, then acquisition can be performed with minimal impact on the state of the system

## REFERENCES

- 504ensicsLabs. (2014). LiME – Linux Memory Extractor Instructions v1.4. [https://github.com/504ensicsLabs/LiME/blob/master/doc/LiME\\_Documentation\\_1.4.pdf](https://github.com/504ensicsLabs/LiME/blob/master/doc/LiME_Documentation_1.4.pdf)
- (SWGDE), Scientific Working Group on Digital Evidence. (2013). SWGDE Best Practices for Mobile Phone Forensics
- Aljaedi, Amer, Lindskog, Dale, Zavorsky, Pavol, Ruhl, Ron, & Almari, Fares. (2011). Comparative analysis of volatile memory forensics: live response vs. memory imaging. Paper presented at the Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on.
- Ayers, Rick, Brothers, Sam, & Jansen, Wayne. (2014). Guidelines on Mobile Device Forensics NIST Special Publication 800-101 Revision 1. <file:///G:/ECU%20WORKS%20DOCUMENTS/DIT%20Meeting%20with%20Supervisor/NIST.SP.800-101r1%20Guidelines%20on%20Mobile%20Device%20Forensics.pdf>
- Burdach, Mariusz. (2006). Physical memory forensics. USA: Black Hat.
- Case, Andrew. (2012a). Analyzing Malware in Memory. <http://blog.hackeracademy.com/wp-content/uploads/2012/12/THA-Deep-Dive-Analyzing-Malware-in-Memory.pdf>
- Case, Andrew. (2012b). Android Forensics with Volatility and LiME: YouTube.
- Computer Security Incident Handling Guide (2012).
- Cisco. (2014). Cisco 2014 Annual Security Report.
- Group, The Internet Society Network Working. (2002). Guidelines for Evidence Collection and Archiving RFC 3227.
- Hoog, Andrew. (2011). Android Forensics: Investigation, Analysis and Mobile Security for Google Android: Syngress.
- Hubbard, Donovan. (2013). Best practices for Linux Live host analysis. from <http://forensics.donovanhubbard.com/2013/01/live-response-on-linux.html>
- Electronic Crime Scene Investigation: A Guide for First Responders, Second Edition (2008).
- Kercher, K. (2013). Best Practices: Linux Live Analysis. from <http://somethingk.com/main/?p=100>
- Leppert, Simon. (2012). Android memory dump analysis. Student Research Paper, Chair of Computer Science, 1.
- Ligh, Michael Hale, Case, Andrew, Levy, Jamie, & Walters, Aaron. (2014). The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory: John Wiley & Sons.
- Macht, Holger. (2013). Live Memory Forensics on Android with Volatility. (Diploma), Friedrich-Alexander Universitat, Nurnberg.
- Nagy, Tamer, Lindskog, Dr. Dale, & Zavorsky, Dr. Pavol. (2014). Analytic Comparison between Live Memory Analysis and Memory Image Analysis in Android Environment. Paper presented at the The 2nd World Congress on Computer Applications and Information Systems. <http://nngt.org/digital-library/upload/conference4/p19.pdf>
- Nasim, Faisal, Aslam, Baber, Ahmed, Waseem, & Naeem, Talha. (2015). Uncovering Self Code Modification in Android. In S. El Hajji, A. Nitaj, C. Carlet & E. M. Souidi (Eds.), Codes, Cryptology, and Information Security (Vol. 9084, pp. 297-313): Springer International Publishing.
- OpenSignal. (2015). Android Fragmentation Visualized (August 2015). from <http://opensignal.com/reports/2015/08/android-fragmentation/>
- Sims, Gary. (2013). Obad was the nastiest piece of Android malware discovered in 2013. from <http://www.androidauthority.com/obad-nastiest-piece-android-malware-discovered-2013-324830/>

- Sylve, Joe, Case, Andrew, Marziale, Lodovico, & Richard, Golden 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>
- Talley, Andre V. (2014). Content analysis tools in android memory forensics. (1554445 M.S.), Utica College, Ann Arbor. Retrieved from <http://ezproxy.ecu.edu.au/login?url=http://search.proquest.com/docview/1527126221?accountid=10675>  
<http://kx7gx4pm8t.search.serialssolutions.com/?&genre=article&sid=ProQ:&atitle=Content+analysis+tools+in+android+memory+forensics&title=Content+analysis+tools+in+android+memory+forensics&issn=&date=2014-01-01&volume=&issue=&spage=&author=Talley%2C+Andre+V>. ProQuest Dissertations & Theses Global database.
- Tamma, Rohit, & Tindall, Donnie. (2015). *Learning Android Forensics* J. Ursell, R. Youe & A. Varangaonkar (Eds.),
- Thing, Vrizlynn LL, Ng, Kian-Yong, & Chang, Ee-Chien. (2010). Live memory forensics of mobile phones. *digital investigation*, 7, S74-S82.
- Tinaztepe, Emre, Kurt, Doğan, & Gulec, Alp. (2013). *Android OBAD. Technical Analysis Paper*
- Comodo Malware Analysis Team. [https://www.comodo.com/resources/Android\\_OBAD\\_Tech\\_Reportv3.pdf](https://www.comodo.com/resources/Android_OBAD_Tech_Reportv3.pdf)
- Unuchek, Roman. (2013). The most sophisticated Android Trojan. from <https://securelist.com/blog/research/35929/the-most-sophisticated-android-trojan/>
- Valenzuela, Ismael. (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>
- Volatility. (2013). *Linux Memory Forensics*. Retrieved 4/12/2014, 2014, from <https://code.google.com/p/volatility/wiki/LinuxMemoryForensics>
- Wächter, Philipp. (2015). *Practical Infeasibility of Android Smartphone Live Forensics*. (Master), Friedrich-Alexander-Universität. Retrieved from [https://www1.cs.fau.de/filepool/gruhn/thesis\\_waechter.pdf](https://www1.cs.fau.de/filepool/gruhn/thesis_waechter.pdf)