Analysis of Uapush Malware Infection using Static and Behavior Method on Android

Syaifuddin*[1], Zamah Sari[2], Mohammad Khairul Masduqi[3]
[1,2,3] Universitas Muhammadiyah Malang
saifuddin@umm.ac.id*[1], abdzamahsari@umm.ac.id[2], khairulmasduqi@webmail.umm.ac.id[3]

Abstract

This research combines static and behavior analysis to detect malwares on Android system. The analysis process was completed by implementing analysis process on a malware-infected application running on an Android device. The analysis process was implemented based on specific stages, started from implementing behavior analysis on a malware-infected application running on Android device. Moreover, this behavior analysis ran the application on an Android emulator; afterwards, the file processing running on Android would be analyzed using the tool designed on this research to determine whether or not the executed application has been infected by malware. By utilizing behavior analysis, this research aimed to construct LiME kernel module being able to be executed on Android to collect data running on Android memory. This collected data would be analyzed further using volatility as data scanning. The second analysis utilized static analysis by checking the application on android system before running. During the static analysis, the application extraction was executed to generate some files to be analyzed to verify malware infection.

Keywords: *Malware, Android, Virtualization, Uapush*

1. Introduction

Malware, short from Malicious Software, is as hostile or intrusive software, intentionally developed to be embedded in a computer system to steal information data and even destroy a computer system [1].

An Android system is a Linux-based operation system having open access operation system. Android system has some superiority from having open source operation system, enabling multitasking, providing simple notification to great number of existing application and software running on this system. Therefore, these advantages provided on open source Android system, at the same time, can also bring some drawbacks by allowing open platform to all developers (users) to design and develop their own applications running on cellular devices. Hence, it eases software developers to develop malwares as applications to intrude Android system [2].

There are some malware analysis techniques, some of which are malware static analysis and malware behavior analysis. Malware static analysis is a malware analysis by checking applications suspected as malicious, without executing the file. On the other hand, malware behavior analysis is direct analysis by running and analyzing file system, memory, process, network traffic, and other modifications after executing the malware [3].

Memory is divided into two types, namely non-volatile and volatile memory. The analysis processed on non-volatile memory can be implemented to a memory retaining its data when the system is not running, exemplified by a hard disc, flash disc, and SD/micro SD/mini SD. However, the analysis process on volatile memory is implemented on memory such as RAM. Furthermore, on non-volatile memory analysis, there are some possible risks related to its integrity when implementing careless analysis processing. The analysis on volatile memory greatly reduce those risks because the researcher had made memory duplication containing all data in the system such as process, registry, files and system or specific applications in the system. Due to its function as a bridge between hard disc and processor, RAM, therefore, records all previous processes running in the system minimizing the potential risks. Even though the system itself is inactive, the analysis process is still able to run from duplicated RAM [4].

Android system operation, having big popularity in the world, has become the main target from malware developers. Android not designed to provide maximum security becomes the

Paper submitted on July 22, 2017; Revision on July 26, 2017; Received August 01, 2017

following reason. Initially, old version of Android showed significant security vulnerability; however, the following Android types exhibit adequate security level. This research employed new Android version, Lollipop, having better security level. A question needed to be answered is whether the types of attack and malware characteristics will exactly identical in different Android version. Thus, in responding to this question, the research performed analysis process on Uapush malware reported the biggest infection in 2016 for comparing between KitKat and Lollipop Android version.

Uapush.a is a malware included in adware Trojan Android having high level of threat, sending Short Message Service (SMS) messages, and stealing private information such as IMSI, IMEI, important information from a device, bookmarks and call history sent to C&C server in China. This type of malware is based on C&C located in China [5].

2. Research Method

Testing was performed to test the superiority of Lollipop and KitKat. Furthermore, this testing was conducted to answer the existing problems often found in different version of Android system operation related to malware infection; hence, compatibility testing was conducted, static and behavior analysis.

The testing itself had been designed to recognize the malware characteristics of Uapush on Android. The research ran using an Android emulator having similar functions to the original system. There were two methods combined to achieve more complete results. The first method employed was behavior analysis being an Android analysis technique running an application presumably infected by malware in the emulator; afterwards, the static analysis will be conducted by reading the application's source code presumably infected by malware [6].

Before installation, implementation and testing process were implemented, the system design and topology were developed. This system design would simplify the implementation and testing processes in analyzing Uapush-type malware. The following Figure 1 illustrates the system design.

2.1 General System Design

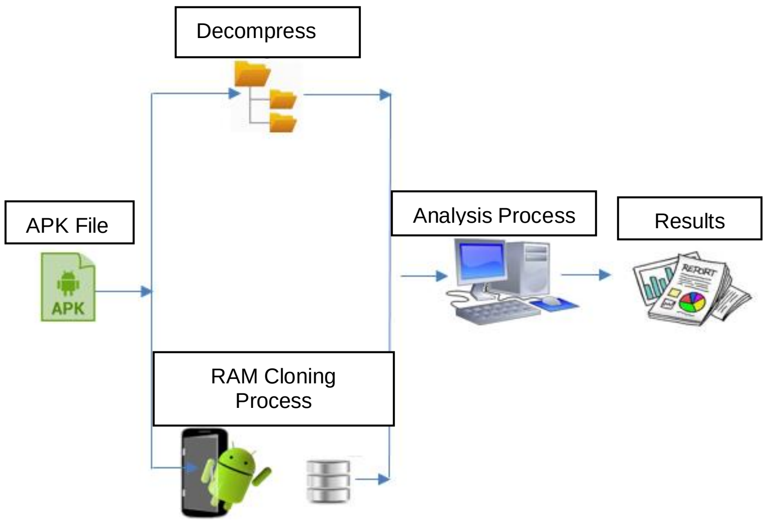


Figure 1. General System Design

Based on Figure 1, the research utilized victim smartphones, with rooted process and without rooted process. In order to simplify the architecture, it employed system operations in the form of virtual machine installed by those system operations. The input of this research is an Android application presumably infected with Uapush-type malware. Moreover, two analysis methods were employed using static and behavior analysis, commonly used in malware analysis processing. Static analysis performed in this research was conducted by decompressing the APK file and analyzing the result. Meanwhile, behavior analysis was started by running or executing the APK file into a device following by RAM cloning [7].

2.2 Architecture of Dynamic Analysis

Based on previously presented Figure 2, the researcher developed virtual machines in window host, each of them installed with rooted and non-rooted Android. Each virtual machine was infected by Uapush-type malware resulting into 2 virtual machines. Subsequently, the analysis process of live response on each virtual machine infected by malware was conducted [8].

Android_root

Figure 2. Architecture of Behavior Analysis

2.3 Architecture of Static Analysis

APK
malware_Uapush

Figure 3. Architecture of Dynamic Analysis

Figure 3 represents architecture design for analysis process in the windows host using Windows 8.1 Pro. Dex2jar and JD-GUI had been installed in Windows system used to analyze the application infected by Uapush-type of malware.

3. Research Result and Discussion

3.1 Dynamic Analysis

On this testing, a kernel module was created, and the results from compile LiME were stored into the devices. The following command was used to run kernel module [9].

```
$adb push lime-goldfish.ko /sdcard/lime.ko

$adb forward tcp:4444 tcp:4444

$adb shell insmod /sdcard/lime.ko “path=tcp:4444 format=lime” &
```

After running decompile LiME in the devices, the file would be accessed by suing Netcat to obtain capture results from the device RAM to be processed further in the analysis. The following command was used to access the file using Netcat:

```
$nc localhost 4444 > lime.lime
```

The previous command used was LiME TCP transfer to obtain the results of device RAM capture; consecutively, the scanning process was conducted for the results of RAM capture using volatile. Image RAM analysis was conducted to determine the characteristics from the calculator application infected by Uapush-type malware having characteristics of command and control server and investigated malicious commands to steal IMEI and IMSI. Furthermore, observing open file and running process executed by a malware infected application became two processes conducted in this research. The analysis results are presented in the following section.

3.1.1 Network Analysis

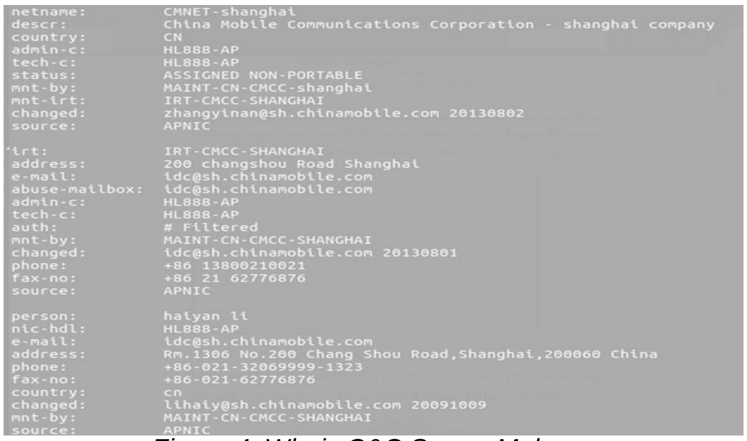
Analyzing process in network is necessary because several types of malware do communication through network. Such communication will be restored in RAM. Therefore, comprehensive study to analyze operating system and reverse engineering is required, volatile memory can be the substitute to analyze this case.

There were two plugins used, linux_route_cache plugin employed to observe network route running in the device and linux_arp plugin to observe default gateway. Table 1 presents the results of these examinations [10].

Table 1. Network Analysis of Uapush Image RAM in KitKat

Network Analysis	
Interface	Eth0
Destination	117.135.141.99
Gateway	10.0.2.2

From the results obtained from the route cache running in the device, it was observed that IP 117.135.141.99 conducted network route configuration in the device using gateway 10.0.2.2 allowing IP 117.135.141.99 to communicate with external parties. After it was checked using Whois, the results are as Figure 4.



```
netname:        CMNET-shanghai
descr:          China Mobile Communications Corporation - shanghai company
country:        CN
admin-c:        HL888-AP
tech-c:         HL888-AP
status:         ASSIGNED NON-PORTABLE
mnt-by:         MAINT-CN-CMCC-shanghai
mnt-irt:        IRT-CMCC-SHANGHAI
changed:        zhangyinan@sh.chinamobile.com 20130802
source:         APNIC

irt:            IRT-CMCC-SHANGHAI
address:        200 changshou Road Shanghai
e-mail:         idc@sh.chinamobile.com
abuse-mailbox:  idc@sh.chinamobile.com
admin-c:        HL888-AP
tech-c:         HL888-AP
auth:           # Filtered
mnt-by:         MAINT-CN-CMCC-SHANGHAI
changed:        idc@sh.chinamobile.com 20130801
phone:          +86 13800210021
fax-no:         +86 21 62776876
source:         APNIC

person:         haiyan li
nic-hdl:        HL888-AP
e-mail:         idc@sh.chinamobile.com
address:        Rm.1306 No.200 Chang Shou Road,Shanghai,200060 China
phone:          +86-021-32069999-1323
fax-no:         +86-021-62776876
country:        cn
changed:        lihaiy@sh.chinamobile.com 20091009
mnt-by:         MAINT-CN-CMCC-SHANGHAI
source:         APNIC
```

Figure 4. Whois C&C Server Malware

Who is informed C&C Server of Uapush-type malware was located on 200 Changhou Road, Shanghai, China.

3.1.2 IMEI and IMSI Analysis

Linux_yarascan plugin was used to identify and obtained the proof that the application had been infected by Uapush malware having previously installed in the device. The results are presented by Table 2.

From the previous results, the calculator application after being installed altered its name to kfines.RealCalc using a process running on PID 1323 having sub-process address of 0xb3f4b854 and 0xb3f4b803. It contained commands to acquire some device information such as IMEI and IMSI.

Table 2. Analysis of Data Stealing of Uapush Image RAM in KitKat

Analysis of Data Stealing	
Task	kfines.RealCalc
PID	1311
Rule	R1
Addr	0xb3f4b854, 0xb3f4b803

3.1.3 Open File Analysis

In order to identify the process to opening files in the infected application, the researcher utilized linux_lsof plugin. The results are presented by Table 3.

Table 3. Open File Analysis of Uapush Image RAM in KitKat

Open Files	
Process	29 open file processes
PID	1311
Address	0xd802f400

As informed by the results, the calculator application had completed 29 open file processes with PID process 1311 having address on 0xd802f400. Afterwards, the following analysis process on every open file process executed by the application can be implemented in order to identify each activity from every open file process conducted by the application infected malware.

Table 4. Analysis of Uapush Image RAM in Lollipop

File Description	
File Name	lime.lime
File Size	796,9 MB
Md5hash	ca7debc3782bfa5893c7c2ca1148a4ac

From the result showed in Table 4, RAM Uapush run on Android Lolipop with size of 796.9 MB and named as lime.lime was aimed to perform offline analysis, in order to prevent data alteration from Uapush-infected file through network. Therefore, Md5hash could help to prevent significant change from backed-up RAM image.

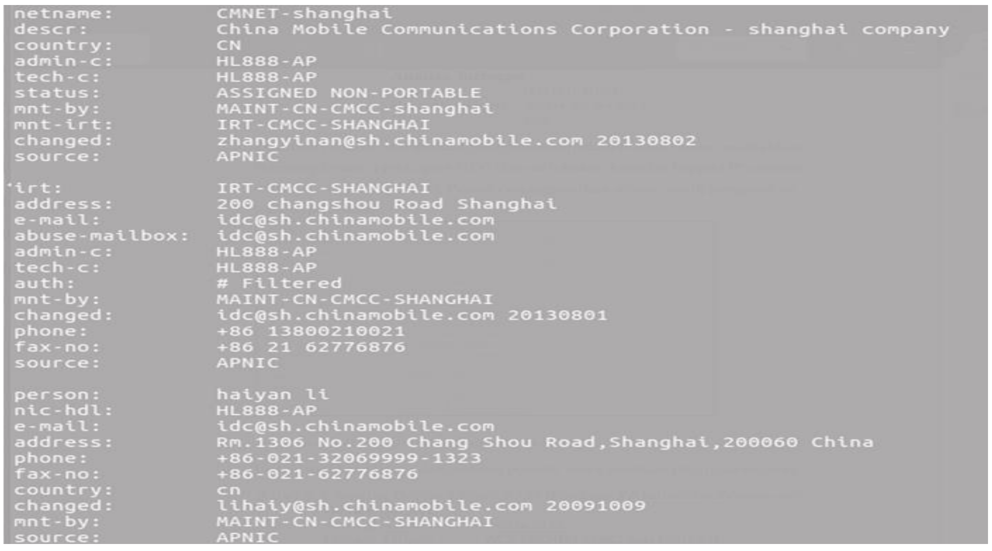
3.1.4 Network Analysis

Linux_route_cache plugin was used to examine the network gateway running on the device, and linux_arp plugin was used to identify its default gateway. The results are presented by Table 5.

Table 5. Network Analysis of Uapush Image RAM in Lollipop

Network Analysis	
Interface	Eth0
Destination	117.135.141.99
Gateway	10.0.2.2

From the results, it was identified from the existing route cache in the device that IP 117.135.141.99 conducted network route configuration in the device using gateway 10.0.2.2 as the gateway from device default to give access to IP 117.135.141.99 to communicate and to collect data from the infected device. The results after checked using Whois are as Figure 5.



```
netname:         CMNET-shanghai
descr:           China Mobile Communications Corporation - shanghai company
country:         CN
admin-c:         HL888-AP
tech-c:          HL888-AP
status:          ASSIGNED NON-PORTABLE
mnt-by:          MAINT-CN-CMCC-shanghai
mnt-irt:         IRT-CMCC-SHANGHAI
changed:         zhangyinan@sh.chinamobile.com 20130802
source:          APNIC

irt:             IRT-CMCC-SHANGHAI
address:         200 changshou Road Shanghai
e-mail:          idc@sh.chinamobile.com
abuse-mailbox:   idc@sh.chinamobile.com
admin-c:         HL888-AP
tech-c:          HL888-AP
auth:            # Filtered
mnt-by:          MAINT-CN-CMCC-SHANGHAI
changed:         idc@sh.chinamobile.com 20130801
phone:           +86 13800210021
fax-no:          +86 21 62776876
source:          APNIC

person:          haiyan li
nic-hdl:         HL888-AP
e-mail:          idc@sh.chinamobile.com
address:         Rm.1306 No.200 Chang Shou Road,Shanghai,200060 China
phone:           +86-021-32069999-1323
fax-no:          +86-021-62776876
country:         cn
changed:         lihaiy@sh.chinamobile.com 20091009
mnt-by:          MAINT-CN-CMCC-SHANGHAI
source:          APNIC
```

Figure 5. Whois C&C Server Malware

Who is informed that the C&C Server of Uapush-type malware was located on 200 Changhou Road, Shanghai, China.

3.1.5 IMEI and IMSI Analysis

Linux_yarascan plugin was used to identify and acquire proof that the application infected by Uapush malware installed in the device. The results are presented by the following Table 6.

Table 6. Analysis of Data Stealing of Uapush Image RAM in KitKat

Data Stealing Analysis	
Task	kfines.RealCalc
PID	1323
Rule	R1
Addr	0xb3f4b854, 0xb3f4b803

From the previous results, the calculator application after being installed altered its name to kfines.RealCalc using a process running on PID 1323 having sub-process address of 0xb3f4b854 and 0xb3f4b803. It contained commands to acquire some device information such as IMEI and IMSI.

3.1.6 Open File Analysis

In order to identify the process to opening files in the infected application, the researcher utilized linux_lsof plugin. The results are presented by Table 7.

Table 7. Open File Analysis of Uapush Image RAM in KitKat

Open Files	
Process	27 open file processes
PID	1323
Addrs	0xd802f400

As informed by the results, the calculator application had completed 27 open file processes with PID process 1323 having address on 0xd802f400. Afterwards, the following analysis process on every open file process executed by the application can be implemented in order to identify each activity from every open file process conducted by the application infected malware.

1. The Comparison of Uapush Characteristic between KitKat and Lollipop
   The following parameters show the comparison results on Uapush characteristics.
2. The comparison of process executed by Uapush.

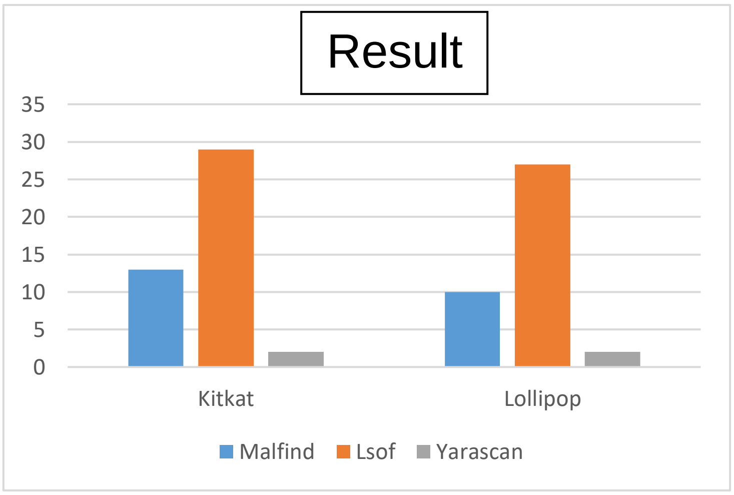


Figure 6. Comparison of Uapush Malicious Process

Figure 6 illustrates that Uapush in KitKat has more injected code and mutexes compared to Lollipop. This condition is caused by more fragile security system in KitKat and imperfect running of file configuration from Uapush kernel code. Having more mutexes in KitKat compared to Lollipop allowed unidentified running of malicious processes.

3.2 Testing Static Analysis

Static analysis becomes the second stage in malware analysis. It was started by choosing or finding application presumably infected with Uapush malware, the application was downloaded from http://www.down20.com/f-883953215. An android application detected to be infected by android malware would be analyzed using static analysis. The application infected by malware having APK extension renamed into ZIP extension. Afterwards, it was extracted by suing WinRAR because an APK file can be illustrated as an archive (a ZIP file) containing Dalvix Executable File (with DEX extension). The result of this extraction contains some files having files with DEX extension. Those DEX files, after that, were converted into JAR format using Dex2jar resulting a file with JAR extension.

The last step was decompiling JAR file using JD-GUI in order to be compatible with all Java source codes examined and analyzed to detect android malware infection. This malware is commonly embedded in the main application's class file or – in some cases – a specific root. Subsequently, the malware process in the android system would be analyzed. Finally, analyzing the resulting effects to the android system was conducted.

3.2.1 Analysis Process

On the previous process using dynamic analysis, the RealCalc.apk application would be scanned to detect malware. From the previous analysis results, the application was detected to be infected by malware with abnormality in some required permissions.

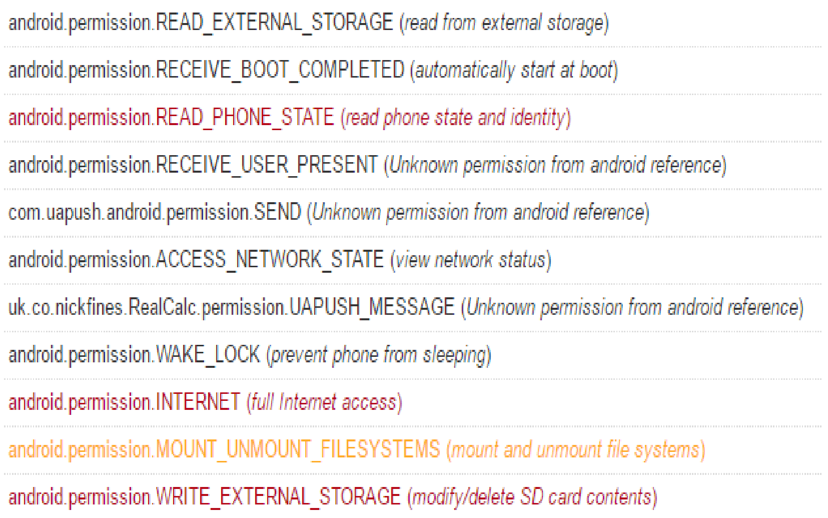

android.permission.READ_EXTERNAL_STORAGE (*read from external storage*)
android.permission.RECEIVE_BOOT_COMPLETED (*automatically start at boot*)
android.permission.READ_PHONE_STATE (*read phone state and identity*)
android.permission.RECEIVE_USER_PRESENT (*Unknown permission from android reference*)
com.uapush.android.permission.SEND (*Unknown permission from android reference*)
android.permission.ACCESS_NETWORK_STATE (*view network status*)
uk.co.nickfines.RealCalc.permission.UAPUSH_MESSAGE (*Unknown permission from android reference*)
android.permission.WAKE_LOCK (*prevent phone from sleeping*)
android.permission.INTERNET (*full Internet access*)
android.permission.MOUNT_UNMOUNT_FILESYSTEMS (*mount and unmount file systems*)
android.permission.WRITE_EXTERNAL_STORAGE (*modify/delete SD card contents*)

Figure 7. Detail Permission on File RealCalc.apk

The researcher performed analysis to RealCalc.apk application using some computer forensic tools. Renaming APK extension into ZIP extension was the first process conducted. RealCalc.zip file, as the result, was then extracted using WinRAR software generating some files having DEX extension.

Utilizing Dex2jar software, those DEX files were converted in the command prompt window generating a JAR file. The last process was decompiling JAR file using JD-GUI to examine the whole Java source code in the RealCalc.apk application. Figure 7 presents class file named Pushservice.class, actually unnecessary in RealCalc.apk application.

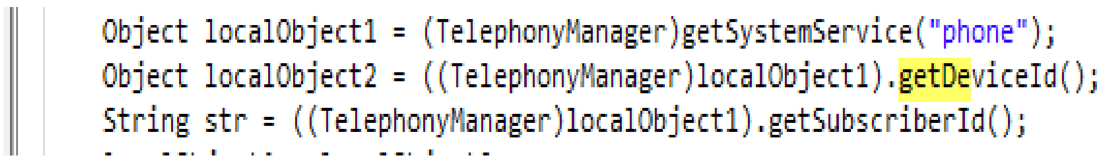
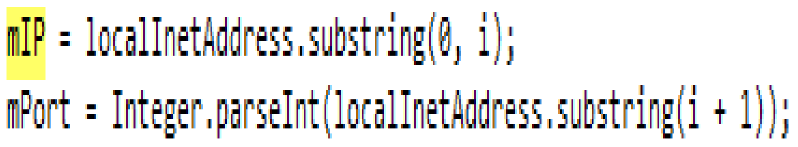

```
Object localObject1 = (TelephonyManager)getSystemService("phone");
Object localObject2 = ((TelephonyManager)localObject1).getDeviceId();
String str = ((TelephonyManager)localObject1).getSubscriberId();
```

Figure 8. Commands to Examine IMEI and IMSI

As presented in Figure 8, a source code containing "TelephonyManager" having function to gather IMEI (International Mobile Equipment Identity) and IMSI (Information Mobile Subscriber Identity) data from the Android infected by Uapush malware.

After reading IMEI and IMSI data, this malware collected these data. Both IMEI and IMSI are personal identity from every cellular device including an Android device. The information regarding IMEI and IMSI in an Android device was sent to a server having IP 117.135.141.99 with PORT 9000.



```
mIP = localInetAddress.substring(0, i);
mPort = Integer.parseInt(localInetAddress.substring(i + 1));
```

Figure 9. Script Used to Steal IMEI and IMSI Information from the Android Device

In the case of RealCalc application, the android malware started to be active after installed and run in the android device. The Uapush malware then read and collect IMEI and IMSI information from the infected device undetected by the user.

From those conducted analyses to the Java source code of RealCalc.apk application, this research summarized the malware processes infected RealCalc.apk application. This malware was active after RealCalc.apk was installed in the android device. Moreover, it ran malicious commands as well as new service in the device installed with RealCalc.apk.

4. Conclusion

The conducted implementation and testing results from the final paper entitled "*Analysis of Uapush Malware Infection using Static and Behavior Method on Android*" presents results as follows:

1. Malware analysis using image RAM capture results obtained from LiME in an Android should be implemented by developing kernel according to the device version used in the research resulting accurate data to illustrate the whole system activity.
2. Malware analysis using static method reveals more complete results to illustrate the workflow of an application before running.
3. By using malware characteristics, the analysis process is important to examine the overall malware activity in the system. All malware characteristics were identified in this research from behavior analysis.
4. The research shows the comparison discrepancy in the number of malware in the Android system between KitKat and Lollipop. The number of mutexes and malicious processes in KitKat is considered higher than those of the Lollipop presented by Graphic 1. This condition is caused by weaker security system in KitKat; hence, malwares can easily multiply the number of mutexes by camouflaging themselves in different processes.

References

[1] A. Kurniawan and Y. Prayudi, "*Live Forensics Technique on Zeus Malware Activities to Support Malware Forensic Investigation*," *HADFEX* (Hacking Digital Forensics Exposed), June 2014, Pp. 1–5, 2014.

[2] R. Novrianda, Y. N. Kunang, and P. H. Shaksono, "*Malware Forensics Analysis in Android Platform*," 2014.

[3] P. Richardus and E. Indrajit, "Malware Analysis."

[4] R. A. Pangestu, "*Analysis of Top 3 High Level Malware Infections on Zeroaccess, Alureon.dx, and Zeus using Digital Forensics based on Volatile Memory in Windows XP and Windows 7 Operation Systems*," University of Stuttgart, No. 9560291, Pp. 2–4, 2012.

[5] N. Threat, I. Report, N. Security, N. Threat, and I. Laboratories, "*Nokia Threat Intelligence Report*," 2016.

[6] Y.-H. C. Ming-yang su, Kek-Tung Fung, Yu-Hao Huang, and Ming-Zhi Kang, "*Detection of Android Malware: Combined with Static Analysis and Dynamic Analysis*," IEEE, Pp. 1013–1018, 2016.

[7] M. F. Agung, "*Basic Concept of Malware Analysis*," 2011.
[8] R. Adenansi and L. A. Novarina, "*Malware Dynamic*," Vol. 1, Pp. 37–43, 2017.
[9] F. Freiling, "*Practical Infeasibility of Android Smartphone Live Forensics*," Practical Infeasibility of Android Smartphone Live Forensics, 2015.
[10] Carbone, Richard. *"Malware Memory Analysis of the IVYL Linux Rootkit: Investigating a Publicly Available Linux Rootkit Using the Volatility Memory Analysis Framework,"* Defence Research and Development Canada-Valcartier Research Centre Quebec, Quebec Canada, 2015.