Study on Enhancement of Emulator to Incapacitate Analysis Evasion by Android Malicious Apps

Mijoo Kim, Woong Go, and Tae Jin Lee
Cyber Security R&D Team
Korea Internet & Security Agency (KISA)
Seoul, Korea (Republic of)
e-mail: {mijoo.kim, wgo, tjlee}@kisa.or.kr

Heung Youl Youm
Department of Information Security Engineering
Soonchunhyang University
Asan, Chungnam, Korea (Republic of)
e-mail: hyyoum@sch.ac.kr

***Abstract*—While smartphones are closely intertwined with our daily lives, with their influence expanding as their use has become more popular, security threats such as leak of personal information, illegal billing, and sending of spam using malicious apps that cause damage to smartphone users and give rise to social problems are also increasing. To solve such problems, security companies, research institutes, and academe worldwide are developing technologies to detect and cope with mobile malicious apps. Note, however, that malicious apps are also becoming more intelligent and elaborative to increase survivability by bypassing the existing detection means and countermeasures. As such, this paper describes the techniques of mobile malware to evade dynamic analysis and proposes measures to enhance emulators to incapacitate such analysis evasion by Android malicious apps.**

Keywords-evasive mobile malware; dynamic analysis; android malicious apps.

I. INTRODUCTION

Smartphones have evolved in close relation to our daily lives that we feel the global smartphone market has reached saturation. They have greatly changed our life pattern as smartphones help find the optimal route to a destination, check exercise, conduct financial transactions, and create new value-added services in combination with various Information Technology (IT) convergence technologies, such as Internet of Things (IoT).

Although people enjoy greater convenience in life with smartphones, they also experience the adverse effects of being exposed to security threats in various forms such as leak of sensitive information like personal information and account information, invasion of privacy by wiretapping text messages, infection by malware, inducement of billing such as small amount payment, control of terminal with illegally obtained privilege, and smishing. Moreover, the scope and level of damage from such security threats to smartphones are increasing, causing social problems. Android terminals are particularly prone to such smartphone security threats because of their openness and high market share. According to the smartphone Operating System (OS) market share analyzed by IDC [1] for the second quarter of 2015, Android had the largest market share with 82.8%; a joint report by Interpol and Kaspersky [2] disclosed in October 2014 indicated that 98.05% of mobile malware targeted the users of Android smartphones.

To minimize security threats to smartphone including Android, security companies, research institutes, and academe worldwide are developing countermeasure technologies; app markets have introduced analysis systems to detect malicious apps, and they are carrying out various programs to cope with mobile malware.

However, malicious apps are also becoming more intelligent and elaborative to increase survivability by having built-in self-protective technologies to bypass the existing detection systems and countermeasures in a same way as x86-based malware.

According to a report by LastLine [3], analysis-evading malware more than doubled from 35% in January 2014 to 80% in December, and such high figure has been maintained since then. Although there has been no report on the statistics of mobile malware, one can predict that it will evolve to a form similar to x86-based malware considering the typical evolution of malware.

The analysis evasion techniques of mobile malware target the dynamic analysis systems/services used by app markets and others for automated runtime analysis of a large volume of apps. They evade the analysis mostly using the environmental and time limitation of dynamic analysis. In February 2012, Google unveiled “Bouncer” as the malicious app analysis system for its Android Market and disclosed that the number of malicious apps decreased by 40% after Bouncer was introduced [4]. Note, however, that many technologies for detecting the Bouncer environment and evading analysis have emerged. Moreover, although various dynamic analysis tools and services were developed to detect and analyze Android-based mobile malicious apps, there are academic papers proving that they could be evaded through bypass technology as in the case of Bouncer.

In this paper, we propose measures to enhance emulators to incapacitate such analysis evasion by Android malicious apps. And the rest of this paper is organized as follows;

Section 2 describes the existing techniques of mobile malware to evade dynamic analysis. Section 3 specifies the proposed measures to enhance emulator to incapacitate analysis evasion and the result of experiment. And then we conclude in Section 4.

 ISBN: 978-1-61208-516-6

II. TECHNIQUES OF EVADING ANALYSIS OF MOBILE MALICIOUS APPS

The review of the trend in studies of evasion of dynamic analysis of mobile malicious apps shows that the cases can be mainly categorized into two types.

The first type is detecting the app running environment and not operating or executing the malicious behavior if it is not an actual terminal. A representative case of such type can be the bypassing of Google Bouncer announced by Jon O. and Charlie M. at SummerCon2012 [5]. The technique bypassed the verification system and enabled a malicious app to be registered in the Android Market by modifying the code when it receives the environment data under which the app runs during the verification stop when an app is registered. The "BrainTest" app, which actually got registered in Google Play in 2015 and infected more than 2 million devices, detected the analytical environment of Google Bouncer by checking the Internet Protocol (IP) address and domain character string and bypassed the analysis.

Techniques of evading analysis by detecting the virtual environment have been reported in many papers or presentations.

Timothy V. and Nicolas C. [6] showed that the analysis could be evaded after detecting the dynamic analysis system -- which was a virtual terminal -- by analyzing the difference of behavior between an actual terminal and a virtual terminal, performance, hardware and software components, and system design. Methods using the difference in behavior include checking the data that have the characteristics of emulator using Android Application Programming Interface (API), detecting the network emulations, and detecting the underlying emulator. The method using the performance difference detects the emulator by comparing the performances of Central Processing Unit (CPU) and graphic of actual terminals and emulator. The study also described the method of detecting the virtual environment according to the existence of hardware and software component.

Thanasis Petsas, et al. [7] deduced the static elements, dynamic elements, and hypervisor elements for detecting a dynamic analysis system. Static elements are the fixed values of a virtual terminal distinguishable from an actual terminal, and they include International Mobile Station Equipment Identity (IMEI), International Mobile Subscriber Identity (IMSI), and routing table. Dynamic elements are the values that dynamically change in an actual terminal but are fixed or are not provided in a virtual device; they include various sensors such as accelerator sensor, magnetic field sensor, rotation vector, proximity sensor, and gyroscope. Hypervisor elements use the configuration difference between a Virtual Machine (VM) emulation and the actual OS such as identifying the Quick Emulator (QEMU) scheduling and execution. The study tested 12 tools for the dynamic analysis of malicious apps including DroidBox [8], TaintDroid [9], Andrubis [10], CopperDroid [11], and Apk Analyzer [12] using 10 malicious apps that attempt to evade detection using the static, dynamic, and hypervisor elements and found that almost all dynamic analysis tools could not detect the evasion attempt except in the case of the attempt to evade detection using very simple static elements such as IMEI.

Yiming Zing, et al. [13] proposed Morpheus, a framework that automatically generates heuristics to detect an Android emulator by analyzing the difference between an actual terminal and a virtual terminal. Using Morpheus, they deduced more than 10,000 types of heuristics including basic heuristics such as network, power management, audio, USB, radio and software components, and configurations as well as the heuristics to detect QEMU, such as QEMU, Goldfish virtual hardware, Bluetooth, Near Field Communication (NFC), and vibrator and heuristics to detect VirtualBox and Personal Computer (PC) hardware. Their study showed that various evasion technologies can be applied against the dynamic analysis of malicious apps under the virtualization environment.

At HITCON2013, Tim Strazzere [14] announced various methods of evading emulators. He showed that an emulator can be detected with the checking of system attributes, checking of QEMU pipe to communicate with the host environment and checking of terminal contents such as address book, Short Message Service (SMS) transfer history, call list, and battery level.

The second type of technique of evading malicious app analysis is in logic bomb form by specifying the malicious behavior to be carried out only when the specific predefined conditions based on user interaction, time, and environment are satisfied.

The case of carrying out malicious behavior by detecting user interaction is similar to the technique of bypassing the analysis though a sandbox in the existing x86-based malware since it remains in hiding until it detects the intervention of human user such as mouse click and intelligent response to a dialog box. User interaction in the mobile environment occurs in the form of touch on a screen, touch on a popup, and information input. Although user interaction can be easily generated using a monkey that generates an event for a random coordinate value when simple interaction such as popup and button touch is required, there is a limitation as to what the monkey can do when an intelligent interaction such as continuous and accurate button touch or information input based on user judgment is required. An example is the "Horoscope" app, which attempted to leak the information by disguising as an app providing horoscope information. The Horoscope app [15] induced the user to touch a button twice continuously and accurately to obtain horoscope information in an attempt to leak the information stored in the smartphone.

Malicious behavior based on time condition is a case of carrying out malicious behavior not right after the app is run but after a specific period has passed or when a particular time is reached. A typical tool for the dynamic analysis of malicious app runs an app for a very short period since it cannot spend too much time analyzing an app. Therefore, it cannot detect a malicious app if the malicious app does not carry out malicious behavior during a short period. For example, Bouncer determines malicious behavior by observing an app for 5 minutes for dynamic analysis. It means that a malicious app designed to carry out malicious

 ISBN: 978-1-61208-516-6

behavior in 5 more minutes, being judged as a normal app since it does not show malicious behavior during the period of dynamic analysis. The “BrainTest” app used the time condition in addition to the detection of virtual environment so that it did not carry out malicious behavior during verification by Good Play but ran the malicious code at the command of the attacker after the app was downloaded.

The type based on environmental condition is a case of initiating malicious behavior when the predefined terminal environment conditions, such as network environment change (Long Term Evolution (LTE) <-> Wireless Fidelity (Wi-Fi)) or use of Global Positioning System (GPS) are satisfied.

The analysis can also be bypassed by initiating malicious behavior on various conditions such as combination of two or more normal apps, receipt of command by the attacker, receipt of text message containing a specific keyword, call from a specific number, and receipt of text message.

III. Enhancement of Emulator to Incapacitate Analysis Evasion by Android Malicious App

Most malicious app analysis tools and services that are currently available cannot detect evasive malware, and tools that claim to handle evasive malware use the actual terminals for the analysis. Note, however, that using the actual terminals has limitations in terms of analysis of a large volume of apps, restoration, and maintenance cost.

As such, this paper describes ways to enhance the emulator to incapacitate analysis evasion by malicious apps.

TABLE I. API List For Emulator Detection

	API	value
1	Build.ABI	armeabi
2	Build.ABI2	unknown
3	Build.BOARD	unknown
4	Build.BRAND	generic
5	Build.DEVICE	generic
6	Build.FINGERPRINT	generic
7	Build.HARDWARE	goldfish
8	Build.HOST	android-test
9	Build.ID	FRF91
10	Build.MANUFACTURER	unknown
11	Build.MODEL	sdk
12	Build.PRODUCT	sdk
13	Build.RADIO	unknown
14	Build.SERIAL	null
15	Build.TAGS	test-keys
16	Build.USER	android-build
17	TelephonyManager.getDeviceId()	All 0’s
18	TelephonyManager.getLine1 Number()	155552155xx
19	TelephonyManager.getNetworkCountryIso()	us
20	TelephonyManager.getNetworkType()	3
21	TelephonyManager.getNetworkOperator() .substring(0,3)	310
22	TelephonyManager.getNetworkOperator() .substring(3)	260
23	TelephonyManager.getPhoneType()	1
24	TelephonyManager.getSimCountryIso()	us
25	TelephonyManager.getSimSerial Number()	89014103211118510720
26	TelephonyManager.getSubscriberId()	310260000000000
27	TelephonyManager.gerVoiceMailNumber()	15552175049

The analysis of malicious apps using an emulator can overcome various limitations of using actual terminals. Note, however, that many recently announced malicious apps check the runtime environment of the app and do not carry out malicious behavior if it is an emulated environment. Considering the trend of x86-based malware, it can be predicted that more evasive malicious codes will appear in the mobile environment.

Therefore, enhancement of the emulator is needed so that the evasive malicious app cannot recognize the virtual environment. This study modified the framework of the emulator such that the data used in analysis evasion were the same as the actual terminal so that the malicious apps cannot recognize the emulator environment.

TABLE I shows the key APIs and values that can be used by malware for the recognition of emulator according to a study [6]. Each value means running environment is emulator or likely emulator or possibly emulator.

The Android framework of data corresponding to the 27 APIs was modified to change the emulator default values. As an example, Fig. 1 shows the changed source code of IMEI value called by build.DEVICE API. Fig. 2 shows the before and after the modification of IMEI value.

```
public String getDeviceId() {
    String deviceId = "";

    deviceId = "357242043237511";
    Taint.addTaintString(deviceId, Taint.TAINT_IMEI);
    Taint.log("[PhoneInfo]" + "[getDeviceId]" + "[IMEI=(" + deviceId + ")" + "]");

    return deviceId;
```

Fig. 1 Modification of IMEI value

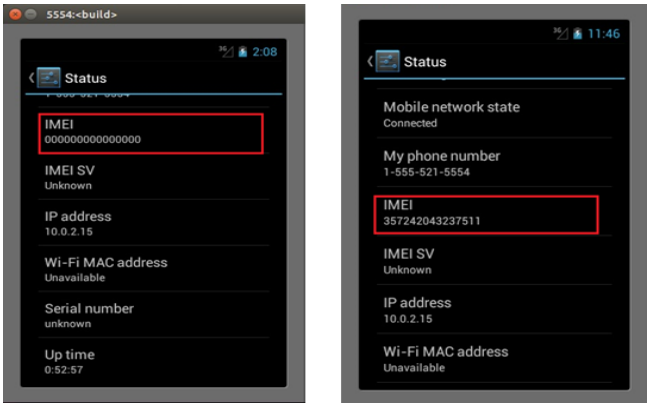

Fig. 2 IMEI data before and after emulator modification

And the result of experiment to verify effectiveness for emulator modification, the app developed to check the IMEI data for emulator environment -- and terminate the process in the case of emulator -- did not run normally in the case of default emulator but ran normally in the case of emulator with modified framework as shown in Fig. 3.

 ISBN: 978-1-61208-516-6

Fig. 3 The result of the experiment

IV. Conclusion

This study reviewed the issues of dynamic analysis evasion that incapacitates malicious app detection and analysis by bypassing the existing dynamic analysis system as a means of self-protection by mobile malicious apps, which are becoming more intelligent and elaborative. The review shows that current dynamic analysis technologies have limitations.

Dynamic analysis evasion technologies include the type that does not show malicious behavior by detecting the virtual environment such as emulator and the type that evades analysis by initiating malicious behavior only when specific conditions such as user interaction, time, and environment are met.

As such, this study proposed the enhancement of the emulator to incapacitate the analysis-evading behavior of malicious apps in an Android malicious app dynamic analysis system and showed that such analysis evasion can be incapacitated by modifying the Android framework without using the actual terminal.

Nonetheless, additional studies are needed to enable the analysts to change the data dynamically since the modified data are hardcoded and can be evaded. Moreover, it is necessary to conduct studies to return the actual terminal value of sensors, batteries, and levels in addition to the terminal attribute-specific data corresponding to 27 APIs listed in this study, and we plan to continue studies to solve such issues.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R0132-16-1004, Development of Profiling-based Techniques for Detecting and Preventing Mobile Billing Fraud Attacks).

References

[1] IDC, http://www.idc.com/prodserv/smartphone-market-share.jsp, retrieved: October 2016.

[2] Kaspersky, “Mobile cyber threats”,Kaspersky Lab&Interpol Joint Report, 2014.

[3] Lastline, “Labs Report at RSA: Evasive Malware’s Gone Mainstream”, 2015

[4] Google Mobile Blog, “Android and Security”, 2012.

[5] J. Oberheide, C. Miller, “Dissection the Android Bouncer”, SummerCon, 2012.

[6] T. Vidas and N. Christin, “Evading Android Runtime Analysis via Sandbox Detection”, ASIA CCS’14, pp. 447-458, 2014.

[7] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis and S. Ioannidis, “Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware”, EuroSec’14, Article No. 5, 2014.

[8] DroidBox, https://github.com/pjlantz/droidbox, retrieved: October 2016.

[9] TaintDroid, http://appanalysis.org/, retrieved: October 2016.

[10] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Veen and C. Platzer, “Andrubis – 1,000,000 Apps Later: A View on Current Android Malware Behaviors”, BADGERS’14, pp. 3-17, 2014.

[11] CopperDroid, http://copperdroid.isg.rhul.ac.uk/copperdroid/, retrieved: October 2016.

[12] Apk Analyzer, https://www.apk-analyzer.net/, retrieved: October 2016.

[13] Y. Jing, Z. Zhao, G. Ahn and H. Hu, “Morpheus: Automatically Generating Heuristics to Detect Android Emulators”, ACSAC’14, pp. 216-225, 2014.

[14] T. Strazzere, “Dex Education 201 Anti-Emulation”, HITCON2013, 2013.

[15] C. Zhengm, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han and W. Zou, "SmartDroid: an Automatic System for Revealing UI-based Trigger Conditions in Android Applications", SPSM'12, pp. 93-104, 2012.

 ISBN: 978-1-61208-516-6