Scientific Journal of Impact Factor (SJIF): 4.72
e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 5, Issue 01, January -2018

Implementation and Validation on Improvements for Dynamic Analysis System to Detect Evasive Mobile Malware

Mijoo Kim[1], Woong Go[2], Jun Hyung Park[3], Heung Youl Youm[4]

[1]Security Technology R&D Team 1, Korea Internet & Security Agency
[2]Security Technology R&D Team 1, Korea Internet & Security Agency
[3]Security Technology R&D Team 1, Korea Internet & Security Agency
[4]Department of Information Security Engineering, Soonchunhyang University

Abstract —*As the use of mobile devices such as smart phones and tablets has increased, and with a heavier reliance on them in everyday life on the rise, the malicious codes targeting mobile devices have been steadily increasing. A worldwide effort has been directed towards developing technologies to detect and cope with mobile malicious apps. However, malicious apps are also becoming more intelligent and sophisticated to increase survivability by bypassing the existing analysis means, obfuscating code, detecting the analysis environment (a sandbox, emulator, etc), which is used to analyze malicious apps, and suspending malicious behavior accordingly. In this paper, we examine the analysis-evasive techniques used in malicious mobile apps, and the methods to deal with them. We also implement and verify the improvement plans for a dynamic analysis system to incapacitate the analysis-evasive behaviors of malicious apps.*

Keywords*-Evasive Mobile Malware; Evasive Malicious Application; Mobile Malware Detection; Mobile Malware Dynamic Analysis*

I. INTRODUCTION

As the use of mobile devices such as smart phones and tablets has increased, and with a heavier reliance on them in everyday life on the rise, the malicious codes targeting mobile devices have been steadily increasing. According to McAfee[1], by the third quarter of 2017, the cumulative number of detected malicious mobile codes exceeded 20 million. Most of the malicious codes designed for mobile devices targets Android devices with the openness and high market share. According to IDC's smartphone OS market share in Q2 of 2015[2], Android has the largest share with 82.8%, and according to a report of INTERPOL and Kaspersky Lab in October 2014[3], 98.5% of malicious mobile apps are targeting Android smartphone users.

In counteracting to malicious code, a worldwide effort has been directed towards advancing the detection and analysis systems, developing the static and dynamic analysis tools, and sharing the malicious code analysis methods. However, malicious apps are also becoming more intelligent and sophisticated to increase survivability by having built-in self-protective technologies (concealment, detection evasion, etc) to bypass the existing detection systems and countermeasures. According to a report by LastLine[4], the number of malicious codes with analysis-evading function has continuously increased, therefore, the percentage of malicious codes with the function of analysis evasion has increased from 35% in January 2014 to 70% in December, and such high ratio has been maintained. Malicious mobile code is also expected to follow the evolution of x86-based malicious code. A variety of evasive mobile malwares have been found, including 'BrainTest', a variant malicious app that has been distributed through Google Play and infected over 2 million devices in 2015. An evasive mobile malware can bypass the analysis of an existing analysis system and cause serious harm through propagation and spread of malicious apps; it is necessary to develop a technology that can cope with this. Although static analysis allows rapid analysis and detection of massive malicious apps, it can be easily bypassed by obfuscation. In this study we focus on analysis-evasive behaviors that bypass a dynamic analysis system performing an analysis based on the behaviors of a malicious app.

In this paper, we propose and implement the improvement plans for a dynamic analysis system which incapacitate malicious behaviors of a malicious app. We also validate the improvement plans against an malicious app that performs analysis-evasive behaviors. This paper is organized as follows:

In Section 2, we describe the related works on the various analysis-evasive behaviors used in malicious apps. In Section 3, we discuss the countermeasures for evasive mobile malwares. In Section 4, we design and implement a dynamic analysis system in order to counteract against evasive mobile malware. In Section 5, we validate a dynamic analysis system against an actual malicious app that conducts analysis-evasive behaviors. And finally, we conclude this paper in Section 6.

II. RELATED WORKS

The dynamic analysis-evasive techniques of malicious mobile apps include a method of detecting an emulator, which is mainly used for dynamic analysis, and a method of using logic bombs, which operate based on the conditions predefined by an attacker.

3.1. Evasion through emulator detection

The first type is detecting the emulator by analyzing the difference between an actual terminal and the emulator, and not operating or executing the malicious behavior if it is the emulator. 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. This type of analysis-evasive behavior occurs because the application dynamic analysis system runs mainly in a virtual environment such as an emulator, taking advantage of the difference between an actual terminal and a virtual environment. In fact, the "Brain Test" app, which was 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 various 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 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 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.[8] 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, NFC, and vibrator and heuristics to detect VirtualBox and 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[9] 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, SMS transfer history, call list, and battery level.

3.2. Evasion by Conditional Behavior (Logic Bombs)

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 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[10].

The type of malicious behavior based on time conditions is when malicious actions are taken place only after a certain period of time without reaching the malicious action immediately after the execution of the app. 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 within a short period of time. For example, Google's 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 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 (LTE<->Wi-Fi) or use of 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. COUNTERMEASURES

In this section, we describe countermeasures to detect evasive mobile malwares. The detection of an malicious app is often performed by detecting the emulator, which is used for dynamic analysis, and not operating or executing the malicious behavior. A malicious app detects an emulator by analyzing the difference between the actual terminal and the emulator and using the inherent characteristics of the underlying emulator. Therefore, the easiest way is to use an actual terminal instead of an emulator for dynamic analysis. The advantage of using actual terminals for dynamic analysis is to incapacitate almost all attacks based on the difference between the actual terminal and the emulator (e.g. checking system attributes and identifying QEMU execution). The disadvantage is that there may be a cost burden, including securing a large number of terminals in an environment used for a large volume of automated analysis, replacing the terminals according to their lifetimes, and securing the terminals according to their versions. In addition, since an actual user does not use an device on the move, dynamic sensing data cannot be provided although the sensors are supported. In order to lessen the burden of the terminal use cost, mounting only the minimum functions of a terminal which are necessary for analysis onto a board also provide similar characteristics.

Another method is to analyze the difference between the terminal and the emulator and modify the emulator to have the characteristics of the terminal. The advantage of this method that it is possible to incapacitate the detection behavior of emulator by modifying the Android framework so that an emulator can have the characteristics of a terminal, and by randomly generating dynamic data for the sensors. In addition, there is little hardware cost issue because it is easy to expand the number and version of an analysis tool without using a large volume of costly automation environment. However, the following disadvantages exist: The Android framework needs to be modified. There are more issues of development than those of an terminal, and they are being addressed based on the announced major techniques. The emulator is not 100% identical with the terminal, so it is difficult to cope with analysis evasion that uses an unknown method or a method that a framework cannot be modified.

Furthermore, a terminal can be used together with an emulator, which is modified to have the characteristics of a terminal. First, the analysis is performed through the emulator, and additional analysis is performed on a malicious app that is suspicious of analysis evasion through the terminal, thereby using the pros and cons of the terminal and the emulator as complementary roles. However, there is a need for further consideration in terms of the criteria for selecting applications to be delivered to the terminal and the consistency of the terminal and emulator analysis results.

IV. DESIGN AND IMPLEMETATION

In this paper, we propose a method among the countermeasures against evasive mobile malware described in Section 3 in order to improve the existing dynamic analysis system by modifying the emulator as an actual terminal. An emulator and an actual terminal have differences in their CPU information in build/telephony information, terminal boards, terminal brands, product names, manufacturers, hardware serial numbers, IMEI, IMSI, phone numbers, Mobile Country Code (MCC), Mobile Network Code (MNC), voice mail number information, etc. Also, since an emulator does not support sensors unlike an actual terminal, it cannot return data when an application requests the sensing information.

With the aim of enabling the analysis on the analysis-evasive behaviors by allowing the actual terminal information and the virtual sensing data values to be returned to a malicious app that tries to detect the emulator and bypass the analysis based on the terminal's attributes and sensor information, we propose improvements in this study for a dynamic analysis system.

In order to incapacitate a malicious app's analysis-evasive behavior, we developed an anti-evasion framework service and an app that run in the background so that information of an actual terminal can be transmitted as follows. The composition of the anti-evasion service logic is shown in Figure 1.

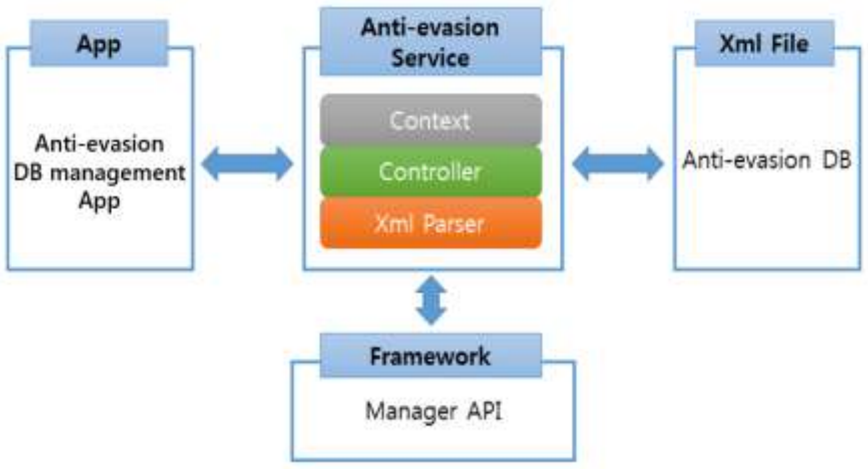

Figure 1. Components for Anti-evasion service

The anti-evasion service with the app can get/set the data values stored in the DB from/to Android system by interacting with APIs and attributes in each manager of the framework. The components of the service include the controllers that respond to analysis evasion by interworking with managers and the XmlParser for reading or storing values from/to DB in Xml file format.

- AntiSystemController: Build information-related data management
- AntiTelephonyController: Phone information-related data management
- AntiContactController: Generate contact data
- AntiCallLogController: Generate call log data
- AntiPowerController: Change battery level and status
- AntiCellController: Change radio signal strength at random

In addition, the followings are anti-evasion framework service and the application processing procedure.

(1) Initial data setup process
When the emulator is running, the Android system runs the respective services through the INIT process. In this case, the anti-evasion service is registered in the service manager beforehand so that the anti-evasion service can be started in the initialization process. The anti-evasion service that is linked with the API of each framework-level reads the data from the DB, where analysis-evasive bypass data (virtual terminal data) is managed, and replaces it through the API.

(2) Data reset process
If the initially set data value needs to be changed, the data value is changed through the anti-evasion application at the application level, and the corresponding value is stored in the anti-evasion service DB. When the resetting process is performed, the initial data setting process of the above (1) is repeated through rebooting for the actual terminal attributes DB to be applied.

(3) API modification and return process
In case of calling the API corresponding to the manager at the framework level, the related terminal information needs to be changed. In this case, it is necessary to modify the internal API code so that the information can be replaced with the API of each controller in the anti-evasion service.

In order to perform malicious app analysis and verification, we implemented a tool into the emulator having built-in analysis evasion incapacitation in order to analyze the malicious behaviors of API call and data log extraction against the network, file system, H/W resource access, processes, call, SMS, phone information and service information.

V. VERIFICATION

In order to verify the technique proposed in this paper, we installed an actual malicious app with the analysis evasion function on an emulator with the proposed technique and on the other without the proposed technique, and compared the resulting behaviors respectively. The two malicious apps used for verification were the “SMSSecurity" variant app announced in December 2016 and the zero-day mobile ransomware, "Charger" announced in January 2017.

First, Figure 2 shows a screenshot of the execution results of each analysis emulators for the "SMSSecurity" variant app.

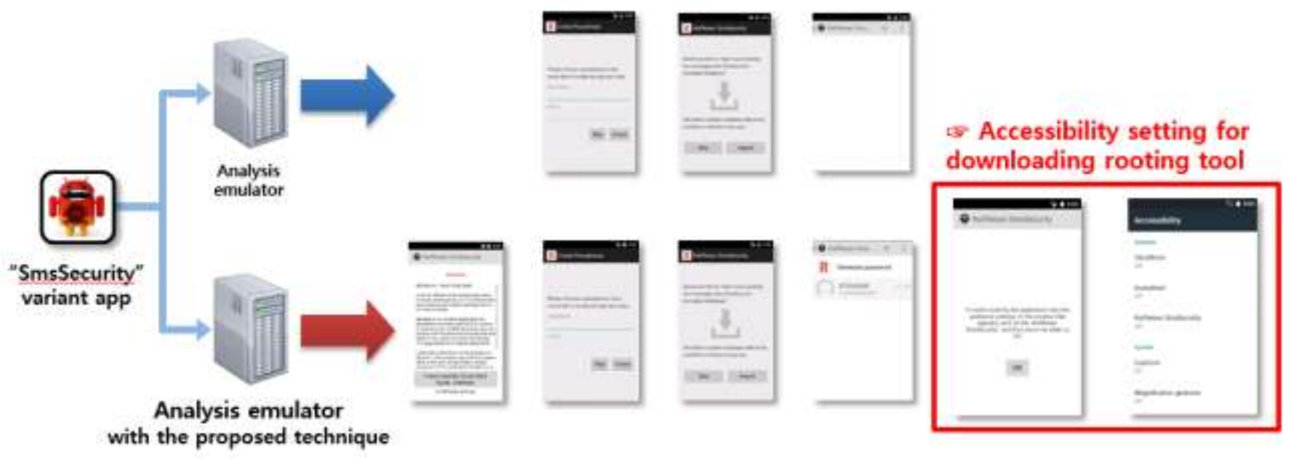
"SmsSecurity" variant app
Analysis emulator
Analysis emulator with the proposed technique
☞ Accessibility setting for downloading rooting tool

Figure 2. Comparison of behaviors

As shown in Figure 2, no malicious behavior was found in the analysis emulator that did not have the analysis evasion incapacitation, and in the analysis emulator equipped with the analysis evasion incapacitation, additional malicious access activity to download the rooting tool was found.

And as the result of the execution results of each analysis emulator for the "Charger" ransomware, no malicious behavior was found in the analysis emulator that did not have the analysis evasion incapacitation, and in the analysis emulator equipped with the analysis evasion incapacitation, an attempt to access information (e.g. SMS and contact information) and an attempt to leak network data were found in the logs.

VI. CONCLUSION

In this paper, we address the issues of the analysis evasion, a function of malicious and increasingly sophisticated mobile code, and confirmed the limitation of the dynamic analysis technology on malicious app at present. The dynamic analysis-evasive techniques used in malicious apps include a method of detecting an emulator which is mainly used for analysis and a method of performing a malicious behavior only when the conditions predefined by an attacker meet.

In this paper, we proposed and implemented a method to improve the dynamic analysis system by modifying the emulator's Android framework as a countermeasure against the analysis evasion through emulator detection. We verified that the proposed method can be used to incapacitate the evasive behaviors of an malicious app and reveal the hidden malicious behaviors of the malicious app by performing the validation against the actual malicious app that performs analysis evasion actions.

In the future, we hope that the results of this study will be applied to the malicious app analysis and detection systems and incapacitate the evasion behaviors of malicious apps, thereby increasing the malicious app detection rate and contributing to the development of malicious app analysis tools.

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2015-0-00003, Development of Profiling-based Techniques for Detecting and Preventing Mobile Billing Fraud Attacks)

REFERENCES

[1] McAfee, "McAfee Labs Threat Report", December, 2017.
[2] IDC, http://www.idc.com/prodserv/smartphone-market-share.jsp, retrieved: October 2016.
[3] Kaspersky, "Mobile cyber threats",Kaspersky Lab&Interpol Joint Report, 2014.
[4] Lastline, "Labs Report at RSA: Evasive Malware's Gone Mainstream", 2015.
[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] Y. Jing, Z. Zhao, G. Ahn and H. Hu, "Morpheus: Automatically Generating Heuristics to Detect Android Emulators", ACSAC'14, pp. 216-225, 2014.
[9] T. Strazzere, "Dex Education 201 Anti-Emulation", HITCON2013, 2013.
[10] 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.