2011 Seventh International Conference on Computational Intelligence and Security

Kernel-based Behavior Analysis for Android Malware Detection

Takamasa Isohara, Keisuke Takemori and Ayumu Kubota
KDDI R&D Laboratories
Saitama, Japan
{ta-isohara, takemori, kubota}@kddilabs.jp

***Abstract*—The most major threat of Android users is malware infection via Android application markets. In case of the Android Market, as security inspections are not applied for many users have uploaded applications. Therefore, malwares, e.g., Geimini and DroidDream will attempt to leak personal information, getting root privilege, and abuse functions of the smartphone. An audit framework called logcat is implemented on the Dalvik virtual machine to monitor the application behavior. However, only the limited events are dumped, because an application developers use the logcat for debugging. The behavior monitoring framework that can audit all activities of applications is important for security inspections on the market places. In this paper, we propose a kernel-base behavior analysis for android malware inspection. The system consists of a log collector in the Linux layer and a log analysis application. The log collector records all system calls and filters events with the target application. The log analyzer matches activities with signatures described by regular expressions to detect a malicious activity. Here, signatures of information leakage are automatically generated using the smartphone IDs, e.g., phone number, SIM serial number, and Gmail accounts. We implement a prototype system and evaluate 230 applications in total. The result shows that our system can effectively detect malicious behaviors of the unknown applications.**

Keywords-component; smartphone security; malware; Android;

I. INTRODUCTION

Android is an open source software stack for smartphones led by Google [1]. One of the smartphone's key characteristics is that a device can run a wide variety of mobile application software on an application runtime environment. However, when a malware is installed on the smartphone, the smartphone user faces a serious threat such as information leakage, getting root privilege and abuse functions of the smartphone as with the situation of malware infection on PC. Today, a lot of android malware such as Geinimi [2] and DroidDream [3] is released on the application marketplace because every android application can be published into the market without third-party's review. One of the solutions that improve safeness of marketplace is to find out an application that acts a malicious behavior by preliminary review and avoid the use of them.

Android platform has system debug monitor called logcat. This mechanism generates log data that contains activity of system and applications. However, the logcat is unsuited for behavior analysis to detect a malware for the following reasons. First of all, only the limited events are dumped into the log data because the logcat is designed for application debugger. Hence, primitive activity information about a file I/O events or process management events is not recorded. Secondly, a log generation of applications activities depends on a programmer of application. Since malware avoid log generation to hide their activity in general, we cannot get sufficient data to analyze an activity of an interested application. Therefore, an audit mechanism that can collect all activities of applications without relying on implementation of application is required to achieve a reliable malware detection system.

In this paper, we propose a kernel-based behavior analysis system for Android malware detection. The system consists of log collector on an android device and log analyzer on a PC. The log collector records application activity on kernel layer because all application cannot avoid this mechanism. However, since kernel-level log will be generated a large amount of data with application execution, the size of log data should be reduces for efficient analysis. Additionally, since log data will contain all activity of applications that runs on Android system, there are many noises for accurate malware detection. Thus, a log collector records events of valuable system call to detect a malicious activity. Additionally, log analyzer picks up a process tree of interested application's activity. A malicious activity is detected by signature matching approach. Signatures are described by regular expression to accurately catch rows that contain an evidence of a malicious activity. Additionally, some data stored on the device is automatically collected and converted into a signature for detecting personal information leakage. We have implemented prototype system and analyze total 230 application collected from Android Market or voluntary web site. The system detects 37 applications that leak some kind of personal information, 14 applications that execute exploit code and 13 destructive applications.

The rest of this paper is organized as follows: Section 2 describes background information on the Android platform, Section 3 design and implementation of our proposal. Section 4 presents results from our Android application study, Section 5 describes related work, and Section 6 summarizes our conclusions.

II. BACKGROUND INFORMATION

This section begins by describing an overview of Android platform and security frameworks. We then discussing classifications of malicious application based on threats of smartphone.

A. *Android Platform*

This section describes the basics of Android necessary for understanding the remainder of this paper.

1) *System Architecture*

Android is a software stack designed for a smartphone. As depicted in Figure 1, Android provides a sandboxed application execution environment. A customized embedded Linux system interacts with the phone hardware. The middleware and application API runs on top of Linux. An application's only interface to the phone is through these APIs. Each application is written in Java and is executed within a Dalvik Virtual Machine running under a unique UNIX uid [4].

2) *Security Enforcement*

To protect a resource and data on the device from an attack, android platform provides a permission-based security enforcement mechanism [5]. Access to a system resources and data is governed by permissions assigned at installation time. The permissions requested by the application and the permissions required to access the application's resources/data are defined in its manifest file. Permission assignment—management of the security policy for the phone—is largely delegated to the phone's owner: the user is presented a screen listing the permissions an application requests at installation time, which they are asked to accept or reject as depicted in Figure 2.

However, this mechanism has a shortcoming to surely protect a user from a threat. Since permission mechanism cannot enforce fine grained access control to a resource or data on a device, it is difficult that the smartphone user figure out a threat that might be caused by executing an application. Additionally, when a root privilege is obtained on the device, this permission mechanism is easily bypassed.

3) *System debugging tool*

Android provides system debugging tool called logcat. This tool monitors an activity of system and applications on Dalvik VM layer and these data is generated as a log data. To output a log data of application, a developer should implement a function that generates log data in desired point of the source code.

B. *Smartphone Threats*

First of all, we focus a potential risk for abuse of private data stored on a smartphone. Since a smartphone is a mobile device that evolved from a traditional cellular phone, it contains various sensitive personal data and a device is always connected to a mobile network. Hence, smart phone users always face a risk of information leakage. An example of sensitive personal data as follows:

- Telephone number/E-mail address: Smartphone subscriber assigned these identifiers by mobile network operator. Additionally, Gmail account
- Contact list: It contains telephone numbers and/or e-mail addresses of an interested person of the device owner.

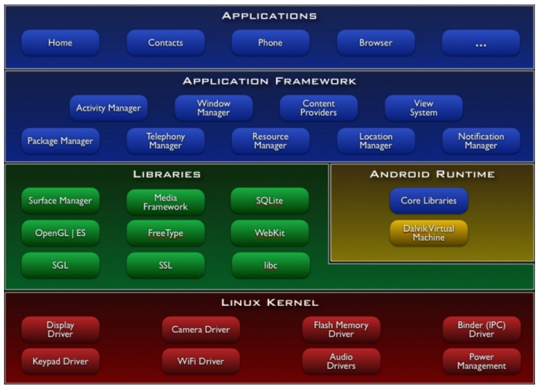
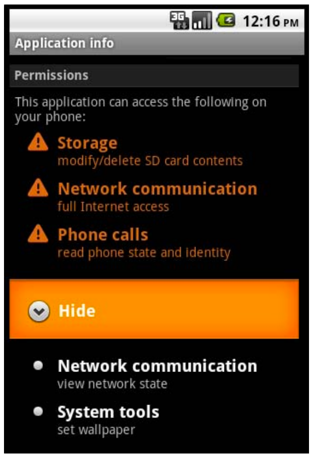


Figure 1. Architecture of Android Platform



12:16 PM
Application info
Permissions
This application can access the following on your phone:
Storage
modify/delete SD card contents
Network communication
full Internet access
Phone calls
read phone state and identity
Hide
Network communication
view network state
System tools
set wallpaper

Figure 2. Confirmation of permission

- IMSI(International Mobile Subscriber Identifier) / ICCID(IC Card ID): Identifier of SIM card.
- IMEI(International Mobile Equipment Identifier): Identifier of mobile device.

Although IMSI/ICCID/IMEI does not immediately expose an identity of a smart phone owner, when a malicious service provider collects these data, a service subscriber faces risk of violation of his/her privacy.

The next most common threat of a smartphone security is cracking of security enforcement caused by a root privilege escalation and abuse of the root privilege. Since the Android platform is build on the top of customized embedded Linux system, a root privilege of the device can be obtained by attacking to a vulnerability of Linux zone. A method to gain the root access is called "jailbreak" and many jailbreak techniques are provided on web site [4]. When the root privilege is obtained on the smartphone, security framework loses its effectiveness. Hence, Jailbreak is considered as a threat on smart phone security because security architecture loses effectiveness under a rooted environment.

In addition, there is application software that only performs when a device has root environment. This application software has risk for leaking personal information or performing an activity that contrary to security policy defined based permission. Hence an activity that needs root privilege should be also considered as a threat on smart phone.

III. Kernel-based Behavior Analysis System

This section described design and implementation of kernel-based behavior analysis system. At first, architecture of proposed system is described. Then, we present how to collect and analyze kernel-level log data.

A. System architecture

To reveal a threat hidden in application software, one of the effective approaches is dynamic analysis of application activity. Therefore, we propose a kernel-based behavior analysis system to expose malicious activities of application software. Figure 4 depicts architecture of the proposed system. Components of the system are an Android smartphone and a server machine. The android smartphone is used to collect a log data of application process. A logging module is implemented as a kernel-level system monitor to surely collect an activity of application process. Therefore, even if targeted application process attempts to avoid inspection of its own activity, system surely collects activity of process. And then collected log data will be sent to a server machine for analysis.

B. Collection of log data

Log collection is the first step of malicious activity analysis. To collect a log data that contains a lot of useful contents with few noise contents is important subject of proposed system. This section describes what kind of log data is collected and an approach for noise contents reduction.

1) Kernel-based logging module

We classify Log data produced by an application execution into two categories on the basis of its source of data; one is application-level log and the other is kernel-level log. An application-level log is generated by applications themselves and a kernel-level log is generated by operating system. Though both of them can be used to figure out application's behavior, there are some differences in its characteristics. First is its understandability. It is easy to figure out what happened by an application execution from reading of an application-level log. On the other hand, it is difficult to obtain the same level of comprehensiveness from kernel-level log analysis. The second is a certainty of data collection. An application-level log is generated by application process itself, while a kernel-level log is generated by operating system. Therefore, an application-level log has low certainty because malicious application can easily avoid to output log data by themself.

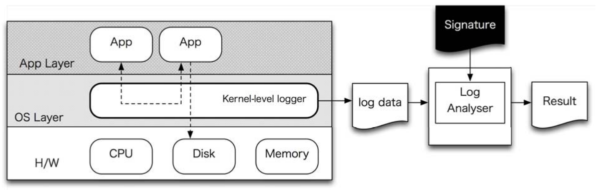


Figure 3. System architecture

Based on the above discussion, we focus a kernel-level log as a material to reveal malicious activities of application software because it is difficult to cheat logging mechanism. Additionally, since there are some research activities to securely logging data in the area of Linux operating system, robustness of a proposed evaluation system will be improved when secure logging system is implemented into Android platform.

2) Selection of system calls

One of the problems for kernel-level log analysis is amount of log data. As a kernel-level logging module collects all activities that occur on operating system, a large number of log data are collected. These collected data includes a lot of noise contents irrelevant to a malicious activity. Additionally, since capacity of storage on a smart phone is limited, it is important to reduce amount of log data. To resolve these problems, we collect only logs that are related with interested system calls for malware detection.

Linux operating system has about 300 system calls and these system calls can be classified into some categories depending on function of operating system such as process management, memory management and device management. To reveal malicious activities of application process, we can obtain helpful information from process management activity and file I/O activity. Process management is activity that creates process by executing a binary code or kills process of application. File I/O is activity that creates, deletes, reads and writes file objects on a system. This category contains network access because socket is handled as same as file descriptor. From these activities, we can find out what kinds of processes are active on a system. Table 1 lists system calls that we have selected.

C. Analysis of log data

Log analysis is the second step of malicious activity analysis. In this step, it is required to reduce error rate both a false negative and a false positive. This section describes how a system analyzes a log data.

1) Extraction of targeted process tree

In log collection phase, we have selected system calls recorded to log data in order to reduce noise. By doing so, we can obtain log data that contains record of interested activities. However, this log data also contains record of non-interested activities because log data have collected of all processes that run on operating system. Therefore, we take an additional approach that removes log data of uninterested process to reduce noise from collected log data. Here, interested processes are process created by execution of targeted application binary code and its child processes.

TABLE I. List of selected system calls

Category	Name of system call
Process management	clone, execve, fork, getuid, getuid32, geteuid, geteuid32
File I/O	accept, bind, connect, mkdir, open, read, recv, rename, rmdir, send, stat, unlink, vfork, write

To fulfill the requirement, a method for picks up a process tree of interested process is implemented in this step. Detailed descriptions of proposed procedure are as follows.

To pick up a process tree for log analysis, the first step is identification of a process created by execution of targeted application code. We can achieve this by using an Android logging system that provides a mechanism for collecting system debug output. This mechanism called "Logcat" and generated log messages can view by *logcat* command via a debug support tool for android named ADB (Android Debug Bridge). From logcat messages, we can find out a process identifier (PID) of interested application process. Though this step uses application level log message, this message is reliable because it is generated by android system. Since a parent process generate a child process by execution of fork()/clone() system call, tracing an interested process tree can be achieved by detecting lines that include "fork" or "clone" pattern from kernel-level log and analyze a detected line to identify a PID of child process.

2) Signatures for malicious activity detection

Since proposed system detects malicious activities by using signatures, low error rate of a false negative and a false positive is achieved by carefully described signatures. In this section, we clarify a cause of both errors and describe details of signature generation for accurate malware detection.

When rule set of system is insufficient, it will be a cause for high false negative. On the other hand, if a rule set has a lot of unrelated words, a false positive rate will be increased. Therefore, we carefully select an adequate number of keywords based on a type of threat that described in Section 2. To detect a personal information leakage, we select personal identifiers and device identifier. This category includes words such as a telephone number, Gmail account, Android ID, IMEI, IMSI, SIM serial number. Since this information differs from device to device, the logging module automatically gets information from the device and will send it to the server. In addition, we have investigated that some mobile advertising services collect privacy sensitive data. Although we do not judge this case immediately to be a threat, we select the name of a mobile advertising service provider to whether the kernel-level logging module can detect or not as a trial. In the case of detecting jailbreak application, we set a name of known exploit binary file as a signature. Lastly, since an android application is restricted to execute a program required root privilege, a destructive application can be detected by using a command name that requires root privilege as a signature.

Properly selected keywords are useful to detect a malicious activity without false negative error. However, we should also consider a false positive error. For example, when a Mailer application makes a user directory by using user's account name such as "foo@example.com", a system wrongly detect this activity as an information leakage.

Based on the above understanding, we take an approach that uses an assistance keyword and regular expression to generate a signature. An assistance keyword is name of system call or path name of file. To generate a signature combined with selected keyword and assistance keyword, regular expressions is effectively works. Additionally, the number of signatures will be shrunk as a side effect of using a regular expression. By using this technique, the system achieves detection of an interested threat without context analysis.

IV. APPLICATION ANALYSIS RESULTS

We have implemented prototype system on Android smart phone. First, we compare amounts of log data to evaluate an efficiency of proposed noise reduction method. Then, we enumerate signatures used to threats detection. Finally, we report on an application study that uses a prototype system to analyze activities of 230 popular third-party Android applications.

A. Experiment environment

We have implemented a strace command into a HTC Magic running Android 2.1 based modified ROM image. The collected data is transferred to notebook PC via adb shell. And then, the log data is sent to a server machine to detect a threat.

B. Efficiency of Log Size Reduction

We have proposed two noise reduction methods: one is selecting system call at tracing activities of processes on the system, and the other is picking up a process tree of interested process. To evaluate an efficiency of these methods, we investigate number of output lines of kernel-level log. We selected 5 applications from Android market and execute them on the prototype system. Table 2 enumerates the number of lines of log data on each application. From table 2, we can see a selecting system call dramatically reduces number of output lines in every case. Additionally, picking up a process tree method also reduces number of lines especially in the case of "Friend App" and "Facebook". Therefore, these two methods contribute to reduce amount of log data.

C. Signatures

To clarify the behavior of an application has done on the system, we have generated 16 patterns of signatures in this evaluation. Signatures can be classified into three categories based on threats that application might have. Table 3 enumerates signatures for each threat categories.

D. Case Study Analysis

We have collected total 230 applications as a sample of analysis. The greater part of sample applications were downloaded from Android Market and some applications that are not published in Android Market was downloaded from voluntary sites.

TABLE II. NUMBER OF LINES OF LOG DATA

Application	Raw data	System call filtered	Process Tree filtered
Friend App	71081	25474	25129
fring	44297	1367	801
Facebook	98825	6620	574
Norton Mobile Security	117903	42399	41387
Wireless tether	26967	2747	932

TABLE III. SIGNATURES FOR MALWARE DETECTION

No.	Threat	Signature
1	Info. Leakage	^recv\\(.*(%IMSI%|%SIM%).*$
2	Info. Leakage	^recv\\(.*(%IMEI%|%ANDROID%).*$
3	Info. Leakage	^recv\\(.*(%TEL%|%MAIL%).*$
4	Info. Leakage	QuattroWirelessSDK
5	Info. Leakage	(ADMOB|AdMob|admob)
6	Info. Leakage	AdWhirl
7	Info. Leakage	FlurryAgent
8	Jailbreak	^(execve|read)\\(.*/asroot
9	Jailbreak	^(execve|read)\\(.*/exploid
10	Jailbreak	^(execve|read)\\(.*/rageagainstthecage
11	Jailbreak	^(execve|read)\\(.*/gingerbreak
12	Abuse of root	^execve\\("/(system/)?(bin|sbin|xbin)/su",
13	Abuse of root	^open\\(".*/iptables",
14	Abuse of root	superuser(\\....)?
15	Abuse of root	^execve\\(".*/busybox",
16	Abuse of root	^execve\\(".*/(chmod|chown|ls\\b|mkdir|rmdir)",

1) Information leaking application detection

We have found total 37 applications that leak some kind of personal sensitive data. In details, 30 applications leak IMEI, Android ID. 3 applications leaks IMSI, SIM serial number. 4 applications leaks telephone number or e-mail address.

We have found 80 applications that match with signature that uses a name of mobile advertise service providers. In these 80 applications contains 37 applications as described previously.

2) Jailbreaking application detection

We have found total 14 applications detected execution of exploit code. In details, 1 application executes *asroot*, 5 applications execute *rageagainstthecage*, 7 applications execute *exploid* exploit code and 1 application executes *gingerbreak* exploit code.

All applications that execute *exploid* or *rageagainstthecage* are Trojan program infected with "Droid Dream". Since these applications silently get a root privilege on a target device, applications has a serious threat and it should be detected as a malicious application.

3) Destructive application detection

We have found that 13 applications execute "su" command in their activity. Applications that match with "superuser" pattern were found 8 applications.

V. RELATED WORK

Enck et al. [7] describe the design and implementation of a framework to detect potentially malicious applications based on permissions requested by Android applications. The framework reads the declared permissions of an application at install time and compares it against a set of rules deemed to represent dangerous behavior. For example, an application that requests access to reading phone state, record audio from the microphone, and access to the Internet could send recorded phone conversations to a remote location. The framework enables applications that don't declare dangerous permission combinations to be installed automatically, and defers the authorization to install applications that do to the user. Our proposal focuses an activity of application based on assigned permissions. Hence, the system detects threats caused by execution of a target application while [7] evaluate a risk hiding in a target application.

OS-level protections such as TaindDroid [8] and Saint [9] provide enhanced security mechanisms for Android. These mechanisms address an activity of a target application. In [6], personal information on the phone is labeled and traces a flow of marked data by modifying android API. Although these systems detect an information leakage, other types of threats cannot detect. Our system can detect a wide variety threats by generating signature for interested event.

VI. CONCLUSIONS

We have proposed a kernel-based behavior analysis for Android malware inspection. The system collects system call events generated at application runtime and analyzes it. The system achieves collecting log data that only contains data of target activities. Log data is analyzed by signature-based pattern matching. We define three categories of threats and generate signatures correspond to each category of threat. We have implemented prototype system and analyze total 230 application collected from a marketplace or voluntary web sites. The system detects 37 applications that leak some kind of personal sensitive data, 14 applications that execute exploit code and 13 destructive applications. In other words, kernel-based behavior analysis can be applied for security inspections for Android application markets.

REFERENCES

[1] Android. https://www. android.com/.

[2] Android.Geinimi, http://www.symantec.com/security_response/writeup.jsp?docid=2011-010111-5403-99

[3] DroidDream malware, http://blog.mylookout.com/2011/03/security-alert-malware-found-in-official-android-market-droiddream/

[4] DalvikVM.com. http://www.dalvikvm.com/.

[5] Security and Permissions. http://developer.android.com/guide/topics/security/security.html.

[6] iPhone Jailbreaking vs. Android Rooting. http://android.appstorm.net/general/iphone-jailbreaking-vs-android-rooting/.

[7] ENCK, W., ONGTANG, M., AND MCDANIEL, P. On Lightweight Mobile Phone Application Certification. In Proceed- ings of the 16th ACM Conference on Computer and Communica- tions Security (CCS) (November 2009).

[8] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Moni- toring on Smartphones. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (2010).

[9] ONGTANG, M., MCLAUGHLIN, S., ENCK, W., AND MC-DANIEL, P. Semantically Rich Application-Centric Security in Android. In Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC) (2009).