공학석사학위논문

Execution Offloading 기술을 사용한 모바일 기기를 위한 클라우드 보안 솔루션

Cloud Security Solution of Mobile Applications based on Execution Offloading

2015년 8월

Seoul National University
School of Electrical and Computer Engineering
Ali Almokhtar

Execution Offloading 기술을 사용한 모바일 기기를 위한 클라우드 보안 솔루션

Cloud Security Solution of Mobile Applications based on Execution Offloading

백윤흥 교수님

이 논문을 공학석사 학위논문으로 제출함
2015년 8월

Seoul National University
School of Electrical and Computer Engineering
Ali Almokhtar

Ali Almokhtar의 석사 학위논문을 인준함
2015년 8월

위 원 장 윤성로 (인)
부위원장 백윤흥 (인)
위 원 정교민 (인)

Abstract

So far, security mechanisms for mobile devices have had difficulties to protect from malicious threats due to the limited resources of mobile devices. With the prevalence of cloud computing, one of promising solutions to overcome the difficulties is to exploit cloud environments, where a remote virtual machine performs the resource-consuming security analysis instead of a mobile device. However, existing cloud-based solutions are still insufficient because of the code coverage problem and security level degradation. Therefore, this thesis proposes a static and dynamic analysis based security solution called SORCloud. For dynamic analysis, it offloads a process of a suspicious application to a remote virtual machine for dynamic security analysis, by which SORCloud resolves two problems mentioned above. Through comprehensive experiments, we show how efficiently the proposed scheme works and detects malicious behavior.

Keywords : Offloading, Cloud Systems, Static and Dynamic Analysis, Malwares analysis tools

Student Number : 2013-23848

Contents

List of figures

List of Tables

Chapter 1

Introduction

The number of malwares targeting mobile devices such as smartphones or tablets is growing fast. Mobile devices usually have several types of critical information: user's position, certificates including personal information which is used for the financial transactions, private contacts, and a gallery containing pictures and videos, and so on. This nature of the mobile devices tempts malicious attackers to steal the valuable information through malware attacks, which makes it necessary to protect the mobile devices against the information leakage.

Of course, the malware attacks are not new threats. The malwares on mobile devices are not quite different from those of PCs (personal computer) such as desktops and laptops. In order to protect from malware attacks, there have been proposed a lot of solutions to detect the malwares. However, the legacy solutions are not suitable for mobile devices because of the limited capacity and computing resources of mobile devices.

One of alternative solutions to overcome the limitation of mobile devices is to detect malwares by using separate servers. The basic concept is that separate powerful servers take on the detection which requires a heavy workload on behalf of mobile devices. Recently, with the prevalent use of the cloud computing, Virtual Machines (VM) are widely used as the separate servers. In this thesis, therefore, every separate server is assumed to oper-

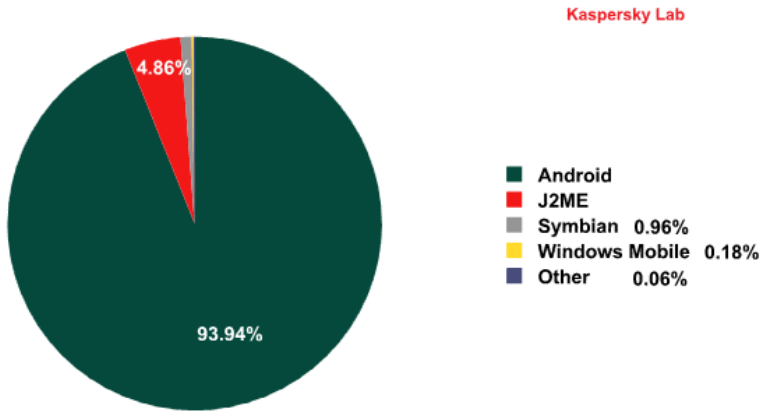

Figure 1: Distribution of mobile threats by platform , 2004-2012.[23]

ate as the VM in the cloud. This kind of solutions can be broadly classified into two different approaches, sandbox-based and replay-based. As Figure .1 shows that mobile threats targeting android platforms have massively increased due the huge the number of usage of Android platform, almost 94% of the attacks attack android based on applications

Firstly, the sandbox-based approach[18, 26, 29, 31, 20, 32, 22] literally uses a VM as a sandbox[1]. In this approach, the required security modules, e.g., a static malware detector or a dynamic behavior analyzer, are installed into the VM, and a suspicious program is executed and analyzed through the installed security modules in the VM acting as a sandbox. Thus, this approach can avoid the overload of mobile devices for detection. Furthermore, the information can always be protected even if the suspicious program fulfills its task since the VM is not a real mobile device but just a sandbox. However, it cannot be guaranteed that the behaviors of the suspicious pro-

[1] Sandbox is a security mechanism for separating untrusted or suspicious programs.

gram are examined thoroughly, which is called a Code Coverage Problem, since the inputs to the program, e.g., typing numbers or pressing a volume button, are not generated from a real user, but an emulator.

Secondly, in the replay-based approach[34, 24], all the events that occur in the mobile device are replayed in the VM. Similar to the sandbox, the required security modules are installed into a VM, and they examine the behaviors of the suspicious program. The main difference is that the inputs to the application program are sent from the real user's device in real-time, and what the suspicious program does in the mobile device are replayed in the VM. In other words, the VM executes the suspicious program one more time with the same inputs as the mobile device. Therefore, it does not have the code coverage problem due to the use of the actual user's inputs. However, it needs the initial overhead to make the same environment as the mobile device in the VM, and the communication overhead to transmit user's input to the VM. The fatal shortcoming is that it cannot prevent the information leakage since it is a post processing method. That means even if malicious behaviors are detected in the VM, the information has been already stolen from the mobile device.

To sum up, the sandbox-based approaches offer the secure analysis environment which can prevent the information leakage, but have the code coverage problem. On the other hand, the replay-based approaches provide the complete examination, but cannot guarantee the information leakage prevention, which causes the degradation of security level. So far, we have had to abandon one of code coverage and security level because of the trade-off between two different approaches.

We propose a new approach to overcome the tradeoff, which is an offloading-based security solution for mobile devices called SORcloud (Security ORiented cloud). SORcloud also installs the required security service modules on a VM, executes a suspicious program in the VM, and makes it analyzed through the security modules. It is noteworthy that execution offloading[2] is introduced for dynamic analysis for the behaviors of mobile devices.

The rest is organized as follows.In Section 2, we discuss related work. In Section 3, we explain SORcloud framework design and implementation. Section 4 evaluate SORcloud. we conclude in Section 5 and finally in Section 6, we discuss future work.

[2]The execution offloading is a technique that gets some parts of a program code run in a remote cloud in order to avoid mobile device's overload.

Chapter 2

Background

Android is open platform for mobile platform, embedded and wearable devices, it's based on Linux kernel made by google with user interface based on direct manipulation. The android application frame work has different structure, it doesn't have main function and return call, though the developers should design the applications in terms of components, Android applications are executed by Dalvik . Dalvik is open-source software and it's a process virtual machine made specifically for android and considered as integrated part of android software stack. .apk files are the application packages for android systems and consists of 3 compressed files 1- class byte code 2-resources 3- Binary native files. After these components are compressed, it is signed by a key created by SDK.

Android applications are written in Java, and compiled into bytecode for the java virtual machine which is translated in to dalvik bytecode and stored in .dex (Dalvik Executable) and .odex (optimized dalvik executable) files. The main advantage of using dalvik that it is optimized for low memory requirements as it uses less space and the constant pool has been modified to use 32-bit indices to simplify the process of the interpreter and according to google developers, dalvik allows the device to run multiple instances of the VM efficiently.

offloading is considered one of the cloud computing techiques that is

being developed to connect mobile devices with limited resources to powerful cloud servers to perform complicated operations in much less time for efficiency and robustness , The act of transferring a process between two machines in a network (the source and the destination node) during its execution,I offloading process, the code is segmented into parts and might be offloaded statically or dynamically depending on the complexity of the code and process state transfer of every part in the code

Chapter 3

Related Work

An android application sandbox system called **aasandbox** [18] is one of the early sandbox based approaches. An application is executed in a fully isolated environment, where low-level interactions like system calls are logged for monitoring and analysis. The other sandbox based approach is **Appspalyground** [26], which consists of detection components and exploration mechanisms to analyze smart phone applications. The automatic exploration mechanism is used to allow more parts of the application to be executed, which can increase the code coverage. **Mobile-sandbox** [29] is a sand-box based hybrid system combining static and dynamic analysis. It detects the malicious behaviors of an application by logging calls to native (non-Java) APIs. **Andrubis** [31] is also a hybrid system designed to analyze unknown applications. It performs more efficient dynamic analysis by using the results of the static analysis. **Taintdroid**[32, 20] is a dynamic taint tracking system which has the ability to track multiple sources of sensitive data. It provides the real time analysis by leveraging Android's virtualized execution environment. **Droidbox** [22] is based on the Taintdroid approach. It provides an effective way for dynamic analysis, and generates the reports for information leakage via network, file and SMS. Furthermore, the analysis process could show the cryptography operations which is being done in the execution using Android API[17]. **Secloud**[34] is one of replay based

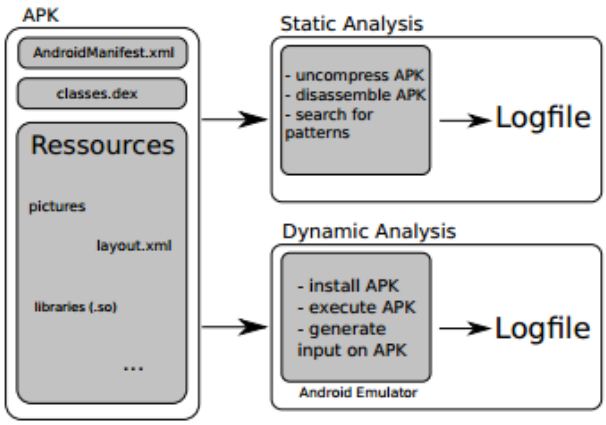

Figure 3. Design of the Android Application Sandbox (AASandbox). AASandbox consists of three main parts: the APK, static and dynamic analysis methods, and resulting dataset for further analysis.

Figure 2: AASandbox [18]

approaches. It replicates a device registered to a designated cloud, and replays the replica in the cloud through the synchronization of the device and the replica by passing the device inputs and network connections. It allows the server to perform a resource-intensive security analysis. Another similar approach is **Paranoid Android** [24]. It provides security checks on remote servers, and applies multiple detection techniques simultaneously. The difference between the two replay based approaches is that in paranoid approach the tracing and replay process are done in the application level and it has the advantage of removing the non-deterministic inputs.

Sandbox and replay based approaches are similar to our work, In sandbox approaches, the main difference is that sandbox uses user data generated by emulator, however we use user input's state data generated by de-

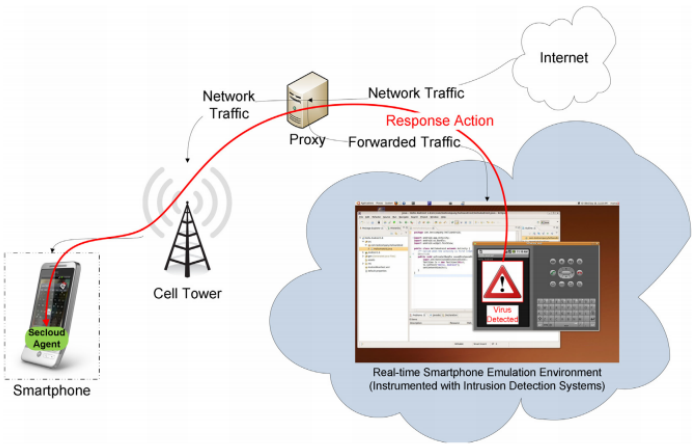

Fig. 1 – Secloud's high-level architecture.

Figure 3: Secloud [34]

vice which increases code cover and makes it hard to malwares detection to speculate the type of environment they are running in. In replay based approaches, instead of replicating all the data to the VM and by the time replay based solutions detect the suspicious behavior, the device would have already been attacked, however SORcloud offloads the required data from device to the VM using the the specific data state, which minimize the bandwidth and storage capacity , besides that the device can't fully execute the application until it has been confirmed by security models in cloud that it's out of any suspicious activity.

A study on tools of malware analaysis tools has been done to find out which one is more suitable for injection process to trigger the offloading process in SORcloud

3.1 Androgaurd

Androgaurd is Android analysis tool written by Python to disassemble and decompile Apk files, Dex/Odex, Android's binary xml and Android Resources. Androgaurd can analysis a bunch of Android apps with ipython/sublime text editor, Androgaurd uses androlyze.py which interactive tool with high level comments, it can also measure the efficiency of the obfuscators and determine if the application has been pirated or not, it also has the ability to check if the application is listed in the open source database of malwares so it will notify whether the application has been listed the black listed database. There is a risk indicator as well for suspected malicious applications. It has also ability to reverse engineering of applications including transform android's binary xml into classic xml, and also it can visualize the application with gephi [2] or with cytoscape [6] or PNG/DOT output.

3.2 Andriod-apktool

Andriod-apktool is a tool for reverse engineering, it's also capable to decode resources to original form and repacking them after having some modifications, there is also possibility to analysis and debug smali code line by line. Smali code is assembly language based on Jasmin syntax which has full functionality dex format[10]. APKTool requires you to use a separate app such as Notepad++ to edit the decompiled binaries. Once that's done, you then have to go back to APKTool to recompile the modified app [5] .

3.3 Dex2Jar

Dex2Jar is designed to read Dalvik executables format of Android application into jar format which is understandable and readable by other java reversing tools, more ever the dex2Jar is used by APKinspector to transform the jar file into understandable format Dex2jar contains of 6 components:

1- dex-reades : is designeed to read the Dalvik Excutable format.

2- dex-translator : is designed to do the convert job and it reads the dex instruction to dex-ir format after some optimize after that convert to ASM format.

3- dex-ir : used ti dex-translator to represent the dex instruction.

4- dex-tools : tools to work with ,class files like modify a apk and DeObudscate a jar

5- d2j-smali : is to disassemble dex tosmali files and assemble dex form smali files

6- dex-writer : is to write dex same way as dex-reader

3.4 Dexter

Dexter is a static web based malware analysis that allows uploading android applications which needs to be analyzed. The tool extracts as much information as possible from either legitimate or malicious applications (APKs) and displays them in various different views[16]. It shows a quick overview of all metadata and included packages of the application, more ever the dependency graph shows all included packages and its interconnections with ability to go through all the method list of each method and each method

will show list of classes and functions.

3.5 APKInspector

Apkinspector is a group of tools in one user interface. After the .apk has been loaded you can load the Smali representation of functions by selecting the function in the Methods tab in the side view. APKInspector comes with Jad, a Java decompiler. It should be able to decompile most classes, but regularly creates mistakes that either prevent a recompilation or sometimes make the class very hard to understand. Also it might fail completely in some cases, then the Smali representation must be used. There are new analysis features have been developed recently [4]:

• Reverse the Code with Ded[1] for Java Analysis

• Static Instrumentation

• Combine Permission Analysis

3.6 API monitor

API Monitor is an open source software that allows you to monitor and control API calls which are revoked by the applications and services, it's one of the most powerful applications to track down and analysis the problems occurred during running in android application, and it supports 32 bit and 64 bit applications, there are more than 13000 API's definitions from 200 DLL's plus over 17000 methods. It also decodes and displays 100 different types of unions and structures. It has the ability to display tree which explains the hierarchy of API calls and the duration, call stack for

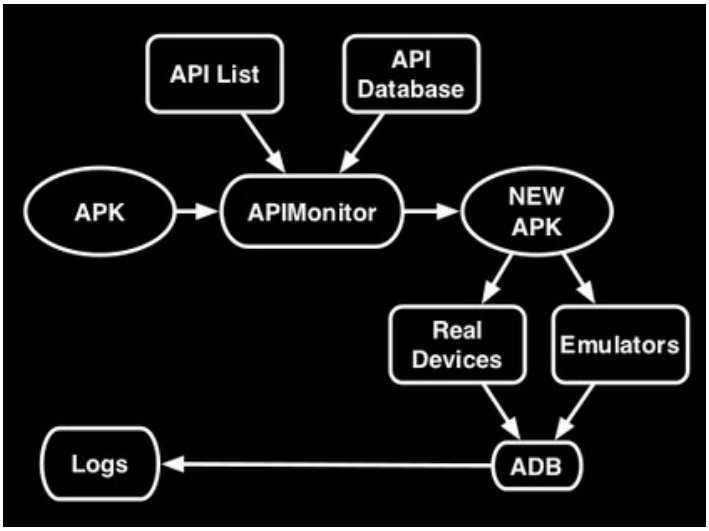

Figure 4: APImointor architecture[8]

each call.

There are so many malware android tools have been implementing in many researches, and each one of these has its own specifications and drawbacks. Also each one varies on the time consumed to transform the Dalvik executable files into jar files, for example Androgaurd has many other tool compare to other malware analysis tools and it has the ability to visualize the application with gephi or cytospace or PNG/DOT format. More ever it can determine if the application has been pirated or altered, and also it can show indicators if the application might have malicious code or even suspected. Most of tools transform the APK to Java. They, however, do not provide reverse engineering tools for compiled part of APK file.

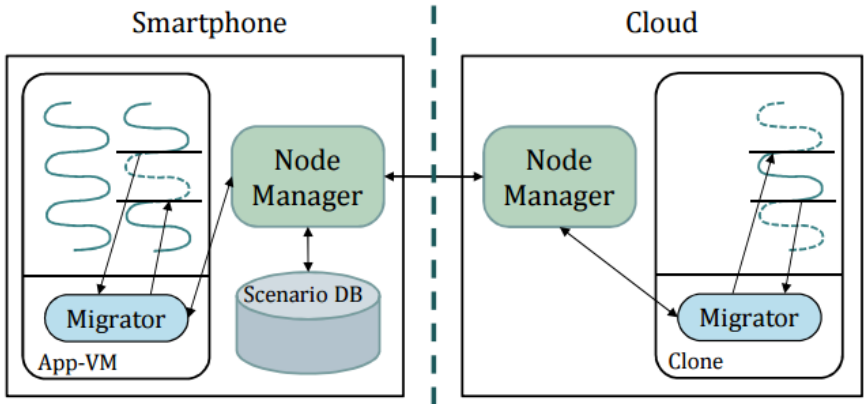

Figure 5: migration overview of CloneCloud[33]

3.7 offloading

there are many several studies try to empower mobile devices through offloading based on systems, one of the efficient mobile cloud computing is called cloneCloud proposed by sangjung Yang in paper named as fast dynamic executation offloading for efficient mobile cloud computing [33], where there were new proposed techniques to reduce the transferred data size by transferring the important heap objects which positively influences the transfer time and the efficiency of the offloading.

In figure 5, it shows when the process is started , the CloneCloud would detect the the point where it has to start offloading to the cloud,and transfer the process to the server in the other side, Also it ends the offloading process when end migration is triggered in the code

one of the other studies was done by satya-narayanan et al.[28] which is considered one of the earliest studies that enhances mobile devices with remotely connected servers, it immigrates the full VM image along with the

process in order to offload the process running on the device, on the other hand , the amount of the transfered data is considered to be around massive (a couple number of gigabytes), another approach was proposed by the same author to lower the load called VM synthesis approach [27] where small overlay of the VM is sent to nearby small cloud by mobile device where the VM is already installed with the base VM where the overlay was produced,therefore the the overlay size was reported to be smaller than the full VM size

Chapter 4

SorCloud

4.1 System Overview

I proposed a new solution called SorCloud, as a hybrid system combining advantages from sandbox based and replay based approaches , SORcloud provides both static and dynamic analysis for mobile devices. While the static analysis is performed when an application is installed into a device, the dynamic analysis inspects the behaviors of the application at run-time. According to the purpose, Sorcloud is implemented based on offloading approach to reduce the communication bandwidth between the mobile device and the cloud due the heavily bandwidth needed between both of them. there are seven modules of SORcloud which can be classified into three main categories:

1) Security modules,

2) Execution offloading modules.

3) System modules,

Each of these categories consists of sub-parts , however some of the modules will be explained in install-time and run-time.

4.2 System Modules

System modules consist of *Installer* and *Packet Manager*, packet manager is in charge of automatic forwarding packet rules in the cloud server, it receives the packet from the designated device with device identification and forwards it to the specific Virtual machine with same device identification using some rules set by SDN (a software-defined network) controller. when every mobile device registers itself into the cloud. the KVM will create VM with same device ID that the mobile has, As shown in figure 6, the main task of the packet manager is to direct the coming packets from the registered devices to the designated Virtual machine in KVM,

According to figure 6, Installing APK file requires 7 steps starting from downloading the APK from third party, and sending the APK to the packet Manager, packet manager will send it to static security analysis module, if it is free of any suspected behaviors,it will be assigned to proper VM for installion, after that, .the installing process will start into the VM in the cloud. After that APK file will be injected with the offloading code and it will be sent back to the mobile device with permission of installing. after the mobile device gets the green light from the cloud, it installs the modified version of the APK in the device. when the mobile device starts installing, it will keep synchronizing with the application in the same VM in the cloud

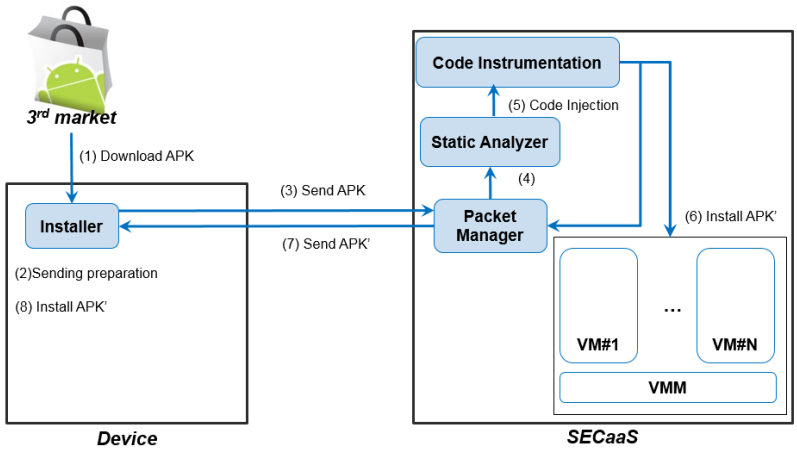

Figure 6: Install time steps

4.3 Execution offloading

Execution offloading modules consist of mainly two parts:

1) *Code Instrumentor*.

2) *Thread Manage*.

4.3.1 Code Instrumentation

For the runtime execution offloading, it is needed to determine which parts of the application code should be offloaded in order to analyze dynamic behaviors of the application at runtime. Code instrumentation is a process to inject the codes which indicate the offloading points for the thread migration. An android application usually consists of various call-back methods, which are invoked only when designated events happen. Since some of the designated events should be analyzed at runtime through offloading, the

offloading points generally correspond to call-back methods. For example, assume that a click event on a button invokes a call-back method. If the call-back method is executed in the cloud, we can monitor the behaviors of this click event.

In SORcloud, we define *target method* which is a call-back method to be monitored at runtime. There are two kinds of target methods:

1- *User Interface call-back method*

2- *Activity Life Cycle call-back method*.

- User Interface call-back methods : `onClick()`, `onLongClick()`,`onFucusChange()`,`onKey()`,`onTouch()`, `onCreateContextMenu()`
- Activity Life Cycle call-back methods : `onCreate()`,`onStart()`, `onResume()`,`onPause()`,`onStop()`,`onDestroy()`

In code instrumentation part, two dummy -empty- methods (`doMigration()` and `doRemigration()`) are declared first. The dummy methods are inserted at the beginning and end of the target method body, respectively.When `doMigration()` method is invoked at the device, the execution offloading starts. In the other way, when `doRemigration()` method is invoked at the cloud, the execution offloading ends.

However, there may be some codes in the target method body which cannot be offloaded. For example, UI related API codes, e.g., getting/setting user input data from/to a UI component , and hardware related API codes,

API	
type	function
Telephony	getDeviceSoftwareVersion() getDeviceId() getNetworkOperatorName() getCellLocation() getLine1Number() onLocationChanged(Location location)
Display	getDisplay(int displayId) getDisplays(String category)
USB connection	getDeviceName() getSerial() getDeviceProtocol()
Bluetooth	getConnectedDevices() getConnectionState) getDevicesMatchingConnectionStates
Camera	getId() createCaptureSession() createCaptureRequest() close()

Table 1: UserInterface call-back methods

e.g., reading GPS or sensor values, cannot be executed in the cloud because these codes do not work correctly in VM. Therefore, we define these methods as *non-offloadable API code* which should be executed only in the device, not in the cloud. If there are any non-offloadable API codes in the target method body, the code is executed in the device. Fig.7 shows an example of Code Instrumentation.

As Figure 7 shows how the jasmin code after inturmentation will look like, The codes written in bold are injected ones by Code instrumentor. `doMigration()` and `doRemigration()` methods are defined in (1). These methods are inserted in the target method to apply execution offloading (2) and to guarantee non-offloadable API code executed in the device (3). Through this, it is determined that which codes are executed in either device (4) or cloud (5). In order to implement Code Instrumentor, we use dex2jar[13] which is one of android reverse engineering tools. Using dex2jar, we unpack apk file and obtain dex file from it. And the dex file is converted into the jar file, after that the jar file is transformed into Jasmin format[21]. So these Jasmin codes are the input of our Code Instrumentor. After finishing instrumenting on Jasmin code, these codes are transformed and packed with apk file by using dex2jar.

4.3.2 Thread Migration

The proposed SORcloud exploits the execution offloading to monitor runtime behaviors of a unknown application by executing the application code in the cloud. More specifically, we use execution offloading technique by implementing thread migration. Offloading framework for thread mi-

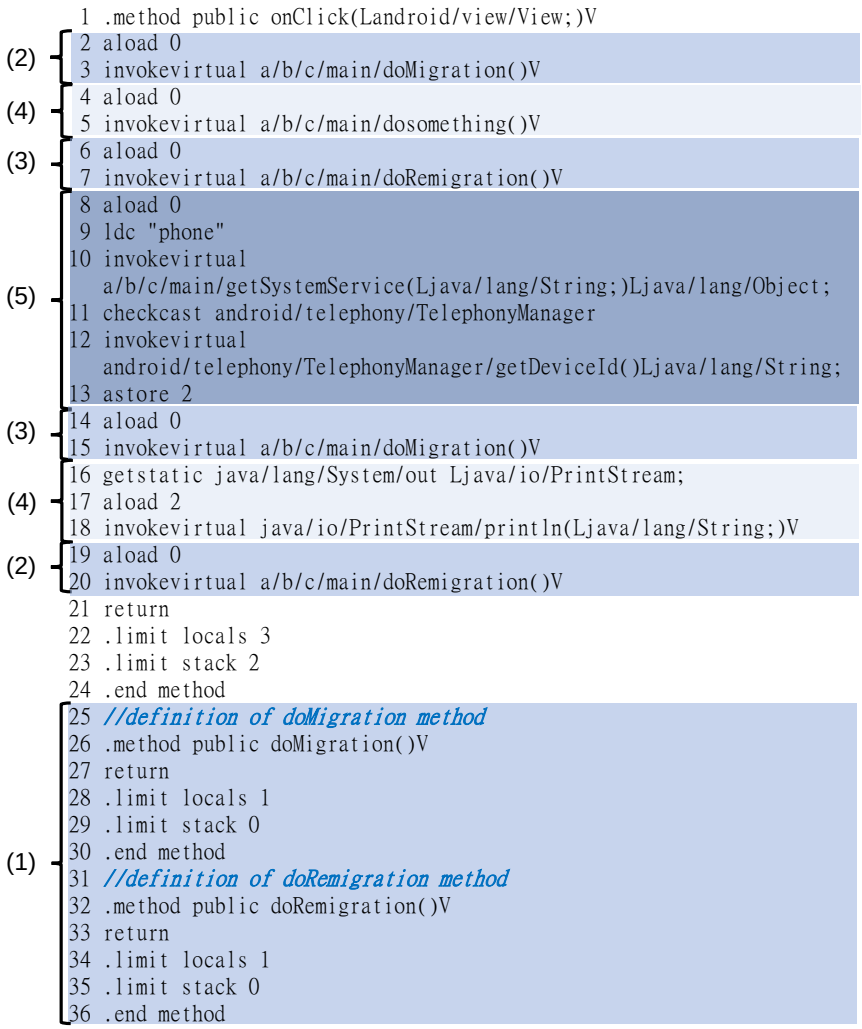

Figure 7: Example of Jasmin code after code instrumentation.

gration has been already suggested in several studies[19, 33], and we use some modules of the existing frameworks. In this subsection, it will be described what modules are being used and how these modules work. In Android framework, each android application runs on an application virtual machine(VM)[1]. Once an application VM is assigned, it allocates a thread to execute the application code. The state of the thread, i.e., program registers, call stack and heap objects, are changing while the application code is being executed. For the thread migration between a device and a server, these state should be transferred between them. This state transfer is handled by two modules, State Manager and Offloader. State Manager captures and restores the state, and Offloader sends and receives the state from a device to a server and vise verse. State Manager exists for each application VM in both a device and a server. When the codes injected by Code Instrumentor are executed, the interpreter of an application VM signals to State Manager to capture the state of the thread and to suspend execution of the thread. When State Manager receives the state from Offloader, it restores and resumes the suspended thread with the received state. Offloader implemented as Android application sends and receives state of a thread from and to state manager as well as transfers them between a device and a cloud.

4.4 Security Modules

Static Analyzer and *Dynamic Analyzer* are security modules.

[1] In this work, we use Dalvik VM because Android 4.0.3 is used in our experiment

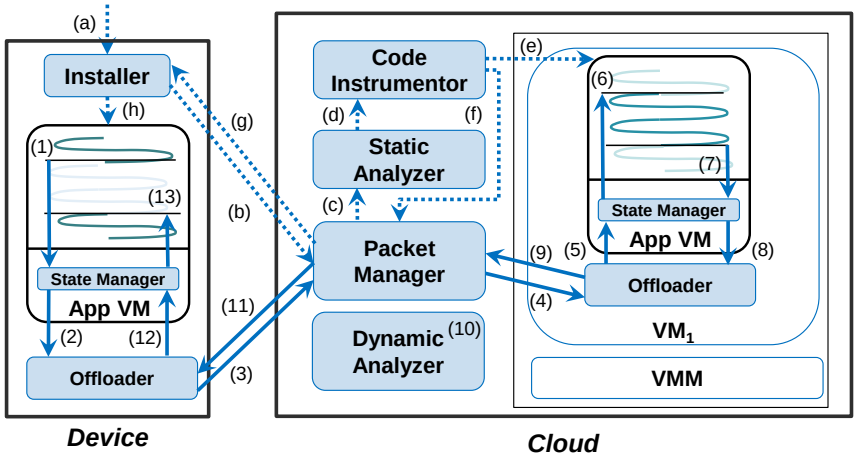

Figure 8: SORcloud overview.

These modules, furthermore, can be divided into two types according to when they work, install-time and run-time modules. In this section, we briefly explain how each module works at install-time and run-time.

Fig.8 shows the overview of SORcloud.The dotted and solid lines present the flows of static analysis and dynamic analysis, respectively. The execution offloading process at the install-time is started when a user wants to install a new application into device (a). Before the installation, the application file(.apk) is sent to Packet Manager in the cloud (b). After Static Analyzer receives the file from Packet Manager, it decides whether it is a malware or not (c). If any malicious activities are not found by Static Analyzer, the apk file would be passed to Code Instrumentor (d). Code Instrumentor inserts a bit of codes to trigger execution offloading at runtime. After that, the instrumented apk file is installed in virtual machine(VM) in cloud (e) and is sent to packet manager (f). Finally, the device receives modified apk file from the

cloud and installs it (g-h).

When the user launches the application which was instrumented at install-time, the runtime process of SORcloud begins. This process is presented with the solid line in Fig.8. When the codes inserted by Code Instrumentor trigger the execution offloading during the application execution, State Manager in the device captures the state of the current application thread, and suspends the thread (1). On receiving the state from State Manager (2), Offloader in the device passes the state to Packet Manager in the cloud (3). Packet Manager forwards the state to Offloader in VM where the application was installed (4). State Manager inside VM receives the state from Offloader, and restores the application thread and resumes the execution (5). During the execution, the dynamic analyzer monitors the behaviors, i.e., network traffic, of the application thread (6). When the execution offloading ends, State Manager in VM captures the state of the current application thread and suspend it (7). If no malicious behavior is detected during execution, the state is sent back to Offloader in the device (8-10). Then State Manager takes over this state (11), restores the application thread, and resumes the execution (12). Whenever the execution offloading occurs, this run-time process is repeated.

4.5 Security Analysis

For security analysis, SORcloud can adopt various security modules. However, this work does not focus on security modules, but on the offloading framework for security analysis. In this work, therefore, we just use two

types of security modules, Source code analysis and Network security modules. The source code analysis module, Static Analyzer in this work, analyzes the source code before the offloading and the network security module, Dynamic Analyzer, checks if the information leakage happens through the network.

As a source code analysis module, the Virustotal website tool[14] is used. This tool examines android applications and URLs with 54 different virus-scanning software products. Static Analyzer automatically sends an APK file that a user clicked on his/her device to the tool through the public APIs.

Untangle[15] is used as Dynamic Analyzer in order to prevent the information leakage by malicious application. It is an open source solution that combines the GUI web-based network management and control for network security. In this work, Dynamic Analyzer is configured to block the traffic outgoing to specific websites. It monitors traffic generated by the running application and reports filtering results.

4.6 Evaluation

4.6.1 Experimental setup

In this work, we have built the prototype of SORcloud. We used Galaxy Nexus with dual-core 1.2 Ghz CPU and 1 GB of RAM as a mobile device. For the cloud, a quad-core desktop with a 3.4GHz CPU and 32 GB of RAM running CentOS 6.5 is used. And using KVM, 2GHz core and 8GB of memory are allocated to each VM in the cloud. Packet Manager is implemented

on Software Defined Network (SDN)[11] controller. A mobile device and VMs use the same Android 4.0.3 version. State Manager is implemented by modifying Dalvik VM. Installer and Offloader are implemented as an android application. Code Instrumentor is implemented by using dex2jar[13].

4.6.2 Experimental results

4.6.2.1 Efficiency of dynamic analysis

The first concern is the networking traffic caused by transferring the state for execution offloading because the application execution may be delayed due to data exchange time. To show that SORcloud incurs the reasonable amount of traffic, we measure the traffic caused when a user executes an application remotely through Remote Desktop Protocol (RDP)[12]. The RDP provides the user with a graphical user interface to connect to a remote VM actually running the application over a network connection. there are three real world applications have been choose for testing.

4.6.2.2 TinyURL

this application is used to create and share TinyURLs on mobile device. it has many other features like converting a URL to a tinyURL before it's being shared by any application like browser, you tube and others, and also it has the ability to copy and paste the link to the clipboard

4.6.2.3 MOTP

MOTP stands for mobile one time password is a free one time synchronize authentication software for android devices, it consists of client (a J2ME MIDlet) and a server (a unix shell script), the sever part might be considered as radius server[9]

In Table 2, the table shows the data sizes as the offloading happens for each injected function in MOTP, the test was done for function by function in the program

4.6.2.4 DroidWeight

DroidWeight is a free software which allows you to track you weight and keep you updataped about your weights changes[7].

In table 3, the table shows the results of offloading functions after code instrumentation.

	migratin state collection time	buffer size	migration data size	migration state packing time	Number HeapBytes	Number stackbytes	total time
OnCreate()	16	446837	441840	2	386444	48	19
onResume()	12	451065	445952	2	389164	80	15
onclick	25	452139	446996	2	389798	52	28
onlcick(genretate)	32	451045	445928	3	389168	56	36
onresume()	47	496792	489972	3	417195	32	52
onclickprofile_Motp	42	424747	420468	10	375780	60	58
onlickprof_HOTP	48	424747	420468	5	375780	60	58
onlcickprof-totp	37	424747	420468	7	375780	60	47

Table 2: Offloading data for MOTP

	migratin state collection time	buffer size	migration data size	migration state packing time	Number HeapBytes	Number stackbytes	total time
creat_entry	34	613360	605248	5	507840	40	42
delete_entry	37	912355	903036	4	798263	44	43
oncreate	14	160837	157788	2	121972	44	16
oncreate1	12	160837	157788	1	121972	44	13
Onresume()	19	677347	665872	4	537264	32	26
onDestory()	31	587646	579746	4	486522	36	36
BMI Calc	41	652193	642496	5	526491	40	48
BMI Calc (go)	46	779255	768208	4	637251	52	52
show thers	27	2729695	2721660	9	2631565	16	39
months button	24	2729811	2721780	9	2631569	48	35

Table 3: Offloading data for DroidWieght

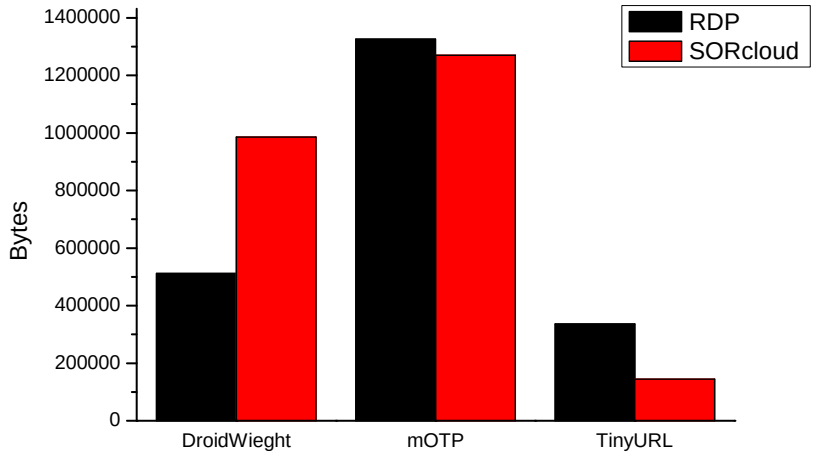

Figure 9: Run-time data transfer overhead

4.6.2.5 comparison

The execution offloading traffic and the RDP traffic are compared for three real world android applications: TinyURL[3], mOTP[9] and DroidWeight[7].

Fig.9 shows the average sizes of the transferred data while the applications are being executed with the same user scenario. In the cases of TinyURL and mOTP, SORcloud incurs less network traffic than the RDP solution. However, when DroidWeight is running, SORcloud incurs more data traffic. these results can be explained as follows. In SORcloud, there is no data transfer when a part of the application code is not offloaded. However, the more frequent offloading causes more network traffic. In the RDP solution, upload data for the user inputs and download data for the screen display are transferred continuously, even in the idle state. Although the amount of network traffic varies according to the type of application, we

```
1 public void DetectEmulator() {
2     // doMigration()
3     // ...
4     // doRemigration()
5     String devicename = Build.DEVICE;
6     // doMigration()
7     Log.d("SORcloud", devicename);
8     if (devicename.equals("laptop")) {
9         // in case of emulator(or VM)
10    } else {
11        // in case of device, do something.
12        String html =
              DownloadHtml("http://facebook.com/");
13        System.out.println(html);
14    }
15    // doRemigration()
16 }
```

Figure 10: Sample Java code of emulator detection

can say SORcloud is comparable with the RDP. That means SORcloud is sufficient to execute applications in real time.

4.6.2.6 Security enhancement

Since many recent approaches use a mobile device emulator for dynamic analysis to detect malwares, malware developers devise techniques to evade the malware detection. One of popular techniques is to stop a malware working in an emulator. Therefore, malware developers exploit some APIs to check the running environment [25, 30].

Although SORcloud also relies on a VM-based emulator in the cloud, malwares have no way to figure out their running environments since the non-offloadable API codes are executed only in the mobile device.

Fig.10 shows a example code for emulation detection. The code at line

Figure 11: Result of network monitor[15]

5 is to get name of device, and the code at line 7-14 is the actual behavior based on the name of device. According to the execution environment, the behavior of this code would be different. If this code is executed in an emulator, we cannot detect the malicious code since nothing happens. On the other hand, in SORcloud, the mobile device name is obtained since the code line 5 is executed in the mobile device. And the state of thread including the object for a device name are migrated to the cloud. Therefore, even if the code at line 7-14 is executed in the emulator, we can detect the malicious behavior as if this code is running in the real mobile device. Fig.11 shows the network behavior is monitored by Dynamic Analyzer.

4.7 CONCLUSION

In this thesis, SORcloud is a cloud based solution for detecting mobile android malware statically and dynamically. The execution offloading technique is introduced to monitor the runtime behavior of applications. SORcloud overcomes limitations of the existing approaches, code coverage problem and security degradation. It is shown that SORcloud can detect efficiently malicious behaviors of unknown applications at runtime.

As Table 4 shows briefly the three common approaches on mobile device security, offloading has better user interactivity compare to sandbox and replay, it consumes less power and one of the important features that it does not have code coverage problem plus it can prevent the attack before it attacks

	Sandbox	Replay	Offloading
User interactivity	No	A little	much
Power consumption	No	much (initial overhead)	A little
Code coverage	Yes	No	No
Prevention	Yes	No	Yes

Table 4: Comparison of Sandbox, replay, and offloading

4.8 FUTURE WORK

Since SORcloud is an extensible cloud based framework, it can easily add or remove security modules. Therefore, it will be the first step to add more security modules such as System Call Monitor and Taint Analyzer to monitor various dynamic behaviors.

SORcloud does not examine the non-offloadable APIs in order to hinder malwares from figuring out the running environment. However, since it may be asked if the non-offloadable APIs are safe, a mechanism to monitor the behaviors of them need to be considered.

References

[1]

[2] androguard - reverse engineering, malware and goodware analysis of android applications ... and more (ninja !) - google project hosting. https://code.google.com/p/androguard/. (Visited on 05/19/2015).

[3] android-tinyurl - tinyurl intergration for the android flatform. https://code.google.com/p/android-tinyurl/.

[4] apkinspector - **moved to github** apkinspector is a powerful gui tool for analysts to analyze the android applications. - google project hosting. https://code.google.com/p/apkinspector/. (Visited on 05/19/2015).

[5] Apktool - a tool for reverse engineering android apk files. https://ibotpeaches.github.io/Apktool/. (Visited on 05/19/2015).

[6] Cytoscape: An open source platform for complex network analysis and visualization. http://www.cytoscape.org/. (Visited on 05/19/2015).

[7] droidweight - a weight tracking android app. https://code.google.com/p/droidweight/.

[8] Kelwin yang, student at google summer of code 2012 — slideshare. http://www.slideshare.net/KelwinYang/. (Visited on 05/19/2015).

[9] motp - one time passwords for android. https://code.google.com/p/motp/.

[10] smali - an assembler/disassembler for android's dex format - google project hosting. https://code.google.com/p/smali/. (Visited on 05/19/2015).

[11] Software defined network. http://en.wikipedia.org/wiki/VirusTotal.

[12] Spice. http://www.spice-space.org/.

[13] Tools to work with android .dex and java .class files. https://code.google.com/p/dex2jar/.

[14] Virustotal. http://en.wikipedia.org/wiki/VirusTotal.

[15] Web filter lite. http://wiki.untangle.com/index.php/Web_Filter_Lite.

[16] Welcome to dexter's documentation! — dexter 1.0 documentation. http://dexter.dexlabs.org/static/docs/. (Visited on 05/19/2015).

[17] Droidbox - android application sandbox. https://code.google.com/p/droidbox, Feb. 2015.

[18] T. Blasing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak. An android application sandbox system for suspicious software detection. In *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, pages 55–62. IEEE, 2010.

[19] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[20] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[21] I. K. Jon Meyer, Daniel Reynaud. Jasmin home page. `http://jasmin.sourceforge.net/`, 2004.

[22] P. Lantz, A. Desnos, and K. Yang. Droidbox: Android application sandbox, 2012.

[23] D. Maslennikov. Jasmin home page. `http://securelist.com/analysis/publications/36996/mobile-malware-evolution-part-6/`, 2013.

[24] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356. ACM, 2010.

[25] T. Raffetseder, C. Kruegel, and E. Kirda. Detecting system emulators. In *Information Security*, pages 1–18. Springer, 2007.

[26] V. Rastogi, Y. Chen, and W. Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220. ACM, 2013.

[27] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[28] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, D. R. O'Hallaron, A. Surie, A. Wolbach, J. Harkes, A. Perrig, D. J. Farber, et al. Pervasive personal computing in an internet suspend/resume system. *Internet Computing, IEEE*, 11(2):16–25, 2007.

[29] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann. Mobile-sandbox: Having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1808–1815. ACM, 2013.

[30] T. Vidas and N. Christin. Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 447–458. ACM, 2014.

[31] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen, and C. Platzer. Andrubis: Android malware under

the magnifying glass. *Vienna University of Technology, Tech. Rep. TRISECLAB-0414-001*, 2014.

[32] B.-g. C. L. P. C. J. J. P. M. William Enck, Peter Gilbert and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.

[33] S. Yang, Y. Kwon, Y. Cho, H. Yi, D. Kwon, J. Youn, and Y. Paek. Fast dynamic execution offloading for efficient mobile cloud computing. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 20–28. IEEE, 2013.

[34] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders. Secloud: A cloud-based comprehensive and lightweight security solution for smartphones. *Computers & Security*, 37:215–227, 2013.

english abstract

Abstract

Cloud Security Solution of Mobile Applications based on Execution Offloading

지금까지 모바일기기를 악성 공격으로부터 방어하기 위한 기술들은 모바일 기기의 제한된 자원으로 인해 어려움을 겪어왔다. 이러한 상황에서 한가지 해결방안은 원격에 존재하는 클라우드환경에서의 가상머신을 통해 이러한 자원을 많이 소비하는 보안 분석을 모바일 기기대신 실행하도록 하는 것이다. 하지만 기존 클라우드를 사용한 방법들은 여전히 코드 커버리지 문제나 보안성 약화와 같은 문제점이 존재한다. 따라서 본 연구에서는 클라우드를 이용해 정적, 동적분석을 실시하는 SORcloud라는 새로운 클라우드기반의 해결책을 제안한다. 동적분석을 위해 SORcloud는 의심스러운 어플리케이션의 프로세스를 원격의 가상머신에서 실행되게 하였고, 이를 통해 기존 클라우드 기반의 방법들이 가지는 단점들을 해결하였다. 그리고 실험을 통해 제안하는 해결책이 얼마나 효율적으로 동작하고, 악성행위를 검출해 내는지를 보였다.

Keywords : 수행 오프로딩, 클라우드 시스템, 정적 및 동적 분석

Student Number : 2013-23848