An Effective Cloud Solution to Ensure the Integrity of Mobile Application via Execution Offloading

Donghyun Kwon, Ali Almokthar, Jungsoo Park, Minho Park, Souhwan Jung, and Yunheung Paek

***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 paper 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.**

***Index Terms*—Dynamic analysis, execution offloading, malware, mobile cloud computing.**

I. Introduction

The number of malware 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 malware 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 malware. 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 malware 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 paper, therefore, every separate server is assumed to operate as the VM in the cloud. This kind of solutions can be broadly classified into two different approaches, sandbox-based and replay-based.

Firstly, the sandbox-based approach [1]-[6] 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 program 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 [7], [8], 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 tradeoff between two different approaches.

In this paper, we propose a new approach to overcome the tradeoff, which is an offloading-based security solution for mobile devices called SORcloud (Security ORiented cloud).

Manuscript received September 14, 2015; revised November 25, 2015.

Donghyun Kwon, Ali Almokthar, and Yunheung Paek are with Seoul National University, Korea (e-mail: ypaek@sor.snu.ac.kr).

Jungsoo Park, Seolah Je, Minho Park, and Souhwan Jung are with Soongsil University, Korea (e-mail: {mhp, souhwanj}@ssu.ac.kr).

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

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 of this paper is organized as follows. In Section II, we discuss related work. In Section III, we explain SORcloud framework design and implementation. In Section IV, we evaluate SORcloud. We conclude in Section V and finally in Section VI, we discuss future work.

II. Related Work

An android application sandbox system called *AAsandbox* [1] 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* [2], 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* [3] 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* [4] 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* [5] 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* [6] 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 [9]. *Secloud* [7] is one of replay based 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* [8]. 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 device which increases code cover and makes it hard to malware 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 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.

III. SORcloud Framework

A. System Overview

As a hybrid system, SORcloud provides both static and dynamic analysis. 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, the seven modules of SORcloud can be classified into three categories, security modules, execution offloading modules and system modules. *Static Analyzer* and *Dynamic Analyzer* are security modules, *State Manager*, *Offloader* and *Code Instrumentor* are execution offloading modules, and Installer and Packet Manager are system modules. 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.

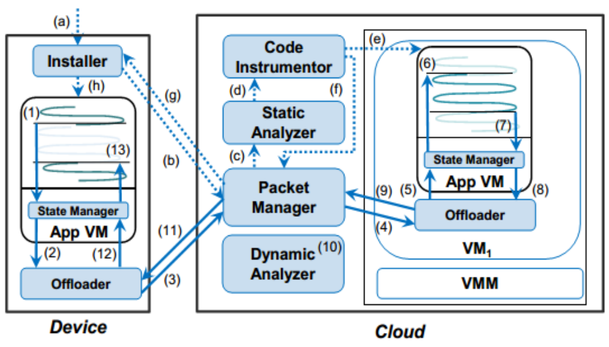


Fig. 1. SORcloud overview. The dotted and solid lines present the flows of static analysis and dynamic analysis, respectively.

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. 1. When the code 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

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

thread, and resumes the execution (12). Whenever the execution offloading occurs, this run-time process is repeated.

B. 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 code which indicates 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. It is noteworthy that SORcloud does not have code coverage problem like replay-based approaches because the call-back methods corresponding real-time user inputs are executed in cloud. 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: *User Interface call-back method* and *Activity Life Cycle call-back method*.

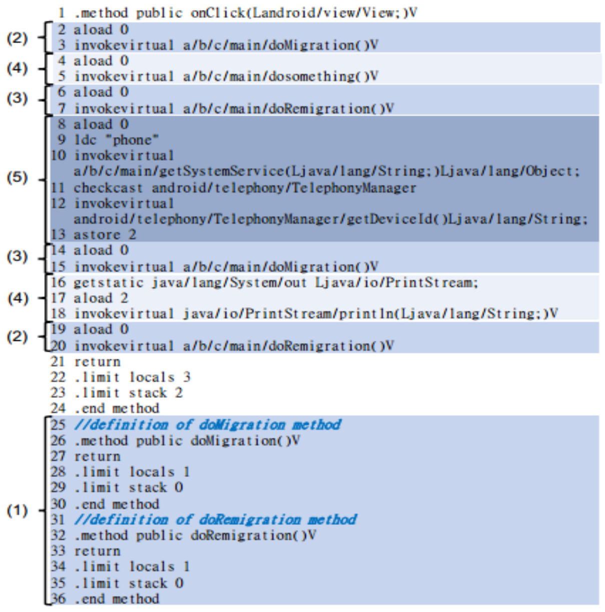
1) User Interface call-back methods : onClick(), onLongClick(), onFucusChange(), onKey(), onTouch(), onCreateContextMenu()
2) Activity Life Cycle call-back methods : onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy()

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 code in the target method body which cannot be offloaded. For example, UI related API code, e.g., getting/setting user input data from/to a UI component, and hardware related API code cannot be executed in the cloud because these 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 code in the target method body, the code is executed in the device. Fig. 2 shows an example of Code Instrumentation.

C. Thread Migration

The proposed SORcloud exploits the execution offloading to monitor runtime behaviors of an 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 migration has been already suggested in several studies [10], [11], 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.



```
.method public onClick(Landroid/view/View;)V
aload 0
invokevirtual a/b/c/main/doMigration()V
aload 0
invokevirtual a/b/c/main/dosomething()V
aload 0
invokevirtual a/b/c/main/doRemigration()V
aload 0
ldc "phone"
invokevirtual
  a/b/c/main/getSystemService(Ljava/lang/String;)Ljava/lang/Object;
checkcast android/telephony/TelephonyManager
invokevirtual
  android/telephony/TelephonyManager/getDeviceId()Ljava/lang/String;
astore 2
aload 0
invokevirtual a/b/c/main/doMigration()V
getstatic java/lang/System/out Ljava/io/PrintStream;
aload 2
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
aload 0
invokevirtual a/b/c/main/doRemigration()V
return
.limit locals 3
.limit stack 2
.end method
//definition of doMigration method
.method public doMigration()V
return
.limit locals 1
.limit stack 0
.end method
//definition of doRemigration method
.method public doRemigration()V
return
.limit locals 1
.limit stack 0
.end method
```

Fig. 2. Example of code instrumentation. The code 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 Android framework, each android application runs on an application virtual machine (VM)[3]. 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 states 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 code injected by Code Instrumentor is 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.

D. 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

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

if the information leakage happens through the network.

As a source code analysis module, the Virustotal website tool [12] 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 [13] 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.

IV. Evaluation

A. 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) 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 [14].

B. Experimental Results

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) [15]. The RDP provides the user with a graphical user interface to connect to a remote VM actually running the application over a network connection. The execution offloading traffic and the RDP traffic are compared for three real world android applications: TinyURL, mOTP and DroidWeight.

Fig. 3 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 can say SORcloud is comparable with the RDP. That means SORcloud is sufficient to execute applications in real time.

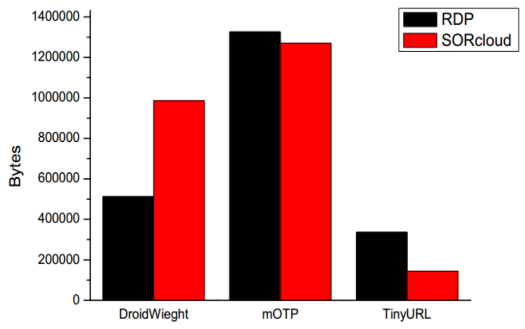
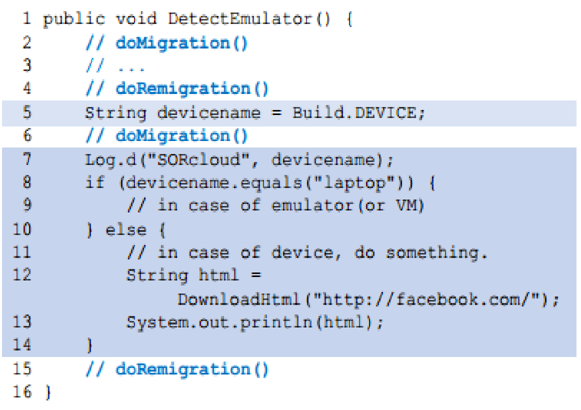


Fig. 3. Run-time data transfer overhead.

Security enhancement. Since many recent approaches use a mobile device emulator for dynamic analysis to detect malware, 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 [16], [17].



```
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 }
```

Fig. 4. Sample Java code of emulator detection.

Although SORcloud also relies on a VM-based emulator in the cloud, malware have no way to figure out their running environments since the hardware related API code is executed only in the mobile device.

Fig. 4 shows a simple example code for emulation detection. The code at line 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 hardware related code at line 5 is executed in the mobile device. And the states 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.

V. Conclusion

In this paper, we proposed SORcloud which 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.

VI. 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 needs to be considered.

Acknowledgment

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-H8501-15-1008) supervised by the IITP (Institute for Information & communications Technology Promotion), the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2014R1A2A1A10051792), IDEC, the Brain Korea 21 Plus Project in 2015 and Inter-University Semiconductor Research Center (ISRC).

References

[1] T. Blasing, L. Batyuk, A. -D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proc. the 2010 5th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 55-62, 2010.
[2] V. Rastogi, Y. Chen, and W. Enck, "appsplayground: Automatic security analysis of smartphone applications," in *Proc. the Third ACM Conference on Data and Application Security and Privacy*, pp. 209-220, ACM, 2013.
[3] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a deeper look into android applications," in *Proc. the 28th Annual ACM Symposium on Applied Computing*, pp. 1808-1815, ACM, 2013.
[4] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. V. D. Veen, and C. Platzer, "Andrubis: Android malware under the magnifying glass," Vienna University of Technology, Tech. Rep. TRISECLAB-0414-001, 2014.
[5] 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)*, vol. 32, no. 2, 2014.
[6] P. Lantz, A. Desnos, and K. Yang, *Droidbox: Android Application Sandbox*, 2012.
[7] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders, "Secloud: A cloud-based comprehensive and lightweight security solution for smartphones," *Computers & Security*, vol. 37, pp. 215-227, 2013.
[8] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: Versatile protection for smartphones," in *Proc. the 26th Annual Computer Security Applications Conference*, pp. 347-356. ACM, 2010.
[9] Droidbox — Android Application Sandbox. [Online]. Available: https://github.com/pjlantz/droidbox.
[10] B. -G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. the Sixth Conference on Computer Systems*, pp. 301-314, ACM, 2011.
[11] 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 *Proc. 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 20-28, 2013.
[12] Virustotal. [Online]. Available: http://en.wikipedia.org/wiki/VirusTotal.
[13] Web filter lite. [Online]. Available: http://wiki.untangle.com/index.php/Web_Filter_Lite.
[14] Tools to work with android .dex and java .class files. [Online]. Available: https://code.google.com/p/dex2jar/
[15] Spice. [Online]. Available: http://www.spice-space.org/
[16] T. Raffetseder, C. Kruegel, and E. Kirda, "Detecting system emulators," in *Proc. of 10th International Conference on Information Security*, pp. 1-18, Springer, 2007.
[17] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders, "Secloud: A cloud-based comprehensive and lightweight security solution for smartphones," *Computers & Security*, vol. 37, pp. 215-227, 2013.

Dong-Hyun Kwon received the BSc degree in electrical and computing engineering from the Seoul National University, Korea in 2012. He is currently working towards the PhD degree in electrical and computing engineering from the Seoul National University, Korea. His research interests include mobile cloud computing and mobile system security.

Ali Almokhtar received the BSc degree in information technology from Multimedia University, Malaysia in 2009. He received his MSc degree from Seoul National University, Korea in 2015 and is currently working towards the PhD degree in electrical and computer engineering from the Seoul National University, Korea. His research interests include mobile cloud computing and big data.

Jung-Soo Park received the B.S and M.S. degrees in electronics engineering from Soongsil University in 2013 and 2015, respectively. He is currently studying for Ph.D. course in electronics engineering from Soongsil University. His research interests include cloud security, mobile security, and identity and access Management system.

Min-Ho Park received the BS and MS degrees in electronics engineering from Korea University in 2000 and 2002, respectively. He received the PhD degree in the School of Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea, in 2010. He is an assistant professor in School of Electronic Engineering, Soongsil University, Seoul, Korea. He was at Samsung Electronics from 2002 to 2004. As a postdoctoral researcher, he was at Carnegie Mellon University for two years since 2011.

Before the postdoctoral researcher at CMU, he was a senior engineer for 3GPP LTE S/W Development Group of Samsung Electronics. His current research interests include wireless networks, vehicular communication networks, network security, and cloud computing.

Souhwan Jung received the B.S and M.S. degrees in electronics engineering from Seoul National University in 1985 and 1987, respectively, and the Ph.D. degree from the University of Washington, Seattle, USA in 1996. From 1996 to 1997 he was a senior software engineer at Stellar One Corporation, Bellevue, USA. In 1997, he joined the School of Electronic Engineering at Soongsil University, Seoul, Korea, and currently serves as a professor. He is an executive director of the Korea Institute of Information Security and Cryptology. He was also a R&D program director of Ministry of Knowledge Economy in Korea for information security area from 2009 to 2011. His research area includes wireless network security, cloud security, mobile security, identity and access management system, and IoT security.

Yun-Heung Paek received the BSc and MSc degrees in computer engineering from the Seoul National University, Korea in 1988 and 1990, respectively. He received his PhD degree in computer science from University of Illinois at Urbana-Champaign in 1997. Currently he is a professor at the Department of Electrical and Computing Engineering, Seoul National University, Korea. His research interests include mobile cloud computing, embedded security systems and re-targetable compiler