Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.1, Feb. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.1.169

DroidSecure: 안드로이드 어플리케이션 권한 상승 완화를 위한 기술에 대한 연구*

응웬부렁,[†] 정 수 환[‡]
숭실대학교

DroidSecure: A Technique to Mitigate Privilege Escalation in Android Application*

Long Nguyen-Vu,[†] Souhwan Jung[‡]
Soongsil University

요 약

안드로이드 플랫폼은 사용자 친화적으로 설계되어 있다. 하지만 이러한 친화적 설계는 취약점이 쉽게 발생할 수 있고, 일반적인 사용자는 쉽게 탐지가 어렵다는 단점을 가지고 있다. 따라서, 본 논문에서는 안드로이드 어플리케이션 분석을 위한 유명한 오픈 소스 분석 도구를 설명하고, 현재 구글의 권한 그룹에 대한 정책의 위험성을 설명한 후 공격자의 권한 상승에 대한 위험을 완화하기 위한 기법을 제안한다. 또한, 21,064의 악성코드 샘플을 조사하여 제안한 기술이 안전하지 않은 응용 프로그램 업데이트 탐지에 대한 증명을 하였을 뿐 아니라 보안 위협에 대한 인식을 고취시키고자 하였다.

ABSTRACT

Android platform is designed to be user-friendly, yet sometimes its convenience introduces vulnerabilities that normal users cannot justify. In this paper, after making an overview of popular open source analysis tools for android applications, we point out the dangerous use of Permission Group in current Google Policy, and suggest a technique to mitigate the risks of privilege escalation that attackers are taking advantage of. By conducting the investigation of 21,064 malware samples, we conclude that the proposed technique is considered effective in detecting insecure application update, as well as giving users the heads-up in security awareness.

Keywords: Android Security, Privilege Escalation, Mobile Malware

I. Introduction

Android has been dominating in the market for the recent years, with 78% of the market-share in the first quarter of 2015, according to IDC [1]. 2014 witnessed an astounding increase of Android malware rate with 75% in the United States compared to the previous year [2].

Malware also come in different forms

Received(09. 03. 2015), Modified(12. 04. 2015), Accepted(12. 18. 2015)

* 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학IT연구센터육성 지원사업의 연구결과로 수행되었습니다(IITP-2015-H8501-15-1008); 또한, 본 연구는 산업통상자원부 및 한국산업기술평가관리원의 우수기술연구센터(ATC) 사업의 일환으로 수행하였음. [10045904, 클라우드 컴퓨팅 환경하에서 정보보안 서비스를 제공하기 위한 SecaaS 프레임워크 원천기술 개발과 이를 이용한 1Gbps급 모바일 정보 유출방지 서비스 구축]

† 주저자, longnv@ssu.ac.kr

‡ 교신저자, souhwanj@ssu.ac.kr (Corresponding author)

Table 1. An overview of open source Malware Analysis Tool

	Static	Dynamic	Assessment	Analysis	System call/API	Signature-based	Last commit
Androguard	x		x	x	x	x	14 days ago
Androwarn	x			x			2 years ago
APKinspector	x			x	x		2 years ago
DidFail	X			X	x		a day ago
Amandroid	X		x	x	x	x	7 days ago
CFGScanDroid	X				x	x	5 months ago
Maldrolyzer	X			X			a month ago
Ella	x			x			20 days ago
DroidBox		x		x	x		A month ago
TaintDroid		x		x	x		a year ago
AndroidHooker		x		x	x		5 months ago

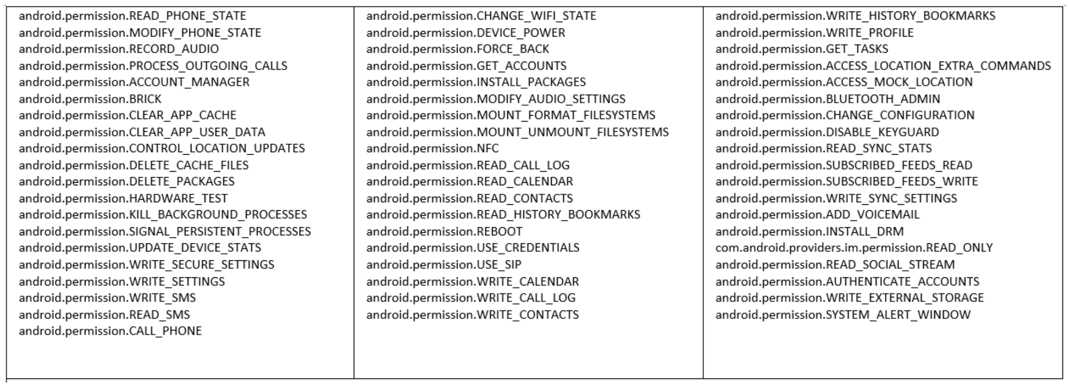
like ransomware (a type of malware that force users pay money to unlock their devices), adware (aggressive advertising application) or chargeware (application associated with premium services like SMS or Phone Call). We have done a survey with 50 different open source analysis tools and realized that half of them were in the state of inactive, the others attract very low contribution from community. Besides few ongoing projects like Androguard or Amandroid, ones could be mentioned are: Androwarn, DidFail, DroidBox. Description of the most 11 popular tools [16-26] is provided in Table 1 (The latest update of this table is in 2015, June 18th).

Google requires developers to declare Permissions in AndroidManifest.xml file. If there are permissions that relate to the same topic, then they will belong to a Permission Group. For example, receive text messages and send text messages belong to the same group of permissions (namely, SMS). Whenever users install an app from Google Play Store, they need to accept all individual permissions given by the developer, or the app will not be installed. The users also cannot modify the permissions of current apps on their phones after they were installed. To improve user experience (UX), Android provides automatic-updates capability so that users do not need to review all of installed apps in their devices every time if there is a new release.

According to current Google policy about app permissions [3]: if you have automatic-updates turned on and you already have an application installed with one permission in Permission Group, then when the app upgrades, user will not need to review or accept that Permission Groups again, even if it requires more permissions in that group. In other word, unless the user turns off auto-updates feature, the app will gain more permissions without needing any approval from the user when it upgrades. Moreover, users no longer see android.permission. INTERNET permission in the approved list anymore because it has been granted to all applications, by default. This issue of Permission Group can be reproduced by creating an app with basic permissions and adding more permissions in the same Permission Group for the next upgrade.

Besides traditional installation through

Table 2. List of dangerous Android Permissions declared in AndroidManifest.xml



android.permission.READ_PHONE_STATE	android.permission.CHANGE_WIFI_STATE	android.permission.WRITE_HISTORY_BOOKMARKS
android.permission.MODIFY_PHONE_STATE	android.permission.DEVICE_POWER	android.permission.WRITE_PROFILE
android.permission.RECORD_AUDIO	android.permission.FORCE_BACK	android.permission.GET_TASKS
android.permission.PROCESS_OUTGOING_CALLS	android.permission.GET_ACCOUNTS	android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
android.permission.ACCOUNT_MANAGER	android.permission.INSTALL_PACKAGES	android.permission.ACCESS_MOCK_LOCATION
android.permission.BRICK	android.permission.MODIFY_AUDIO_SETTINGS	android.permission.BLUETOOTH_ADMIN
android.permission.CLEAR_APP_CACHE	android.permission.MOUNT_FORMAT_FILESYSTEMS	android.permission.CHANGE_CONFIGURATION
android.permission.CLEAR_APP_USER_DATA	android.permission.MOUNT_UNMOUNT_FILESYSTEMS	android.permission.DISABLE_KEYGUARD
android.permission.CONTROL_LOCATION_UPDATES	android.permission.NFC	android.permission.READ_SYNC_STATS
android.permission.DELETE_CACHE_FILES	android.permission.READ_CALL_LOG	android.permission.SUBSCRIBED_FEEDS_READ
android.permission.DELETE_PACKAGES	android.permission.READ_CALENDAR	android.permission.SUBSCRIBED_FEEDS_WRITE
android.permission.HARDWARE_TEST	android.permission.READ_CONTACTS	android.permission.WRITE_SYNC_SETTINGS
android.permission.KILL_BACKGROUND_PROCESSES	android.permission.READ_HISTORY_BOOKMARKS	android.permission.ADD_VOICEMAIL
android.permission.SIGNAL_PERSISTENT_PROCESSES	android.permission.REBOOT	android.permission.INSTALL_DRM
android.permission.UPDATE_DEVICE_STATS	android.permission.USE_CREDENTIALS	com.android.providers.im.permission.READ_ONLY
android.permission.WRITE_SECURE_SETTINGS	android.permission.USE_SIP	android.permission.READ_SOCIAL_STREAM
android.permission.WRITE_SETTINGS	android.permission.WRITE_CALENDAR	android.permission.AUTHENTICATE_ACCOUNTS
android.permission.WRITE_SMS	android.permission.WRITE_CALL_LOG	android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_SMS	android.permission.WRITE_CONTACTS	android.permission.SYSTEM_ALERT_WINDOW
android.permission.CALL_PHONE		

Play Store, users can install application from unknown sources. By disabling security feature in android OS, users can install almost any APK (Android Application Package) files which are downloaded from the Internet. That means the privacy and security of users are exposed to many different types of malicious applications. These applications are aggressive in requiring permissions described in AndroidMafinest file, some of them even try to imitate the same package names as official apps in Play Store [27].

Our contribution in this paper can be summarized as follows:

(1) Provide the latest status of open source malware analysis tools that we have surveyed in this year. These are valuable resources for empirical studies to any researchers in the field.

(2) Discuss the danger of privilege escalation that results from Permission Group in Android and propose a technique to mitigate the risks.

(3) Investigate the trend of malware in the recent years and discuss some malware behaviors based of the analysis result.

(4) Conduct a thorough review of 21,064 Android applications and discover the malicious intention of 22 malware that use the same package name with apps in Play Store. Especially, there are 2 malware are extremely dangerous as they have the same certificate fingerprints with the official ones in Play Store, that means the malware developers already had the private keys of these genuine apps.

The rest of this paper is organized as follows: Section II discusses some related works to android malware analysis. In section III, we introduce the cloud model for detecting privilege escalation. We provide the analysis statistics in section IV and the conclusion in section V.

II. Related Works

Yajin Zhou et al. implement DroidRanger [4] which is the combination of static and dynamic analysis. They separate DroidRanger into two parts:

(1) Detecting known Android malware: Firstly, permission-based filtering filters applications with important permissions like SEND_SMS, RECEIVE_SMS, etc in order to narrow down the number of applications to analyze. After that, behavioral footprint matching is used to detect harmful behaviors through a set of matching rules. Each rule expresses a

sequence of APIs (or information like broadcast receivers) that will be called.

(2) Detecting unknown Android malware: heuristics-based filtering and dynamic execution monitoring are based on anomaly behaviors and dynamic analyses of system calls for suspicious actions. The authors use the kernel module to hook and log system calls selectively. The next step is for manual review if suspicious runtime behaviors were found.

While permission-based filtering is used to filter applications with essential permissions (in order to reduce the number of apps that are needed to analyze), this very first step fails to counter update attack if the malware only requires basic permissions and then silently escalates its privilege by requesting more permissions later on. We also found another research of analyzing Manifest file to detect malware in Android : Borja Sanz et al. [5] conduct an empirical validation of Android malware through machine learning. This technique also does not consider the risk when updating applications, and is not effective to prevent update attack. Our system will try to solve the existing problem by keeping track of each application as they are installed on the device. We will describe in detail of DroidSecure in the next Section.

III. Privilege Escalation Detection Model

The workaround for the problem of Permission Group is temporarily turning off automatic-updates, but this leads to another problem: the app is not at its newest versions at some time, which is also unsecure for users if there is an unpatched vulnerability.

As mentioned in the previous section, DroidRanger [4] and PUMA [5] accidently skip two important characteristics of Android applications, which are Permission Group and automatic-updates. To mitigate the risk caused by this implementation of Android platform, we implement DroidSecure, an Android application that could be able to:

(1) Collect permissions from all installed applications, and then categorize them into sets. Each set represents for 1 application.

(2) Classify permissions of each set into groups, based on Permission Group described by Google,

(3) Use BroadcastReceiver to listen to ACTION_PACKAGE_REPLACED, so that if there is an upgrade of any application, it will trigger our tool to compare with the permissions previously listed in the sets. Any action that tries to add more permissions which belong to our predefined dangerous permissions (Table 2) are considered suspicious, and users will get a warning about that.

Currently in Android OS, there are 31 Permission Groups that are needed to be investigated. [6]. However, in developer web page at the time being[7], Google has taken down some of them, we believe there will be an update of this page corresponding to the release of Android Marshmallow.

Although our implementation on Android phone is effective against aggressive updates, we take a step further by moving the system to the cloud. We try to address the problem with apps that are installed from unknown sources (when users have already disabled the security feature). Whenever a user tries to download an

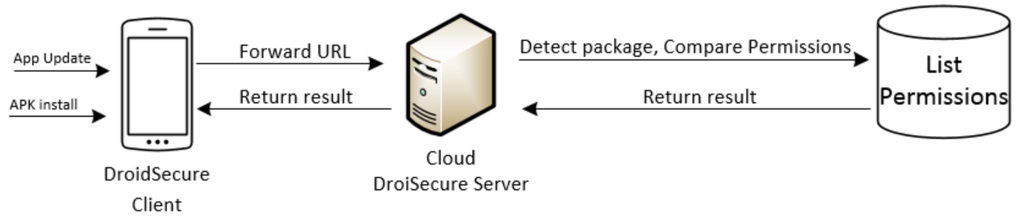


Fig. 1. The implementation of DroidSecure

APK file from other markets (instead of Google Play), DroidSecure triggers the crawler to download that APK from Play Store (since the two have the same package name, defined inside AndroidManifest.xml), investigates the permissions required by the app, and justify with the previous versions (in the case this app has been installed on the phone of the user). In other words, our system tracks down every action related to installation or update/upgrade process of an app. The advantage of this approach is that DroidSecure client which was installed in the device does not consume too much battery power, as well as the power of cloud allows our system to work more efficiently. The scenario is described in Fig. 1.

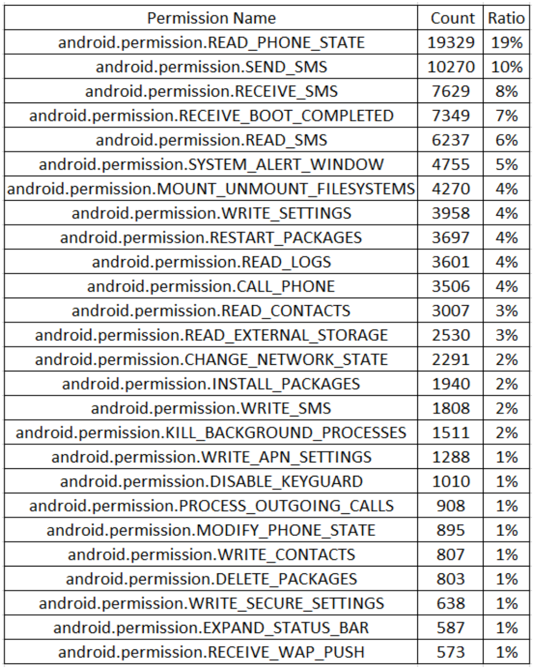
IV. Assessment Analysis

To prove the effectiveness of preventing an application to gain dangerous permission of the proposed technique, we conduct an analysis over 21,064 malicious applications to clarify the aggressiveness of privilege escalation of these apps, as well as provide the understanding of current trend that attackers are using in AndroidManifest declaration. Some of the most dangerous permissions such as android.permission.BRICK or android. permission.SYSTEM_ALERT_WINDOW (that can permantly locks the phone) also appear in our result. We believe such declarations are aimed for rooted phone, where most of security features are disabled or easily bypassed under root power.

As shown in Table 3, we extract and analyze 88 permissions from more than 21,064 malware samples. The result indicates the domination of 26 permissions presented in the table (occupies 95% of the total permissions). Attacker usually develop malware that could be able to steal money from users through premium services (SMS or Phone Call), or ransomware that require users to pay money in order to unlock their phones (by using SYSTEM_ALERT_WINDOW, DISABLE_KEYGUARD). Besides single, deadly permissions like SYSTEM_ALERT_WINDOW or KILL_BACKGROUND_PROCESSES, we can recognize some groups of permissions here. For example: SEND_SMS, RECEIVE_SMS or READ_CONTACTS, WRITE_CONTACTS. Such types of permissions in Permission Group cause no harm when they are used separately, but when the application updates and the app escalates its privilege by requiring more permissions, the phone can be compromised at some levels, or completely.

We also randomly select 878 applications that were downloaded from Google Play Store to analyze their manifest permissions and package names, and then

Table 3. Permission Ratio in Malware



Permission Name	Count	Ratio
android.permission.READ_PHONE_STATE	19329	19%
android.permission.SEND_SMS	10270	10%
android.permission.RECEIVE_SMS	7629	8%
android.permission.RECEIVE_BOOT_COMPLETED	7349	7%
android.permission.READ_SMS	6237	6%
android.permission.SYSTEM_ALERT_WINDOW	4755	5%
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	4270	4%
android.permission.WRITE_SETTINGS	3958	4%
android.permission.RESTART_PACKAGES	3697	4%
android.permission.READ_LOGS	3601	4%
android.permission.CALL_PHONE	3506	4%
android.permission.READ_CONTACTS	3007	3%
android.permission.READ_EXTERNAL_STORAGE	2530	3%
android.permission.CHANGE_NETWORK_STATE	2291	2%
android.permission.INSTALL_PACKAGES	1940	2%
android.permission.WRITE_SMS	1808	2%
android.permission.KILL_BACKGROUND_PROCESSES	1511	2%
android.permission.WRITE_APN_SETTINGS	1288	1%
android.permission.DISABLE_KEYGUARD	1010	1%
android.permission.PROCESS_OUTGOING_CALLS	908	1%
android.permission.MODIFY_PHONE_STATE	895	1%
android.permission.WRITE_CONTACTS	807	1%
android.permission.DELETE_PACKAGES	803	1%
android.permission.WRITE_SECURE_SETTINGS	638	1%
android.permission.EXPAND_STATUS_BAR	587	1%
android.permission.RECEIVE_WAP_PUSH	573	1%

compare with our malware dataset. The evaluation is performed between 18,494,192 app pairs, resulted in 41 applications (in our malware dataset) which have similar package names with 22 applications downloaded from Play Store. We narrow down the scope by investigating their certificate fingerprints inside META-INF directories, and found 3 serious cases of privilege escalation [9-15]. Table 4 describes the package name, permissions escalation (include the number of permissions in the original app and the number of permissions are added in the malware), and the Malware Family which are recognized by different Anti-Virus software in VirusTotal. Each of these apps also has good reputation in Play Store with more than 10 million installations, which facilitate the widespread of these disguised malware.

Especially, there are two cases that have the same certificate fingerprints with the original apps in Google Play Store (tv.pps.mobile and com.outfit7.talkingsantafree). That means when user installs these malicious apps, the Package Manager will allow them to be installed since they have the same certificate fingerprints with the ones in Google Play Store. We believe this is a serious problem, as the private key of the developers have been leaked to the outside world, and their keys are being abused by these malicious apps.

Table 4. 3 malware families with privilege escalation intention

Package name	Permissions raise	Malware Family
tv.pps.mobile	43 to 76	Android/ SystemMonitor
com.scompa. facechanger	8 to 21	Minimob
com.outfit7. talkingsantafree	15 to 19	Android/ SMSKey.L

To summarize our assessment analysis:

(1) We provide a feature to eliminate special permissions that are used by malware to compromise rooted devices, so that users with rooted phones could have safety at some levels, even though the feature allows "installation from unknown source is enabled"

(2) With the combination of Permission Group, users could avoid unsafe upgrade of applications in Play Store, or privilege escalation through third party applications.

(3) We also found the problem of similar certificate fingerprints between official applications and malware. This may lead to a breach that can be abused to spread malware to many devices.

V. Conclusion

In this paper, we have a brief introduction of the popular analysis tools and their newest updates. These tools play an important role in Android malware analysis field. After that, we discuss our concern about privilege escalation of malware apps inside and outside AndroidManifest, by taking advantage of current Google policy in the platform. Our experiment shows the risk caused by Permission Group and similar Package Name installation can let the malware completely compromise the Android system. While the downloaded apps from Google Play are still relatively small compared to the whole Play Store, we expect to have a thorough investigation with more than 1 million apps in the near future. The technique of malware that use similar package name is more prevalent than we could expect, it urges Google policy to be more rigorous in permission and installation management.

In the near future, we plan to extend our current system to work with the new scheme of App Permissions in Android Marshmallow while still maintain the backward compatibility with Lollipop and Kitkat. The drastic changes in Android Marshmallow leads to the removal of Permission Group [28], which could bring some pitfalls to the newly developed product. Furthermore, we expect to have a thorough investigation with applications in Google Play Store by increasing the number of crawled apps to 20,000 (1000 apps at least for each category in Google Play). That is an important step to have more precise statistics with the current trends, as well as a better insight of malware analysis.

References

〔1〕 Smartphone OS Market Share, Q1 2015,http://www.idc.com/prodserv/smartphone-os-market-share.jsp

〔2〕 2014 Mobile Threat Report, https://www.lookout.com/resources/reports/mobile-threat-report

〔3〕 Android Group Permissions, https://support.google.com/googleplay/answer/6014972?p=app_permissions

〔4〕 Yajin Zhou, Zhi Wang, Wu Zhou and Xuxian Jiang, "Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," Proceedings of the 19th Network and Distributed System Security Symposium, Feb. 2012.

〔5〕 Borja Sanz et al., "PUMA: Permission Usage to Detect Malware in Android," International Joint Conference CISIS'12-ICEUTE´12-SOCO´12, pp. 289-298, 2013

〔6〕 Android Permission Group, https://web.archive.org/web/20150319134451/https://developer.android.com/reference/android/Manifest.permission_group.html.

〔7〕 Android Permission Group Update, https://developer.android.com/reference/android/Manifest.permission_group.html

〔8〕 Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M., and Rajarajan, M. "Android Security: A Survey of Issues, Malware Penetration and Defenses," Communications Surveys & Tutorials, vol.17, no.2, pp. 998-1022, 2015.

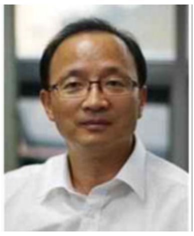
〔9〕 Play Store App: PPS (for Mobile), https://play.google.com/store/apps/details?id=tv.pps.mobile

〔10〕 Malware Android/System Monitor, https://www.virustotal.com/en/file/c98465d75f31591b53345974eaa638faf0807f94ef5f694c633fe4f6d5f547a3/analysis/1440845487/

〔11〕 Play Store App: Face Changer, https://play.google.com/store/apps/details?id=com.scoompa.facechanger
〔12〕 Malware Android/AdDisplay, https://www.virustotal.com/en/file/d26327e28c624bfbd99c45035344ccdbc125e8f30b9aace842dc40f029825a0b/analysis/1440848439/
〔13〕 Play Store App: Talking Stanta, https://play.google.com/store/apps/details?id=com.outfit7.talkingsantafree
〔14〕 Malware SMSKey1, https://www.virustotal.com/en/file/788b5b0b06cdfcd4f3d162b1090d722a7aae37c114d518eceae1730ceec6b070/analysis/1440853733/
〔15〕 Malware SMSKey2, https://www.virustotal.com/en/file/ca04bc361f83d028138c65cc88110ce1ab27e14423715e8070c2486e200e2205/analysis/1440853768/
〔16〕 Androguard, https://github.com/androguard/androguard
〔17〕 Androwarn, https://github.com/maaaaz/androwarn
〔18〕 APKinspector, https://github.com/honeynet/apkinspector
〔19〕 DidFail, https://www.cs.cmu.edu/~wklieber/didfail
〔20〕 Amandroid, https://github.com/sireum/amandroid
〔21〕 CFGScanDroid, https://github.com/douggard/CFGScanDroid
〔22〕 Maldrolyzer, https://github.com/maldroid/maldrolyzer
〔23〕 Ella, https://github.com/saswatanand/ella
〔24〕 Droidbox, https://code.google.com/p/droidbox
〔25〕 TaintDroid, https://github.com/TaintDroid
〔26〕 AndroidHooker, https://github.com/AndroidHooker/hooker
〔27〕 Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., and Vigna, G, "Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications," Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS) Feb. 2014
〔28〕 Android M Permissions: https://www.androidpit.com/android-m-permissions-explained

〈저 자 소 개〉

응웬부렁 (Long Nguyen-Vu) 학생회원
2008년 9월~2012년 9월: Vietnam National University of Information Technology
2014년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈관심분야〉 클라우드 보안, 모바일 보안, 네트워크 보안



정 수 환 (Souhwan Jung) 종신회원
1985년 2월: 서울대학교 전자공학과 졸업
1987년 2월: 서울대학교 전자공학과 석사
1996년 6월: University of Washington 박사
1988년~1991년: 한국통신 전임 연구원
1997년~현재: 숭실대학교 전자정보공학부 교수
〈관심분야〉 클라우드 보안, 모바일 보안, 네트워크 보안