AndroTracker: Creator Information based Android Malware Classification System

Hyun Jae Kang[1], Jae-wook Jang[1], Aziz Mohaisen[2], and Huy Kang Kim[1]

[1] Korea University, Seoul, Korea
{trifle19,changkr,cenda}@korea.ac.kr
[2] Verisign Labs, Reston, VA 20190, USA
amohaisen@gmail.com

Abstract. Thousands of malicious applications targeting mobile devices, including the popular Android platform, are created every day. A large number of those applications are created by a small number of professional underground actors, however previous studies overlooked such information as a feature in detecting and classifying malware, and in attributing malware to creators. Guided by this insight, we propose a method to improve on the performance of Android malware detection by incorporating the creator's information as a feature and classify malicious applications into similar groups. We developed a system called AndroTracker that implements this method in practice. AndroTracker enables fast detection of malware by using creator information such as serial number of certificate. Additionally, it analyzes malicious behaviors and permissions to increase detection accuracy. AndroTracker also can classify malware based on similarity scoring. Finally, AndroTracker shows detection and classification performance with 99% and 90% accuracy respectively.

Keywords: Mobile malware, Android security, malware detection, malware classification, creator information

1 Introduction

The rapid advancements in mobile platforms, such as smartphones, have paved the way for a more connected world featured by massive role of information sharing. The year-over-year reported growth of the smartphones market, which is accompanied by a decay in the use of conventional platforms—such as personal computers, indicates the start of a new era. However, the increased use of smartphones also has many pitfalls. Top among them is mobile security: the Android operating system and platform—by far the largest mobile OS by market share—is mainly being targeted by mobile malware. For example, F-secure, a major security vendor, reported 827 new families or variants of mobile malware 2013, where the overwhelming majority of them (804 families) are based on the Android platform [2].

To address this problem, various efforts are initiated in academia and industry for analyzing, detecting, and classifying mobile malware. Those efforts belong to two major schools of thoughts on analysis: static and dynamic. Static analysis-based techniques mostly analyze the use of requested permissions to decide if an application is malicious or not. However, these approaches have low accuracy, mainly because many benign applications demand excessive permissions as malicious applications. On the other hand, dynamic analysis-based techniques usually enable the use of features extracted from the execution of a running application for deciding whether it is malicious or not. However, this technique detects malicious behavior only under certain circumstances: no evasion techniques are applied by the malware and a timely execution of the malicious code, among other conditions. Furthermore, this technique is computationally expensive, requiring live execution or emulation of the application for features collection. To this end, and for their light-weight characteristics, we advocate the static analysis techniques. To address their shortcoming of low accuracy, we aim to improve them by incorporating new features.

While the Android platform is flexible in providing authors of applications means for application distribution, applications are often attributed to their authors using certificates tied to their author's signing credentials. While the certificates and their metadata driven features can be used for identifying authors of reputed applications, they can also be used for identifying authors of problematic applications, although that is not examined previously in the literature. We hypothesize that certain developers have a significant role in the creation and distribution of malicious apps most prevalent to today's mobile security landscape. Indeed, we find that as much as 70% of the malicious applications in a dataset sample discussed in this paper are using 4% of the total number of certificates used for the overall applications.

To realize this insight, we design a comprehensive system called AndroTracker. AndroTracker has two modules: a light-weight detector and a similarity based classifier. To this end, the contributions of this work are as follows:

- First, we introduce a light-weight detection method for Android malware. For that, AndroTracker uses creator's information to filter malware for the baseline analysis. Using the serial number of a certificate as a feature, the system filters potential malware for detection. For other malware, the system analyzes particular parts of applications based on functionality and permission to achieve high detection accuracy.
- Second, we introduce a classification method for the detected malware samples. Based on a similarity scoring algorithm, AndroidTracker classifies malware into similar families. The algorithm compares the API call sequence, permission, and system command extracted from the malware as features.
- We introduce an automation procedure and productization of AndroTracker. We tested the system using 51,179 benign applications and 4,554 malware samples. Our system labeled more than 99% of the benign applications correctly, and labeled the malicious applications more than 90% to their proper family correctly, showing promising results for both detection and classification.

2 Related Works

There are two major approaches for analyzing Android malware: static analysis and dynamic analysis. Static analysis is a way of checking functionalities and maliciousness of an application by analyzing its source code—without executing the application. This approach is useful for finding maliciousness in applications with a behavior triggered by certain conditions—where the behavior-based approach fails. On the other hand, dynamic analysis is a method to examine an application during runtime. The approach may miss some parts of the code that are not executed, but it can easily reveal certain malicious behaviors that are too complicated to unveil with static analysis.

Analyzing the requested permissions is a popular way in static analysis. Enck et al. [3] proposed Kirin, which certifies an application at the installation time using a set of predefined security rules. They analyzed the type of malicious behaviors and defined the rules configured with permission and intent information. AdDroid separates privilege from advertising framework to Android platform [4] and prevents advertising library from accessing sensitive information allowed for other permission of the application. Felt et al. [5] analyzed real-world mobile malware from 2009 to 2011. Also, they discussed the effectiveness of using permission distribution in order to classify benign application and malware. Peng et al. [10] applied simple Naive Bayes to requested permissions, and developed hierarchical mixture model for classification. Also, Sarma et al. [11] compared distribution of permissions between benign application and malware, then determined critical permissions to detect malware. Permission based malware detection approach has a major weakness seen in its low accuracy. The problem arises from the laxity of Android permission architecture. Application developers can request permissions that are not necessary; and thus the distribution of permissions becomes incorrect.

There are several dynamic analysis based approaches. AppsPlayground performs functions like information leakage detection, sensitive API monitoring, and kernel level monitoring [7]. Burguera et al. [9] proposed Crowdroid, which monitors and logs system calls and sends them to a central server. At the server side, the system operates a K-means clustering to classify benign app and malware. Jang et al. [12] analyzed integrated system logs including system calls, generated while malware running on an emulator. Their system, Andro-profiler, produces a behavior profile with analyzed system logs in human-readable form, detects, and classifies malware with high accuracy.

There are also hybrid approaches that adopt both static analysis and dynamic analysis. Zheng et al. [6] extracted an API call sequence and checked dynamic downloading of malicious code, then generated three levels of signatures. These signatures facilitate identifying malicious code segments, along with class association among applications. Spreitzenbarth et al. [8] presented an automated analyzing system, Mobile-Sandbox. It parses an application's permissions and intent information, and analyzes the suspiciousness of them. Then, it performs dynamic analysis to log actions—especially those based on native API calls.

3 Features and Overview of AndroTracker

In this section, we summarize the major threats from Android malware. Then, we list features used in profiling applications; serial number of certificate, API, system command, intent, and permission. Finally, we introduce our system, AndroTracker.

3.1 Threats from Android Malware

Malware can infect Android devices via many infection routes; as a downloaded application from visits to malicious websites, spam, malicious SMS messages, and malware-bearing advertisements [1]. After malware infects a target device, behaviors of the malware can be categorized depending on their purpose. Zhou and Jiang [13] classified malicious behaviors into privilege escalation, remote control, financial charge, and information collection. Also, Seo et al. [14] listed monetization, information stealing, mobile botnet, and root privilege acquisition as categories. Through examining these behaviors, we rearranged those to particular functions to use in our malware detector module; the functions are using system commands on root privilege, causing financial fraud by hiding SMS notification from the user, and collecting sensitive information.

3.2 Feature Selection

Serial number of certificate: When distributing an application, the creator signs it by his private key and a standard certificate of the public key is generated. There are blanks for the creator's name, organization, and location, while generating a certificate. However, the creator can fill them up with false information since the process has no steps of confirmation. The certificate has a unique serial number according to RFC 2459, the X.509 standard. Based on that, one can check whether certificates are the same or not by comparing the serial number.

Intent: The Android OS has a distinctive structure. Applications on Android do not have a unique entry that programs usually have on other OS. They are made up of Android components; activity, service, broadcast receiver, and content provider. Activity is a UI component related to the screen while service is a background process which is invisible to the user. Broadcast receiver waits for the signals from the system and wakes up the proper actions after receiving. Content provider plays a role of an intermediate unit to share data between applications. These four components work individually, therefore a unit for delivering messages is needed; namely intent. Intent transfers from activity to activity, containing specific instructions about what the application wants.

API: API (Application Programming Interface), documented in Android SDK, is a set of functions provided to control principal actions of Android OS. It is much efficient to consider certain APIs often used by malware, rather than extracting all APIs from the source code of an application. Seo et al. [14] analyzed malware samples and determined suspicious APIs often used by malware. They listed suspicious APIs and compared the number of use between malware and benign applications. We manually collected additional APIs by checking all APIs in Android SDK, which operate in a way similar to the suspicious APIs defined in [14]. These APIs are related to functions such as collecting the user or device information, accessing websites, sending and deleting SMS, and installing an application.

System command: Seo et al. [14] listed commonly used commands by malware. We listed them by excluding the commands used only in a few malware samples. 'chmod', 'insmod', 'su', 'mount', 'sh', 'killall', 'reboot', 'mkdir', 'getprop', 'ln', and 'ps' are commands often used by malware, and they run on rooted Android device. If any of those strings is found in the source code of an application, we mark it as malware. The commands are executed after the malware obtains root privilege on the device. Also, our list contains ‘gingerbreak’ and ‘rageaginstthecage’ because they are root exploits.

Permission: 122 permissions are provided in Android platform to inform the user what actions will be performed and which resources will be accessed by an application. Sarma et al. [11] compared two dataset, applications from the Android Market and malicious applications. They analyzed the distribution of permissions requested by each dataset and found 26 risky permissions to the security and privacy. We use those permissions in our system. One of the permissions, INTERNET, is excluded, since this permission is required for most applications. We regarded that relatively small difference between the percent of applications requested from Android market applications and malware will not play an important role in detection and classification. INSTALL_PACKAGES, a permission usually used for installing a new package downloaded from a server, is included instead. While requested permissions are notified before download, there is another way to extract permissions by analyzing call-graph. Au et al. [15] specified the list of permissions required by every API call, and provided the permission mappings. In our system, we extracted 26 critical permissions applying PScout mapping, along with requested permission. We named this feature as API-related permission.

3.3 System Overview

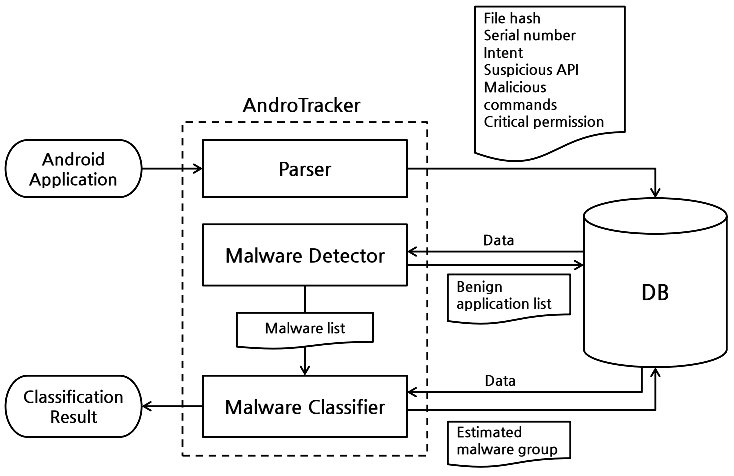

Fig. 1. Overview of AndroTracker

We designed AndroTracker, a system for detecting and classifying Android malware. Fig. 1 shows the overall architecture of AndroTracker. The system is composed of three parts; parser module, malware detector module, and malware classifier module. First, the parser module extracts strings like certificate information, APIs, permissions, commands, and intents from an application. Then, the malware detector module checks whether a certificate serial number of the application is included in a predefined serial number blacklist. If not, it additionally analyzes the application based on other features like APIs, intents, system commands, and permissions. Finally, the malware classifier module groups malware with same families by comparing the similarity of API sequence, permissions, and system commands between the applications. The parsed data and outputs of detector and classifier modules are updated in a shared database.

4 Malware Detection in AndroTracker

In section 3, we defined the representative malicious behaviors using the root privileged system commands, concealing SMS notification to charge rate, and so forth. These kinds of well-known malicious behaviors can be detected by AndroTracker. AndroTracker uses the serial number of certificates as one of the features to detect Android malware. Also, permission based rule is applied as one of the detection rules.

We collected malware samples from malware repository websites such as Contagio Mobile [16], VirusShare [17], and Malware.lu [18]. The samples are collected during January to August 2013. We used malware description of F-Secure antivirus to tag representative family association and labels. Based on the F-Secure antivirus description, malware families that include too many samples were cut down, and families that have only few samples were excluded. The number of refined malware samples is 4,554. Also, we collected 51,179 applications during the same period by downloading from the Android market, GooglePlay, and assumed that they are benign.[3]

4.1 Serial Number Blacklist

Among the collected malware samples, we extracted a serial number of each sample's certificate. 622 serial numbers were observed in total malware samples, but only 24 unique serial numbers comprised 70% of the samples, and surprisingly, 50% of malware samples were signed by five serial numbers. That means particular serial numbers are frequently used by malware creators. However, the counted number of malware signed by the same certificate serial number may have an error, because the counted number depends on the collected samples. In addition, we analyzed commonly found serial numbers in the various malware families and their variants. Among the 622 serial numbers, there were 137 numbers that generated more than 2 families or variants of malware. We made a blacklist of serial numbers from the resulting certificates. The serial numbers creating only one type of malware were excluded from the blacklist. The serial numbers found in over two malware families or variants were included. '93:6e:ac:be:07:f2:01:df' was deleted in the list, since it is a serial number of standard test key for native applications that are built in a device or an emulator. As a result, the serial number blacklist had 136 numbers for malware detection.

[3] Our dataset is available at `http://ocslab.hksecurity.net/andro-tracker`

4.2 Likelihood Ratio of Permission

Benign applications and malware groups have different tendencies of requesting permissions. Malware may request more permissions than benign applications, or may often request permissions that have privacy or financial fraud risks. We analyzed the distribution of the critical permissions in each benign application and malware samples we have in order to calculate likelihoods of the permissions.A permission such as SEND_SMS was required for 1.82% and 1.04% of benign applications, and 43.98% and 35.07% of malware samples, according to requested and API-related permission respectively.

The likelihood of permission on each category can be calculated by Naive Bayes Classifier. Au et al. [15] found out that most of the Android permissions have subtle correlation with any other permission, so we can assume that they are relatively independent. Let n and m be the number of applications and the number of critical permissions respectively. The permission vector for application i is $a_i = (a_{i,1}, a_{i,2}, ..., a_{i,m})$, where $a_{i,j}$ is 1 if application i uses permission j, and otherwise 0. Also, we put $c_i \in$\{benign, malicious\} which indicates category of application i. Using Bayes' Theorem and assuming that $P(c_i = malicious) = P(c_i = benign)$, the likelihood ratio Λ is

$$\Lambda(a_i) = \frac{P(c_i = malicious|a_i)}{P(c_i = benign|a_i)} = \prod_{j=1}^{m} \frac{P(a_{i,j}|c_i = malicious)}{P(a_{i,j}|c_i = benign)} \ .$$

An application is supposed to have a variable of any category value following a uniform distribution. If one of the conditional probabilities is zero, then the whole multiplication becomes zero. To avoid such case, the conditional probabilities are calculated using the Laplace estimation, $P(a_{i,j}|c_i) = \frac{\sum_{i=1}^{n} a_{i,j}+1}{n+2}$. An application's likelihood ratio is compared with a predefined threshold value T_L in order to detect malware.

4.3 Malware Detection

Our detection algorithm is designed with mainly three ideas: 1. existence of certain serial numbers generating many kinds of malware, 2. malicious behaviors like command usage on root privilege, hiding SMS notification, and collecting sensitive information, and 3. high likelihood of malware under given distribution of permissions.

The detection algorithm starts by checking the application's serial number of a certificate for fast scanning by applying the blacklist we established in section 4. There were a few applications that have serial number in the blacklist but do not use any suspicious APIs. The step excludes them to avoid over-detecting. Secondly, the algorithm checks the usage of the system commands listed in section 3. These commands, which run on rooted device or to root a device, are found in malicious codes.

The next step is finding malware that conceals SMS notification. Malware that behaves in this way has a purpose of subscribing to premium services confirming and noticing by SMS. These applications use sendTextMessage() to send SMS. Also, they get the highest priority of SMS receiving intent and call abortBroadcast() to hide a notification of SMS to other applications and users. The step checks whether an application uses above methods and an intent filter, to catch such malware. For the final step, the algorithm adopts a permission based detection rule. It calculates likelihood ratio under given distribution of permissions. Two likelihood ratios are obtained using requested critical permissions and API-related critical permissions. To complement a limitation of permission based detection method, it checks whether an application sends SMS or collects sensitive information like device ID, phone number, serial number of SIM card, and location of the device. Collecting more than two kinds of the information is considered sufficiently suspicious.

5 Malware Classification in AndroTracker

Malware applications detected by our malware detector are fed into the malware classifier module. Suspicious API strings, malicious commands, and critical permissions are used for similarity scoring. These features are chosen because they are highly related to the behavior of malware.

5.1 Similarity Scoring

A similarity score between two malware is computed by using each similarity S_i of suspicious API strings, malicious commands, and critical permissions. The similarity score SS can be written as $SS = \sum_i w_i \cdot S_i$ where w_i is a weight of relevant similarity. By default, for all i, $w_i = 1/3$ to set up same weights to three similarities.

All suspicious APIs are substituted by a matched character. The suspicious API string of an application, which is made of substituted characters in parsed order, reflects the calling sequence of APIs. The similarity between two strings is calculated using Needleman-Wunsch algorithm that finds the best shared alignment of two sequences. The similarity of malicious commands is measured by applying the Jaccard coefficient. The Jaccard coefficient computes the number of elements in the intersection divided by the number of elements in the union. The order of the malicious commands is not considered. The critical permissions are compared using the Levenshtein distance. This metric calculates a minimum number of character edits to make two strings the same. It is meaningless to consider the order of permissions, so we applied the Levenshtein distance after sorting the strings. A value of similarity is calculated as the number of edits over the maximum length of two strings. Each similarity of requested permissions and API-related permissions is computed, and the average of two values is used for the similarity of critical permissions.

5.2 Malware Classification

Our classifier module makes groups of similar malware by comparing between a malicious application's signature and each group's signature. A signature is a set of suspicious API string, malicious commands, and requested/API-related permissions of a malicious application. In the case of a group, it is same as the signature of the first application included in the group. A sample loaded into the system is treated as follows:

1. A similarity score is computed between signatures of the sample and the existing group. The score is computed with all existing groups, each.

2. The highest value of similarity scores, $max(SS)$, is chosen.
3. The chosen value is compared with similarity threshold T_S. If $max(SS) \geqslant T_S$, the sample is included in the corresponding group. If $max(SS) < T_S$, the sample results in a new group, and the signature of it represents the group's signature.

6 Experiment

We implemented our system AndroTracker developed by Python 2.7 using the algorithms explained in section 4 and 5. It is tested on collected dataset mentioned at the beginning of section 4. We applied 5-fold cross-validation to test our system. Samples are randomly divided into five equal size subsamples. Only one subsample is used as test data, and it repeats for five times to check for all each subsample.

6.1 Extracting Elements

An APK, which is an Android package, is a compressed file that contains META-INF, lib, res, and assets directories and AndroidManifest.xml, classes.dex, and resources.arsc files. Serial number is extracted from META-INF directory that includes information of the certificate. From AndroidManifest.xml, the application name, requested permission, component and intent can be collected. Classes.dex file is disassembled to obtain a smali code, which is a Dalvik virtual machine code. By following the codes used by components, the system extracts suspicious API, malicious command, and API-related permission. These elements are saved in a database that provides necessary data to detector module and classifier module in AndroTracker.

6.2 Detection and Classification Results

The system was configured with the threshold values $T_L = 1$ and $T_S = 0.7$. 423 benign applications, corresponding to 0.83% of all benign application, were detected as malware. A majority of false positives were detected by the serial number blacklist. On the other hand, the false negatives, measured as malicious samples predicted as benign, were only nine. From the point of view of malware analyst, it is important to have low false negatives, although increased false positives may be annoying. The rule may be a bit 'loose' for accurate detection. However, this substantially reduces false negatives. To find out why some of the benign applications were determined as malicious in our system, we checked the 423 false positives using descriptions from VirusTotal [19], a popular malware scanning site. Interestingly, 329 applications were reported as malware. It implies that applications downloaded from GooglePlay are not perfectly reliable as benign. The descriptions of false positives were applied to offset the classification result.

The classification accuracy of each category is analyzed in Table 1. Accuracy in the table is defined as correctly classified samples in the category divided by the number of all samples the category contains. The average of malware classification accuracy is 90%, and the average of total is 98%.

Table 1. Classification Results for each Category

	Category	Accuracy	Category	Accuracy
Malware (4,554)	AdWo(1,910)	**0.97**	AirPush(243)	0.76
	Boxer(755)	**0.99**	Counterclank(68)	0.68
	DroidDream(14)	0.57	DroidKungFu(24)	0.79
	FakeApp(17)	**0.94**	FakeBattScar(82)	**1.00**
	FakeInst(709)	0.88	FakeNotify(82)	0.82
	Gappusin(153)	0.65	GinMaster(115)	0.72
	Kmin(48)	0.88	OpFake(130)	0.65
	PremiumSMS(25)	0.44	Ropin(64)	0.66
	Smshider(17)	**1.00**	SmsReg(14)	0.64
	SmsSend(10)	0.00	SMStado(74)	**0.97**
Benign app(51,179)		**0.99**		
Average		**0.98**		

Applications in categories like Adwo, Boxer, FakeApp, FakeBattScar, Smshider, and SMStado were classified with high accuracy. However, performance of classifying sample sets like DroidDream, PremiumSMS, and SmsSend were low. The main reason was an overlap of malicious behavior. For example, SmsSend sample which is another version of OpFake according to the analysis of F-Secure, often classified as OpFake or Boxer because sending a message is a common function of them.

We conducted an experiment to estimate the detection accuracy without using the SN blacklist to see the effectiveness of the blacklist. Without SN blacklist, the detection accuracy was 98%, which shows difference of only 1% in the result of the detection with the full SN blacklist. Analyzing time (except parsing and classification process) is increased from 400 seconds to 579 seconds. Consequently, this result shows that the well-collected SN blacklist mainly boost the speed of malware detection (30.9% faster) with slightly enhanced accuracy.

6.3 Performance Evaluation

To demonstrate the performance of our system, AndroTracker was compared with other systems: Andro-profiler [14] and Crowdroid [11]. Two of them are classification systems using system call based on dynamic analysis. The experiment was conducted with samples used in the study of Andro-profiler. They used 709 malware and 350 benign applications. The performance was compared focusing on classification accuracy and speed. Table 2 shows the result. AndroTracker detects and classifies malware much better than Crowdroid, though it has slightly lower accuracy than Andro-profiler. Still, AndroTracker was implemented on larger dataset including more kinds of malware families, and it was carried out successfully with Andro-profiler's dataset at the same time. Also, we checked the processing time of AndroTracker and Andro-profiler. The testing environment was set as Intel(R) Xeon(R) X5660 with 4GB RAM and Windows 7 Enterprise K operating system. To run Andro-profiler, the system takes 55 seconds/MB to analyze malware excluding setting time of an emulator (an added overhead). Andro-

Tracker performed at a speed of 72 seconds/MB. Considering dynamic analysis needs time for booting emulator between testing each sample, AndroTracker spends reasonable time for analysis.

Table 2. Comparing Classification Accuracy with Andro-profiler and Crowdroid

Category		AndroTracker	Andro-profiler	Crowdroid
Malware (709)	AdWo(401)	**1.00**	1.00	0.54
	AirPush(60)	**0.98**	0.95	0.02
	Boxer(42)	**1.00**	1.00	0.43
	FakeBattScar(51)	**0.96**	1.00	0.18
	FakeNotify(59)	**1.00**	1.00	0.80
	GinMaster(96)	**0.99**	1.00	0.11
Benign app(350)		**0.88**	0.97	0.35
Average		**0.96**	0.99	0.35

7 Conclusion

In this paper, we proposed the AndroTracker, an Android malware detection and classification system based on static analysis by using serial number information from the certificate as a feature. AndroTracker mainly checks a serial number, checks suspicious behavior of SMS hiding, detects the malicious system commands in the code, and analyzes the suspicious permission requests. As a result, AndroTracker can detect malware with 99% of accuracy. AndroTracker's classifier module can classify the 20 kinds of malware families with 90% accuracy. Additionally, AndroTracker can help analysts to react efficiently from Android malware's threats by detecting and classifying with high accuracy in a reasonable time (72 seconds/MB).

In the future, we will enhance AndroTracker by adding dynamic analysis functionality to overcome the general drawback of static analysis based system.

Acknowledgements. This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1004) supervised by the NIPA (National IT Industry Promotion Agency). This research was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2014R1A1A1006228).

References

1. McAfee: McAfee Labs Threats Report, Fourth Quarter 2013. `http://www.mcafee.com/au/resources/reports/rp-quarterly-threat-q4-2013.pdf`
2. F-Secure: Mobile Threat Report H2 2013. `http://www.f-secure.com/static/doc/labs_global/Research/Threat_Report_H2_2013.pdf`

3. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. Proceedings of the 16th ACM conference on Computer and communications security, pp. 235–245. ACM, Chicago, Illinois, USA (2009)
4. Pearce, P., Felt, A.P., Nunez, G., Wagner, D.: AdDroid: privilege separation for applications and advertisers in Android. Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 71–72. ACM, Seoul, Korea (2012)
5. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 3–14. ACM, Chicago, Illinois, USA (2011)
6. Zheng, M., Sun, M., Lui, J.C.S.: DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 12th IEEE International Conference on, pp. 163–171. (2013)
7. Rastogi, V., Chen, Y., Enck, W.: AppsPlayground: automatic security analysis of smartphone applications. Proceedings of the third ACM conference on Data and application security and privacy, pp. 209–220. ACM, San Antonio, Texas, USA (2013)
8. Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., Hoffmann, J.: Mobile-sandbox: having a deeper look into android applications. Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1808–1815. ACM, Coimbra, Portugal (2013)
9. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for Android. Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15–26. ACM, Chicago, Illinois, USA (2011)
10. Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Using probabilistic generative models for ranking risks of Android apps. Proceedings of the 2012 ACM conference on Computer and communications security, pp. 241–252. ACM, Raleigh, North Carolina, USA (2012)
11. Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Android permissions: a perspective combining risks and benefits. Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 13–22. ACM, Newark, New Jersey, USA (2012)
12. Jang, J., Yun, J., Woo, J., Kim, H.K.: Andro-profiler: anti-malware system based on behavior profiling of mobile malware. Proceedings of the companion publication of the 23rd international conference on World wide web companion, pp. 737–738. International World Wide Web Conferences Steering Committee, Seoul, Korea (2014)
13. Zhou, Y., Jiang, X.: Dissecting Android Malware: Characterization and Evolution. In: Security and Privacy (SP), IEEE Symposium on, pp. 95–109 (2012)
14. Seo, S.H., Gupta, A., Mohamed Sallam, A., Bertino, E., Yim, K.: Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications 38, 43–53 (2014)
15. Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D.: PScout: analyzing the Android permission specification. Proceedings of the 2012 ACM conference on Computer and communications security, pp. 217-228. ACM, Raleigh, North Carolina, USA (2012)
16. Contagio Mobile, `http://contagiominidump.blogspot.kr/`
17. Virusshare, `http://virusshare.com/`
18. Malware.lu, `http://malware.lu/`
19. VirusTotal, `http://www.virustotal.com/`