Digital Investigation xxx (2018) 1–9

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

An in-depth analysis of Android malware using hybrid techniques

Abdullah Talha Kabakus [a,*], Ibrahim Alper Dogru [b]

[a] Duzce University, Faculty of Engineering, Department of Computer Engineering, 81620, Duzce, Turkey
[b] Gazi University, Faculty of Technology, Department of Computer Engineering, 06560, Ankara, Turkey

ARTICLE INFO

Article history:
Received 29 September 2017
Received in revised form
26 December 2017
Accepted 10 January 2018
Available online xxx

Keywords:
Android malware
Android
Play store
Mobile security
Malware analysis

ABSTRACT

Android malware is widespread despite the effort provided by Google in order to prevent it from the official application market, *Play Store*. Two techniques namely static and dynamic analysis are commonly used to detect malicious applications in Android ecosystem. Both of these techniques have their own advantages and disadvantages. In this paper, we propose a novel hybrid Android malware analysis approach namely *mad4a* which uses the advantages of both static and dynamic analysis techniques. The aim of this study is revealing some unknown characteristics of Android malware through the used various analysis techniques. As the result of static and dynamic analysis on the widely used Android application datasets, digital investigators are informed about some underestimated characteristics of Android malware.

Introduction

Smartphones have changed the life of people dramatically in the last decade thanks to the provided functionalities and mobility. Android leads the mobile operating system market by being used on over 2 billion monthly active devices (Burke, 2017; Popper, 2017). According to a recent report by *IDC*[1], Android dominates the global smartphone market with being used on 85% of smartphones in all around the world (IDC Smartphone OS Market Share, 2017). It is expected that Android's global market share is expected to rise to 90% in 2017 (Bosnjak, 2017). As a result of this popularity, the official application market, *Play Store*, is used to install 82 billion applications in 2016 (Burke, 2017). It is reported that *Play Store* is growing at three times the rate of *Apple*'s *App Store* which is the official application market of *iOS* and the biggest official mobile application market after *Play Store* (Lookout, 2011). As a result of this popularity, *Play Store* attracts the attention of malware developers (Delac et al., 2011; Portokalidis et al., 2010; Wu et al., 2012; Zhou et al., 2012). Android malware has grown by 580% between September 2011 and September 2012 (Protalinski, 2012). According to a recent report by *Check Point*[2], the Android malware app "*Judy*" may have reached as many as 36.5 million users (The Judy Malware Possibly the largest malware campaign found on Google Play, 2017). *McAfee Labs* report that there are around 2.5 million new Android malware samples exposed yearly (McAfee Labs Threats Predictions Report, 2016). Also, they report that total mobile malware grew 79% in the past four quarters to 16.7 million samples (McAfee Labs Threats Report June 2017, 2017). Despite that these reports demonstrate how serious the threat is, the lack of security awareness of Android digital investigators is reported by many researches (Enck et al., 2009; Kelley et al., 2012; King et al., 2011; Mylonas et al., 2013). According to a recent report, while only 17% of participants are interested in permissions while installing the applications, 42% of participants are even unaware of the permissions (Felt et al., 2012). Google uses *Bouncer* which is a service supposed to detect malicious applications which are available on *Play Store* by scanning every available application using dynamic analysis (Alzaylaee et al., 2017; Lockheimer, 2012). Alongside to the *Bouncer*, Google has announced *Google Play Protect* during the event *Google I/O 2017* (Android – Google Play Protect, 2017; Cunningham, 2017). *Google Play Protect* is an always-on service which is bundled with the *Play Store* app. *Google Play Protect* scans the applications automatically even after the installation to ensure the applications remain safe in terms of security. According to the official website of *Google Play Protect*, it is reported that 50 billion applications are scanned by *Google Play Protect* daily (Android – Google Play Protect, 2017). An advantage of *Google Play Protect* over *Bouncer* is that *Google Play Protect* is able to scan applications which are not

* Corresponding author.
E-mail address: talhakabakus@gmail.com (A.T. Kabakus).
[1] http://idc.com.
[2] https://checkpoint.com.

https://doi.org/10.1016/j.diin.2018.01.001
1742-2876/

installed from *Play Store*. To the best of our knowledge, this paper is the first academic paper which introduces the *Google Play Protect*.

Android malware detection systems are generally categorized into two: (1) Static analysis, and (2) dynamic analysis. Both of them have own advantages and disadvantages as it is discussed in Section 3. To combine the advantages of each analysis technique, we propose a hybrid Android malware analysis framework namely *mad4a* which stands for “Malicious Application Detector for Android”. The main objective of this study is revealing the characteristics of Android applications through the proposed framework named *mad4a* which combines static and dynamic analyzing techniques in order to detect malware in Android. We investigate a large variety of Android applications in order to make a conclusion about the characterization and behavior of Android applications. The rest of the paper is structured as follows: Section 2 presents the related work. Section 3 discusses the proposed framework in detail. Section 4 discusses the findings and the result. Finally, Section 5 concludes the paper with future directions.

Related work

The related work can be classified through the technique it uses as follows: (1) Static analysis techniques, and (2) dynamic analysis techniques.

Static analysis

Feizollah et al. (Feizollah et al., 2017). propose an analysis of the effectiveness of intents for identifying malicious applications. They report that intents are a more valuable feature than permissions in terms of detecting Android malware. According to their evaluation, on an average, while an infected application declares 1.18 intent-filters, a benign application declares 1.61 intent-filters. Their proposed approach performs analysis on the smartphones. Due to the lack of both computation and storage resources, and power, *mad4a* is intentionally designed to perform analysis on a remote server. *RiskRanker* (Grace et al., 2012) is a scalable framework which utilizes various static analysis techniques such as the evaluation of program control flow graph and bytecode signatures. *Stowaway* (Felt et al., 2011a) detects the overprivilege by determining the set of API (Application Programming Interface) calls that an application uses which are mapped to the related permissions. They have evaluated *Stowaway* using a set of 940 applications and have found that about one-third of these applications are overprivileged. *Dendroid* (Suarez-Tangil et al., 2014) uses a text mining approach in order to analyze the code chunks in Android malware families. A high-level representation of the Control Flow Graph (CFG) is extracted using the detected code chunks instead of focusing on the specific sequence of instruction in the code chunks. The samples are classified into Android malware families by adopting the standard Vector Space Model and measuring the similarity between malware samples. *Peng* et al. (Peng et al., 2012). propose a static analysis approach solely based on permissions. They discuss the importance of effectively communicating the risk of an application to digital investigators. Also, they propose to use probabilistic generative model for risk scoring which they introduce. Schmidt (Schmidt, 2011) proposes a static analysis approach which uses the amount of free RAM (Random Access Memory), user inactivity in the last 10 s, the number of running processes, the percentage of CPU (Central Processing Unit) usage, and the number of SMS (Short Message Service) messages sent. *Nauman* et al. (Nauman et al., 2010). propose *Apex*, a policy enforcement framework for Android that allows a user to selectively grant permissions to applications as well as impose constraints on the usage of resources. *Apex* enables dynamic permission revocation which is also enabled with the release of Android 6.0 (API Level 23). *Kirin* (Enck et al., 2009) is a static analysis tool which evaluates application's permissions to perform lightweight certification to mitigate malware at installation time. *APK Auditor* (Kabakus et al., 2015) is a permission-based Android malware detection system which consists of three components namely (1) a central server, (2) a signature database, and (3) the Android client to interact with the server to scan applications for threats. *APK Auditor* calculates a malware score based on the requested permissions and then calculates the malware threshold limit dynamically using logistic regression. Finally, *APK Auditor* classifies the application as malicious if the calculated application malware score exceeds the malware threshold limit.

Dynamic analysis

Mahmood et al. (Mahmood et al., 2012). present an approach that utilizes *Robitium* test automation in order to test Android applications automatically in the cloud. The biggest limitation of using *Robotium* framework is that it requires the tested application to be signed in debug mode which is rarely used with the production-ready applications (Bierma et al., 2014). Even though applications which are not signed in debug mode can be resigned, this approach prevents these resigned applications to be distributed in Play Store. Unlike that work, *mad4a* does not have a limitation like that. *AppsPlayground* (Rastogi et al., 2013) is an automated dynamic analysis tool for Android applications. *AppsPlayground* uses permissions, and API calls. *MADAM (a Multi-level Anomaly Detector for Android Malware)* (Dini et al., 2012) is a dynamic analysis tool which concurrently monitors Android at both kernel and user levels in order to detect malware infections. *MADAM* exploits machine learning techniques to distinguish between benign and malicious behaviors. The features *MADAM* uses for the kernel-level analysis are system calls, running processes, memory and CPU usage. The user-level features *MADAM* uses are user-state, keystrokes, called numbers, sent or received SMS, and Bluetooth/Wi-Fi analysis. While monitoring and analysis processes of *MADAM* are performed on the local device, *mad4a* is specifically designed to perform the analysis on a remote server considering the limited resources (e.g., memory, CPU, disk space, battery) of smartphones. *Crowdroid* (Burguera et al., 2011) is a behavior-based dynamic analysis tool which monitors and analyzes system calls per application. Some dynamic analysis approaches (Buennemeyer et al., 2008; Jacoby and Davis, 2004; Kim et al., 2008) use the power consumption as the main malware detection feature for their analysis. Those approaches may be useful for the attacks which target power consumption but it is not sufficient since there are lots of different malware types (Alzaylaee et al., 2017). *mad4a* uses both static and dynamic features in order to cover as many malware types as possible. *TaintDroid* (Enck et al., 2010) is a system-wide information flow tracking tool that can simultaneously track multiple sources of sensitive data such as variables, methods, file, and messages throughout the program execution. According to their evaluation of 30 random and popular applications which are selected from *Play Store*, 15 applications have reported the location of users' to a remote advertising server. *Paranoid Android* (Portokalidis et al., 2010) transfers the recorded execution trace which is recorded on the smartphone to the cloud server over an encrypted channel. The cloud server replays the execution trace within the emulator. *Paranoid Android* uses a network proxy to connect to the Internet in order to intercept inbound traffic. Instead of using a proxy, *mad4a* accesses the network log file related to the simulated application which is located on the device.

ARTICLE IN PRESS

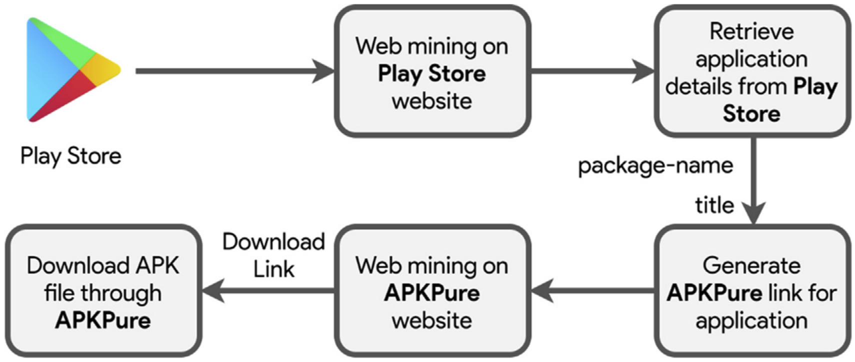

Fig. 1. The process of fetching applications from *Play Store*.

Material and method

Fetching applications from Play Store

The benign applications are fetched from *Play Store* using a third-party website named *APKPure*[3] which provides a web page for the applications available on *Play Store* with a link to download the related application in the following URL (Uniform Resource Locator) format: "https://apkpure.com/app_title/package_name". The benign applications are fetched from the various top charts such as "Top Grossing Games", "Top Selling Games", "Music and Audio", and "Weather" which are available on *Play Store*. The topics of the applications which are fetched from *Play Store* are specifically selected as a diverse range of topics in order to reflect the variety of the Android applications. Information related to the application such as title and package name are extracted from the related web page by using web mining techniques since *APKPure* does not provide an API (Application Programming Interface) to query and retrieve the data defined on its knowledge-base. Therefore, web mining techniques are used to parse the retrieved response from *APKPure*. The whole process of fetching applications from *Play Store* is presented in Fig. 1.

Static analysis

Static analysis techniques use the application's resources in order to investigate the application to categorize it as malicious or benign without executing the application (Chandramohan and Tan, 2012). Static analysis is helpless when the analyzed app is protected with advanced camouflage techniques (e.g., obfuscation) which remove, or limit access to the code (Moser et al., 2007), dynamic loading techniques (e.g., reflection), and encryption algorithms (Bae and Shin, 2017; Tam et al., 2017; Tong and Yan, 2017; Wang et al., 2017). An Android application archive (apk) file contains compiled source code (*classes.dex*), string and constant definitions, images, and the application manifest file *(AndroidManifest.xml)* which is used to define the metadata about the application such as requested permissions, unique package name, version, referenced libraries, and application components (e.g., activities) (Tam et al., 2017). Each apk file is firstly converted into a jar file using the *dex2jar*[4] tool. Then, the jar file is decompiled using the *jd-cli*[5] tool in order to retrieve the application's source code (Java files). *PScout II* (Wain et al., 2012) provides a list of methods defined in Android API which is mapped with the default permissions defined on Android 4.1.1 (API Level 16). *mad4a* uses a service which is implemented Java programming language is developed in order to find the API calls in the decompiled source code recursively and map them with the relative permission groups which are provided by *PScout II*. The list of permission groups with the related permissions is listed in Table 1.

Alongside the method mapping, *mad4a* extracts the permissions which are defined in *AndroidManifest.xml* for the static analysis using the *aapt* tool that is bundled with Android SDK (Software Development Kit). The whole process of the static analysis of *mad4a* is presented in Fig. 2.

Dynamic analysis

Dynamic analysis techniques monitor the application in real-time in an isolated environment, which is also known as a sandbox (Bläsing et al., 2010; Spreitzenbarth et al., 2013). These techniques use the artifacts generated by the application during this monitoring period. Dynamic analysis enables to uncover vulnerabilities that can only be detected at runtime (Bierma et al., 2014; Tong and Yan, 2017). Dynamic analysis evades the restrictions of static analysis such as obfuscation, dynamic loading (Gadhiya et al., 2013). The *monkey* tool which is provided by Android SDK lets generating pseudo-random streams of user events such as clicks, touches, and gestures (Azim and Neamtiu, 2013; Bläsing et al., 2010; Hu and Neamtiu, 2011; UI/Application Exerciser Monkey | Android Studio, 2017). *mad4a* uses *monkeyrunner* which is a tool based on the *monkey* in order to control the real device or emulator through the provided API (Machiry et al., 2013; monkeyrunner, 2017). Additionally, it is possible to write a script using Python programming language to execute batch commands.

mad4a generates a script during runtime after extracting the package name and the main activity information of the application

Table 1
The list of permission groups with the related permissions.

Permission Category	Related Permission(s)
Location	• android.permission.ACCESS_COARSE_LOCATION • android.permission.ACCESS_FINE_LOCATION
Call	• com.android.voicemail.permission.VOICEMAIL • android.permission.USE_SIP
Camera	• android.permission.CAMERA
Contacts	• android.permission.GET_ACCOUNTS • android.permission.WRITE_CONTACTS
Calendar	• android.permission.READ_CALENDAR
Telephony	• android.permission.READ_PHONE_STATE
Microphone	• android.permission.RECORD_AUDIO
SMS	• android.permission.SEND_SMS
Storage	• android.permission.WRITE_EXTERNAL_STORAGE

[3] https://apkpure.com.
[4] https://github.com/pxb1988/dex2jar.
[5] https://github.com/kwart/jd-cmd/tree/master/jd-cli.

Please cite this article in press as: Kabakus, A.T., Dogru, I.A., An in-depth analysis of Android malware using hybrid techniques, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.01.001

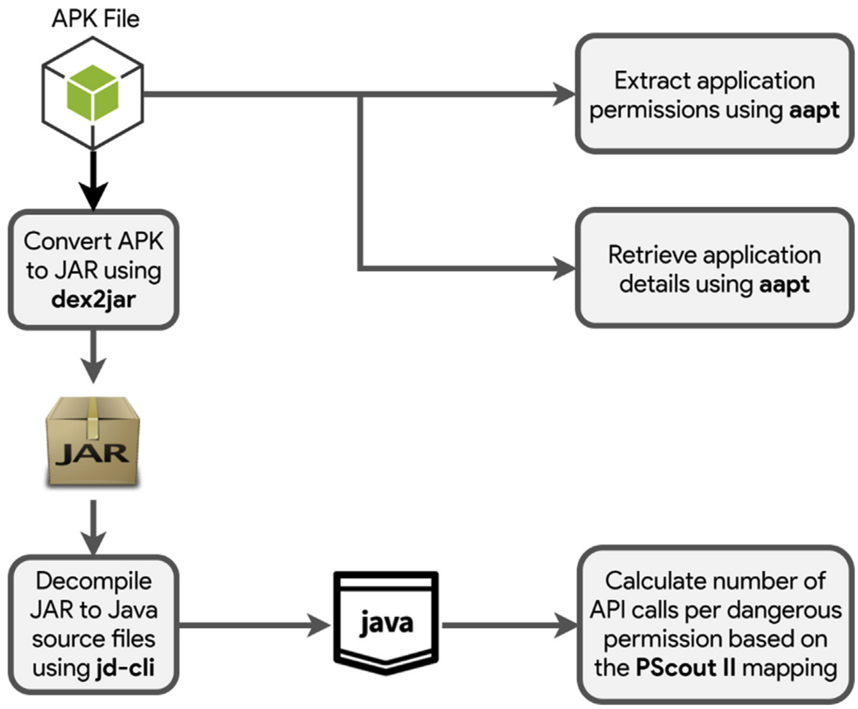

Fig. 2. The static analysis process of *mad4a*.

through the apk file. The generated script used to simulate each application in the dataset on the Android virtual device (emulator). Before running the application on the emulator, the mobile data connection and GPS (Global Positioning System) are enabled in order to reveal whether the application disables these settings or not. Then, the application is installed and run on the emulator. 500 random events are generated on the emulator in order to cover the application's functionality and generate related artifacts. After that, the mobile data connection and GPS are checked in order to reveal whether the simulated application disables these settings or not. Alongside these settings, the network usage of the simulated application in terms of the size of data downloaded or uploaded, and the number of incoming and outgoing connections through the local file located under*/proc/UID/net/xt_qtaguid/stats* are also monitored. The content of a sample application's network log file is presented in Fig. 3. From the available data, we only use *rx_bytes* and *tx_bytes* information which represent the downloaded bytes and uploaded bytes, respectively.

When the monitoring phase is finished, the related network log file is parsed and stored on a relational database management system. The whole process of the dynamic analysis of *mad4a* is presented in Fig. 4.

Result and discussion

The proposed approach is evaluated using the sample applications from various widely used datasets. While the benign applications are fetched from *Play Store*, the malicious ones are retrieved

```
idx iface acct_tag_hex uid_tag_int cnt_set rx_bytes rx_packets tx_bytes tx_packets
rx_tcp_bytes rx_tcp_packets rx_udp_bytes rx_udp_packets rx_other_bytes
rx_other_packets tx_tcp_bytes tx_tcp_packets tx_udp_bytes tx_udp_packets
tx_other_bytes tx_other_packets
2 wlan0 0x0 0 0 1668 18 2403 35 0 0 1004 10 664 8 0 0 1395 19 1008 16
3 wlan0 0x0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 wlan0 0x0 1000 0 1428 12 1902 15 1428 12 0 0 0 0 1902 15 0 0 0 0
5 wlan0 0x0 1000 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 wlan0 0x0 10011 0 40 1 40 1 40 1 0 0 0 0 40 1 0 0 0 0
7 wlan0 0x0 10011 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8 wlan0 0x0 10014 0 1153 13 1703 13 708 8 445 5 0 0 1258 8 445 5 0 0
9 wlan0 0x0 10014 1 356 4 356 4 0 0 356 4 0 0 0 0 356 4 0 0
10 wlan0 0x0 10029 0 6861 12 1116 13 6861 12 0 0 0 0 1116 13 0 0 0 0
11 wlan0 0x0 10029 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12 wlan0 0x0 10076 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig. 3. The content of a sample application's network log file content.

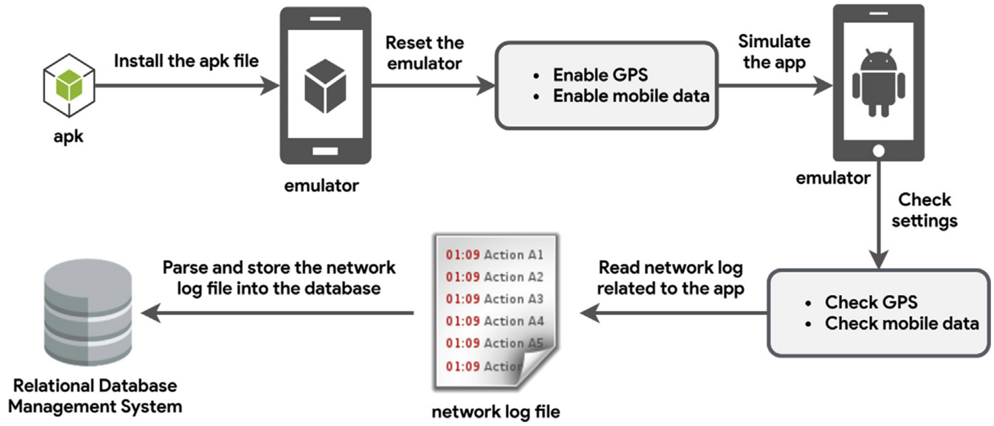

Fig. 4. The dynamic analysis process of *mad4a*.

from various datasets as the statistics about these datasets are listed in Tables 2 and 3.

According to the result of static analysis of *mad4a* which is listed in Table 4, benign applications tend to demand more permissions as well as make more API method calls compared to malicious applications. It is reasonable since the more API method calls mean the need of more permissions to be granted and the Android security mechanism highly relies on the permission model (Barrera et al., 2010; Felt et al., 2011b). In order to access a hardware, or personal information or do potentially dangerous actions such as sending SMS messages, making calls, taking photos, etc., applications need to define the related permissions in their manifest files and those permissions are needed to be granted by end users. Otherwise, when the application tries to complete an action that requires one of these permissions, the application crashes.

The Android security mechanism is mainly based on permissions. According to the static analysis of *mad4a*, the most requested 10 permissions by both malicious and benign applications are listed in Tables 5 and 6, respectively. *android.permission.INTERNET* is the most requested permission by both the malicious and benign applications which is the permission that is needed to be granted in order to connect the Internet. When we investigate the most used permissions, the permissions that are included in the list of malicious applications but not included in the list of most used permissions by benign applications are *android.permission.READ_SMS* and *android.permission.SEND_SMS* which are the permissions needed to be granted in order to read and send SMS, respectively.

Some malicious applications are reported to demand more permissions than they actually use which is also known as "overprivilege" (Kalutarage et al., 2012; Wang et al., 2014, 2013). Overprivilege is against the well-known principle "least-privilege" (Wei et al., 2012). In order to reveal the usage of overprivileged permissions for both malicious and benign applications, the applications in the *mad4a*'s database are analyzed. *mad4a* detects the overprivileged permissions as follows: If an application demands a permission, it should be found in somewhere in the decompiled source code. Otherwise, the application is accepted as overprivileged.

Table 2
The statistics about the dataset used by *mad4a*.

Dataset	Number of Applications
Play Store	2999
ASHISHB Malware[a]	58
Genome Project (Zhou and Jiang, 2012)	728
Drebin (Arp et al., 2014; Spreitzenbarth et al., 2013)	1953
Contagio Mobile[b]	70

[a] https://github.com/ashishb/android-malware
[b] http://contagiominidump.blogspot.com

Table 3
The distribution of malicious and benign applications in the dataset used by *mad4a*.

Application Category	Number of Applications
Malicious	2999
Benign	2809
Total	5808

Table 4
The result of the static analysis of *mad4a*.

Criteria	Malicious	Benign
Average number of API method calls	16	18
Average number of demanded permissions	7	8

Table 5
The most requested 10 permissions by malicious applications.

Permission	Number of Malicious Applications Used By	Percentage of Malicious Applications Use the Permission (%)
android.permission.INTERNET	1776	63
android.permission.READ_PHONE_STATE	1,650	59
android.permission.ACCESS_NETWORK_STATE	1484	53
android.permission.WRITE_EXTERNAL_STORAGE	1204	43
android.permission.RECEIVE_BOOT_COMPLETED	1067	38
android.permission.ACCESS_WIFI_STATE	1042	37
android.permission.READ_SMS	818	29
android.permission.WAKE_LOCK	755	27
android.permission.SEND_SMS	731	26
android.permission.VIBRATE	718	26

Please cite this article in press as: Kabakus, A.T., Dogru, I.A., An in-depth analysis of Android malware using hybrid techniques, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.01.001

Table 6
The most requested 10 permissions by benign applications.

Permission	Number of Benign Applications Used By	Percentage of Benign Applications Use the Permission (%)
android.permission.INTERNET	2975	99
android.permission.ACCESS_NETWORK_STATE	2937	98
android.permission.WRITE_EXTERNAL_STORAGE	2067	69
android.permission.WAKE_LOCK	1976	66
android.permission.ACCESS_WIFI_STATE	1463	49
android.permission.VIBRATE	1310	44
android.permission.READ_EXTERNAL_STORAGE	976	33
android.permission.READ_PHONE_STATE	915	31
android.permission.RECEIVE_BOOT_COMPLETED	861	29
android.permission.ACCESS_FINE_LOCATION	799	27

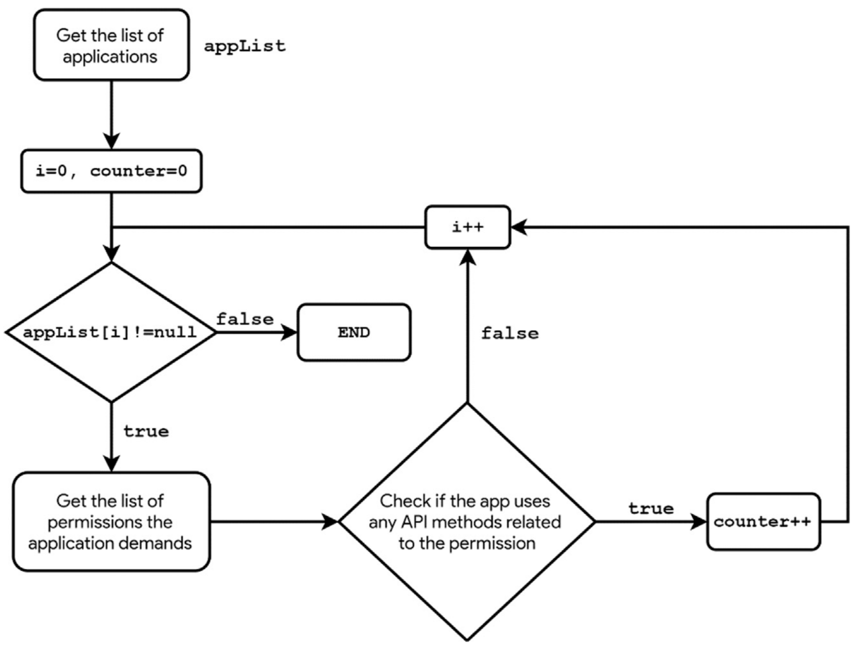

Fig. 5. The proposed algorithm to detect overprivileged permissions.

Table 7
The number of overprivileged permissions for both malicious and benign applications.

Application Category	Number of Overprivileged Permissions	Number of Applications	Average Number of Overprivileged Permission
Malicious	1532	2809	0.55
Benign	163	2999	0.05

Table 8
The most used top three permission categories according to API method calls for the malicious applications.

Permission Category	Number of Applications
android.permission.CONTACTS	2790
android.permission.CALENDAR	10
android.permission.LOCATION	3

Table 9
The most used top three permission categories according to API method calls for the benign applications.

Permission Category	Number of Applications
android.permission.CONTACTS	2994
android.permission. LOCATION	3
android.permission.CAMERA	2

The proposed algorithm to detect overprivileged permissions is presented in Fig. 5.

All the applications stored in *mad4a*'s database are analyzed in order to reveal the usage of overprivileged permissions for both malicious and benign applications. As the result is listed in Table 7, the average number of overprivileged permission is about eleven times fold common in malicious applications compared to the benign applications. The result validates the reports that mention

Table 10
The result of dynamic analysis of *mad4a*.

Criteria	Malicious	Benign
Average number of incoming and outgoing connections	87	233
Average size of download (MB)	15	671
Average size of upload (MB)	2	18
Average number of *INTERNET_CLOSE* action	519	464

Please cite this article in press as: Kabakus, A.T., Dogru, I.A., An in-depth analysis of Android malware using hybrid techniques, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.01.001

Table 11
The comparison of the malware detection techniques of *mad4a* to the related work.

Related Work	Analysis Technique	Malware Detection Method
(Feizollah et al., 2017)	Static analysis	Based on permission and intent-filters of the analyzed application
(Grace et al., 2012)	Static analysis	Based on various static analysis techniques such as evaluation of program control flow graph and bytecode signatures
(Felt et al., 2011a)	Static analysis	Based on determining the set of API (Application Programming Interface) calls that an application uses which are mapped to the related permissions
(Peng et al., 2012)	Static analysis	Based on analysis of permissions
(Schmidt, 2011)	Static analysis	Based on static analysis techniques such as amount of free RAM, number of running processes, percentage of CPU usage
Kirin (Enck et al., 2009)	Static analysis	Based on the evaluation of application's permissions to perform lightweight certification to mitigate malware at installation time
APK Auditor (Kabakus et al., 2015)	Static analysis	Based on calculation of application malware score (namely *AMS*) through analysis of permissions each application demands
(Mahmood et al., 2012)	Dynamic analysis	Based on the utilization of *Robitium* test automation in order to test Android applications automatically in the cloud
MADAM (Dini et al., 2012)	Dynamic analysis	Based on monitoring the operating system at both kernel and user levels
Crowdroid (Burguera et al., 2011)	Dynamic analysis	Based on monitoring and analyzing system calls per application
TaintDroid (Enck et al., 2010)	Dynamic analysis	Based on simultaneously tracking multiple sources of sensitive data such as variables, methods, file, and messages throughout the program execution
Paranoid Android (Portokalidis et al., 2010)	Dynamic analysis	Based on replaying the recorded the execution trace of each application over a network proxy that intercepts the inbound traffic
mad4a	Both static and dynamic analysis	Based on analyzing the permissions and network log of applications

the usage of overprivileged permissions is one of the characteristics of the malicious applications (Ali-Gombe, 2017; Felt et al., 2011a; Wei et al., 2012).

We introduce a new static analysis criterion for each analyzed app namely "*major category*". The major category of the application defines the most used category of API method calls which are decompiled from the application's compiled source file (*classes.dex*). The most used top three permission categories according to API method calls for both malicious and benign applications are listed in Tables 8 and 9, respectively. The result indicates that *android.permission.CALENDAR* is the one needs to be highlighted since being highly used by malicious applications.

629 malicious and 629 benign as total 1258 applications are simulated on the emulator. As the result is listed in Table 10, the benign applications tend to use the network more compared to malicious ones in terms of the number of incoming and outgoing connections, the download and upload size. However, disabling mobile network data is more common in the malicious applications compared to benign applications.

The comparison of the malware detection techniques of *mad4a* to the related work is listed in Table 11.

Conclusion

Smartphones are key targets of malware developers since they contain sensitive information about users such as contact lists which contain personal phone numbers, the details of user's bank accounts, the location of the user, the notes of the user, the calendar of the user, and the private chats of the user. According to the reports, Android is currently the most popular mobile operating system in the world. Android applications are distributed through the official application market namely *Play Store*. Despite that *Google* utilizes some security tools to detect the malicious applications which are available in *Play Store*, it is reported that the store still contains some malicious applications. Hence, a more comprehensive approach is necessary to detect more malicious application while not including the false negative samples. Therefore, in this paper, we propose a hybrid Android malware analysis approach namely *mad4a*. *mad4a* utilizes both static and dynamic analysis techniques in order to provide more comprehensive analysis and cover more malware detection approaches as many as possible. The widely used datasets which are publicly available are used to evaluate the proposed approach. The key contribution of the work is listed below:

- A hybrid approach is used to detect malicious applications instead of solely static or dynamic approach. The applications are monitored in an emulator which is configured for tests and monitored during these tests.
- According to the test result, malicious applications tend to disable the mobile data connection during their executions.
- The size of the data exchanged over the Internet is limited for malicious applications when it is compared to benign applications.
- Benign applications are more complex in terms of requested permissions in order to provide more functionalities compared to malicious applications which focus on its malicious actions.
- Since *android.permission.INTERNET*, *android.permission.ACCESS_NETWORK_STATE*, *android.permission.WRITE_EXTERNAL_STORAGE*, *android.permission.WAKE_LOCK*, *android.permission.ACCESS_WIFI_STATE*, *android.permission.VIBRATE*, *android.permission.READ_EXTERNAL_STORAGE*, *android.permission.READ_PHONE_STATE*, *android.permission.RECEIVE_BOOT_COMPLETED*, and *android.permission.ACCESS_FINE_LOCATION* are the permissions which are used by both the malicious and benign applications, we believe that these permissions cannot be solely used to detect malicious applications.
- Since the permissions *android.permission.READ_SMS* and *android.permission.WRITE_SMS* are only used by malicious applications, these permissions can be effectively used by the malware detection approaches based on permissions analysis.
- The permission *android.permission.CALENDAR* is commonly detected in the malicious applications' decompiled source code which can be used as a distinctive feature by the Android malware detection approaches based on source code analysis. To the best of our knowledge, no related work has indicated this information.
- The overprivileged permissions are more common (about eleven times fold) in malicious applications compared to benign applications. Therefore, this static analysis criterion can be efficiently used to classify Android applications as malicious or benign.
- Since we provide an entirely automated approach, it is possible to apply *mad4a* to bigger datasets.

The proposed approach can be extended by analyzing the API method call patterns in order to identify the reason behind the calls and the permissions these calls demand. As a future work, we would like to include the traces of API calls and explain the meaning of calls as a part of the static analysis.

Declaration of conflicting interests

The authors declare no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

Appendix A. Supplementary data

Supplementary data related to this article can be found at https://doi.org/10.1016/j.diin.2018.01.001.

References

Ali-Gombe, A.I., 2017. Malware Analysis and Privacy Policy Enforcement Techniques for Android Applications. University of New Orleans.

Alzaylaee, M.K., Yerima, S.Y., Sezer, S., 2017. Improving dynamic analysis of android apps using hybrid test input generation. In: IEEE Int. Conf. Cyber Secur. Prot. Digit. Serv. (Cyber Secur. 2017), London, UK.

Android – Google Play Protect, 2017. Google. https://www.android.com/play-protect/. (Accessed 21 August 2017).

Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., Drebin, Rieck K., 2014. Effective and Explainable Detection of Android Malware in Your Pocket. In: Symp. Netw. Distrib. Syst. Secur., San Diego, California, USA, pp. 23–26.

Azim, T., Neamtiu, I., 2013. Targeted and depth-first exploration for systematic testing of android apps. ACM SIGPLAN Not. 48, 641–660. https://doi.org/10.1145/2509136.2509549.

Bae, C., Shin, S., 2017. A collaborative approach on host and network level android malware detection. Secur. Commun. Network. 9, 5639–5650. https://doi.org/10.1002/sec.1723.

Barrera, D., Kayacik, H.G., van Oorschot, P.C., Somayaji, A., 2010. A Methodology for Empirical Analysis of Permission-based Security Models and its Application to Android. Proc. 17th ACM Conf. Comput. Commun. Secur. (CCS '10). ACM Press, Chicago, IL, USA, pp. 73–84. https://doi.org/10.1145/1866307.1866317.

Bierma, M., Gustafson, E., Erickson, J., Fritz, D., Choe, Y.R., 2014. Andlantis: large-scale android dynamic analysis. In: Proc. Third Work. Mob. Secur. Technol. 2014, San Jose, CA, USA.

Bläsing, T., Batyuk, L., Schmidt, A.D., Camtepe, S.A., Albayrak, S., 2010. An Android Application Sandbox System for Suspicious Software Detection. 5th IEEE Int. Conf. Malicious Unwanted Softw. (Malware 2010). IEEE, Nancy, France, pp. 55–62. https://doi.org/10.1109/MALWARE.2010.5665792.

Bosnjak, D., 2017. Report: Android's Global Market Share to Rise to 90% in 2017. AndroidHeadlines. https://www.androidheadlines.com/2017/04/report-androids-global-market-share-rise-90-2017.html. (Accessed 21 August 2017).

Buennemeyer, T.K., Nelson, T.M., Clagett, L.M., Dunning, J.P., Marchany, R.C., Tront, J.G., 2008. Mobile device profiling and intrusion detection using smart batteries. In: Proc. 41st Annu. Hawaii Int. Conf. Syst. Sci. (HICSS 2008), Waikoloa, HI, USA. https://doi.org/10.1109/HICSS.2008.319, 296–296.

Burguera, I., Zurutuza, U., Nadjm-Tehrani, S., 2011. Crowdroid: behavior-based malware detection system for Android. Science (80), 15–25. https://doi.org/10.1145/2046614.2046619.

Burke, D., 2017. Android: Celebrating a Big Milestone Together with You. Google. https://www.blog.google/products/android/2bn-milestone/. (Accessed 21 August 2017).

Chandramohan, M., Tan, H.B.K., 2012. Detection of mobile malware in the wild. Computer (Long Beach Calif) 45, 65–71. https://doi.org/10.1109/MC.2012.36.

Cunningham, E., 2017. Keeping You Safe with Google Play Protect. Google. https://blog.google/products/android/google-play-protect/. (Accessed 21 August 2017).

Delac, G., Silic, M., Krolo, J., 2011. Emerging security threats for mobile platforms. In: 2011 Proc. 34th Int. Conv. MIPRO, Opatija, Croatia, pp. 1468–1473.

Dini, G., Martinelli, F., Saracino, A., Sgandurra, D.M.A.D.A.M., 2012. A multi-level anomaly detector for android malware. In: Kotenko, I., Skormin, V. (Eds.), Comput. Netw. Secur, vol. 7531. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 240–253. https://doi.org/10.1007/978-3-642-33704-8.

Enck, W., Gilbert, P., Chun, B.-G., Cox, L.P., Jung, J., Mcdaniel, P., et al., 2010. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proc. 9th USENIX Conf. Oper. Syst. Des. Implement. (OSDI '10), Vancouver, BC, Canada, pp. 393–407.

Enck, W., Ongtang, M., McDaniel, P., 2009. On lightweight mobile phone application certification. In: Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS '09), Chicago, Illinois, USA, pp. 235–245. https://doi.org/10.1145/1653662.1653691.

Feizollah, A., Anuar, N.B., Salleh, R., Suarez-Tangil, G., Furnell, S., 2017. AndroDialysis: analysis of android intent effectiveness in malware detection. Comput. Secur. 65, 121–134. https://doi.org/10.1016/j.cose.2016.11.007.

Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D., 2011a. Android Permissions Demystified. Proc. 18th ACM Conf. Comput. Commun. Secur. – CCS '11. ACM Press, New York, New York, USA, p. 627. https://doi.org/10.1145/2046707.2046779.

Felt, A.P., Greenwood, K., Wagner, D., 2011b. The Effectiveness of Application Permissions. Proceeding WebApps'11 Proc. 2nd USENIX Conf. Web Appl. Dev. USENIX Association, Berkeley, CA, USA, p. 7.

Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D., 2012. Android permissions: user attention, comprehension, and behavior. In: Proc. Eighth Symp. Usable Priv. Secur. – SOUPS '12, Washington, DC, USA, p. 1. https://doi.org/10.1145/2335356.2335360.

Gadhiya, S., Bhavsar, K., Student, P.D., 2013. Techniques for malware analysis. Int. J. Adv. Res. Comput. Sci. Software Eng. 3, 972–975.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X., 2012. RiskRanker: Scalable and Accurate Zero-day Android Malware Detection. Proc. 10th Int. Conf. Mob. Syst. Appl. Serv. – MobiSys '12. ACM Press, Low Wood Bay, Lake District, United Kingdom, pp. 281–294. https://doi.org/10.1145/2307636.2307663.

Hu, C., Neamtiu, I., 2011. Automating GUI testing for android applications. In: 6th Int. Work. Autom. Softw. Test (AST 2011). ACM, Waikiki, Honolulu, HI, USA, pp. 77–83. https://doi.org/10.1145/1982595.1982612.

IDC Smartphone OS Market Share, 2017. IDC. http://www.idc.com/promo/smartphone-market-share/os. (Accessed 21 August 2017).

Jacoby, G.A., Davis, N.J., 2004. Battery-based intrusion detection. In: Glob. Telecommun. Conf. 2004 (GLOBECOM '04), vol. 4. IEEE, Dallas, Texas, USA, pp. 2250–2255. https://doi.org/10.1109/GLOCOM.2004.1378409.

Kabakus, A.T., Dogru, I.A., Cetin, A., 2015. APK Auditor: permission-based Android malware detection system. Digit. Invest. 13, 1–14. https://doi.org/10.1016/j.diin.2015.01.001.

Kalutarage, H.K., Krishnan, P., Shaikh, S.A., 2012. A certification process for android applications. In: 10th Int. Conf. Softw. Eng. Form. Methods (SEFM 2012), Thessaloniki, Greece, pp. 288–303. https://doi.org/10.1007/978-3-642-54338-8_24.

Kelley, P.G., Consolvo, S., Cranor, L.F., Jung, J., Sadeh, N., Wetherall, D., 2012. A conundrum of permissions: installing applications on an android smartphone. In: Blyth, J., Dietrich, S., Camp, L.J. (Eds.), Financ. Cryptogr. Data Secur, vol. 7398. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 68–79. https://doi.org/10.1007/978-3-642-34638-5.

Kim, H., Smith, J., Shin, K.G., 2008. Detecting Energy-greedy Anomalies and Mobile Malware Variants. Proceeding 6th Int. Conf. Mob. Syst. Appl. Serv. (MobiSys '08). ACM, Breckenridge, CO, USA, p. 239. https://doi.org/10.1145/1378600.1378627.

King, J., Lampinen, A., Smolen, A., 2011. Privacy: Is There an App for that? Proc. Seventh Symp. Usable Priv. Secur. – SOUPS '11. ACM Press, New York, NY, USA, p. 1. https://doi.org/10.1145/2078827.2078843.

Lockheimer, H., 2012. Android and Security. Google. http://googlemobile.blogspot.com/2012/02/android-and-security.html. (Accessed 21 August 2017).

Lookout, 2011. App Genome Report. https://www.lookout.com/resources/reports/appgenome. (Accessed 10 October 2014).

Machiry, A., Tahiliani, R., Naik, M., 2013. Dynodroid: an input generation system for android apps. In: Proc. 2013 9th Jt. Meet. Found. Softw. Eng. (ESEC/FSE 2013), Saint Petersburg, Russia, p. 224. https://doi.org/10.1145/2491411.2491450.

Mahmood, R., Esfahani, N., Kacem, T., Mirzaei, N., Malek, S., Stavrou, A., 2012. A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud. 7th Int. Work. Autom. Softw. Test (AST 2012). IEEE Press, Zurich, Switzerland, pp. 22–28. https://doi.org/10.1109/IWAST.2012.6228986.

Santa Clara, CA, USA McAfee Labs Threats Predictions Report, 2016.

Santa Clara, CA, USA McAfee Labs Threats Report June 2017, 2017.

monkeyrunner, 2017. Google. https://developer.android.com/studio/test/monkeyrunner/index.html. (Accessed 21 August 2017).

Moser, A., Kruegel, C., Kirda, E., 2007. Limits of static analysis for malware detection. In: 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC 2007), Miami Beach, FL, USA, pp. 421–430. https://doi.org/10.1109/ACSAC.2007.21.

Mylonas, A., Kastania, A., Gritzalis, D., 2013. Delegate the smartphone user? Security awareness in smartphone platforms. Comput. Secur. 34, 47–66. https://doi.org/10.1016/j.cose.2012.11.004.

Nauman, M., Khan, S., Zhang, X., 2010. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. Proc. 5th ACM Symp. Information, Comput. Commun. Secur. (ASIA CCS '10). ACM, Beijing, China, pp. 328–332. https://doi.org/10.1145/1755688.1755732.

Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., et al., 2012. Using probabilistic generative models for ranking risks of android apps. In: Proc. 2012 ACM Conf. Comput. Commun. Secur. (CCS '12), Raleigh, North Carolina, USA, pp. 241–252. https://doi.org/10.1145/2382196.2382224.

Popper, B., 2017. Google Announces over 2 Billion Monthly Active Devices on Android. The Verge. https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users. (Accessed 21 August 2017).

Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H., 2010. Paranoid android: versatile protection for smartphones. In: Annu. Comput. Secur. Appl. Conf., Austin, Texas, USA, pp. 347–356. https://doi.org/10.1145/1920261.1920313.

Protalinski, E., 2012. Android Malware up 580% Year-over-year. Next Web. https://thenextweb.com/google/2012/10/25/in-one-year-android-malware-up-580-23-of-the-top-500-on-google-play-deemed-high-risk/#.tnw_CnCxdv0x. (Accessed 21 August 2017).

Rastogi, V., Chen, Y., Enck, W., 2013. AppsPlayground : Automatic Security Analysis of Smartphone Applications. CODASPY '13 Proc. Third ACM Conf. Data Appl. Secur. Priv, pp. 209–220. https://doi.org/10.1145/2435349.2435379.

Schmidt, A.-D., 2011. Detection of Smartphone Malware. Technische Universität, Berlin.

Spreitzenbarth, M., Freiling, F.C., Echtler, F., Schreck, T., Hoffmann, J., 2013. Mobile-sandbox: Having a Deeper Look into Android Applications. Proc. 28th Annu. ACM Symp. Appl. Comput. (SAC 2013). ACM, Coimbra, Portugal, pp. 1808–1815. https://doi.org/10.1145/2480362.2480701.

Suarez-Tangil, G., Tapiador, J.E., Peris-Lopez, P., Blasco, J., 2014. Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families. Expert Syst. Appl. 41, 1104–1117. https://doi.org/10.1016/j.eswa.2013.07.106.

Tam, K., Feizollah, A., Anuar, N.B., Salleh, R., Cavallaro, L., 2017. The evolution of android malware and android analysis techniques. ACM Comput. Surv. 49, 1–41. https://doi.org/10.1145/3017427.

The Judy Malware Possibly the largest malware campaign found on Google Play, 2017. Check Point. https://blog.checkpoint.com/2017/05/25/judy-malware-possibly-largest-malware-campaign-found-google-play/. (Accessed 21 August 2017).

Tong, F., Yan, Z., 2017. A hybrid approach of mobile malware detection in Android. J. Parallel Distr. Comput. 103, 22–31. https://doi.org/10.1016/j.jpdc.2016.10.012.

UI/Application Exerciser Monkey | Android Studio, 2017. Google. https://developer.android.com/studio/test/monkey.html. (Accessed 21 August 2017).

Wain, K., Au, Y., Zhou, Y.F., Huang, Z., Lie, D., 2012. PScout: Analyzing the Android Permission Specification. CCS '12 Proc. 2012 ACM Conf. Comput. Commun. Secur. ACM, Raleigh, North Carolina, USA, pp. 217–228. https://doi.org/10.1145/2382196.2382222.

Wang, C., Li, Z., Mo, X., Yang, H., Zhao, Y., 2017. An android malware dynamic detection method based on service call co-occurrence matrices. Ann. Telecommun. 72, 1–9. https://doi.org/10.1007/s12243-017-0580-9.

Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X., 2014. Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. 9, 1869–1882. https://doi.org/10.1109/TIFS.2014.2353996.

Wang, Y., Zheng, J., Sun, C., Mukkamala, S., 2013. Quantitative security risk assessment of android permissions and applications. In: Wang, L., Shafiq, B. (Eds.), 27th Annu. IFIP WG 11.3. Springer, Newark, NJ, USA, pp. 226–241. https://doi.org/10.1007/978-3-642-39256-6_15.

Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M., 2012. Permission Evolution in the Android Ecosystem. ACSAC '12 Proc 28th Annu Comput Secur Appl Conf, pp. 31–40. https://doi.org/10.1145/2420950.2420956.

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., Wu, K.-P., 2012. DroidMat: android malware detection through manifest and api calls tracing. In: 2012 Seventh Asia Jt. Conf. Inf. Secur., Minato, Tokyo, Japan, pp. 62–69. https://doi.org/10.1109/AsiaJCIS.2012.18.

Zhou, Y., Jiang, X., 2012. Dissecting Android Malware: Characterization and Evolution. Proc. 33rd IEEE Symp. Secur. Priv. (Oakl. 2012), San Francisco, CA, USA. IEEE, pp. 95–109. https://doi.org/10.1109/SP.2012.16.

Zhou, Y., Wang, Z., Zhou, W., Jiang, X., 2012. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp., San Diego, California, USA.