Pervasive and Mobile Computing 32 (2016) 15–25

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

An integrated static detection and analysis framework for android

Jun Song [a], Chunling Han [a], Kaixin Wang [a], Jian Zhao [a], Rajiv Ranjan [b], Lizhe Wang [a,*]

[a] School of Computer Science, China University of Geosciences, Wuhan 430074, Hubei, PR China
[b] Newcastle University, UK

ARTICLE INFO

Article history:
Available online 24 March 2016

Keywords:
Android security
Malware detection
Static detection
Threat degree

ABSTRACT

The security and privacy issues of android system have attracted a lot of attention from both industry and academia in recent years. Static detection is one typical method to analyze malicious code. However, existing single static detection method can introduce high false alarm rate and is only appropriate for a limited scope. In this paper, we propose an integrated static detection framework, which consists of four layers of filtering mechanisms, that is, the message digest (MD5) values, the combination of malicious permissions, the dangerous permissions, and the dangerous intention, respectively. An intuitive threat-degree model is proposed especially on dangerous permissions detection. Furthermore, we implement a prototype system *ASE* and validate its feasibility, performance and scalability. A comprehensive evaluation shows that the proposed framework has obvious advantages especially in efficiency, granularity, layers, and correctness.

1. Introduction

With the rapid development of the wireless mobile technology, the emerging mobile Internet services are changing the original thinking pattern, behavior way, value concept, and the quality of human living. As an open source operating system, android has developed into one of the most widely used mobile Internet platform. In terms of an investigation report released by *Gartner* [1], the market share of android system has accounted for over 80% of the global smartphone and its aggregate sales have reached 1.2 billion in 2014, becoming the biggest growth point for profit in information technology field.

It is well known that the security and privacy issues of android system have attracted a lot of attention from both industry and academia so far. Android mobile devices are facing the growing serious threats caused by malware. A report [2] from a well known computer and network security provider *F-Secure* shows: 149 malware families have been discovered by 2014, and the number of android malware families is nearly 136 which accounts for approximately 91.3% of the total. Generally, these security and privacy issues mainly reflect in following aspects: disclosing user's contact, stealing user's personal material, silently tracking user's location, forcing users to install malware, deducting user's charges secretly, and so on. Therefore, the security threats from malware can produce a grave impact on promotion and development of android products. For the purpose of enhancing security, *Google* is stepping up efforts to improve safety measures on android

* Corresponding author.
E-mail address: Lizhe.Wang@computer.org (L. Wang).

http://dx.doi.org/10.1016/j.pmcj.2016.03.003
1574-1192/

platform. For example, as an efficient alternative, the compulsory authentication mechanism will be introduced when a legitimate user is ready to install an application from the *Google Play* [1].

Over the past few years, there have been quite a few studies on how to detect and prevent the malware on android [3–5]. Specially, the static detection is one of the most common methods to detect malicious code. However, the existing single static detection method can introduce high false alarm rate and is only appropriate for a limited scope. To address these concerns, in this paper, we focus on how to reduce the false alarm rate, improve the detection efficiency, and increase scalability of proposed solution. In order to achieve this goal, we first analyze and compare previous static detection methods. Inspired by these existing solutions, we propose an integrated static detection and analysis framework with four layers of filtering mechanisms. Furthermore, we implement a static detection prototype system named "ASE" and conduct a performance testing on real mobile devices.

The main contributions of this paper are threefold. First, we introduce an integrated static detection framework which provides four layers of filtering mechanisms, such as the MD5 characteristic values, the combination of malicious permissions, the dangerous permissions, and the dangerous intention, respectively. In this work, we perform static detection with emphasis on the `AndroidManifest.XML` file instead of on the `Smali` file. It is noted that, based on this improvement, we can reduce the workload of detection obviously and improve the detection efficiency practically. Second, we present a novel threat-degree threshold model of dangerous permissions on malware detection. Compared with existing detection methods, we use this model to assess and analyze dangerous degree of an application, which not only can improve the detection efficiency, but also can reduce the false alarm rate. Third, this proposed integrated framework achieves a reasonable trade-off between detection complexity and success rate. Owing to the properties of gradually deepening and comprehensive judgment, this proposed framework can reduce the workloads and lower false alarm rate to a certain extent, efficiently detect known malicious applications, and generate an instructive detection report on emerging unknown application as well.

Furthermore, we implement a prototype system *ASE* and conduct a comprehensive evaluation from the respects of feasibility, performance and scalability. Based on this prototype system, we detect 4006 real malware samples and their accuracy rates are nearly 99%. Additionally, utilizing a mobile device clouds, namely, *TestIn*, we perform a comprehensive testing on real mobile devices. In this testing, we use 83 real mobile devices and achieve a 98.80% pass rate, where the versions of android cover from 2.3 to 5.1, respectively. These evaluations show that the proposed framework provides the properties of better performance, high efficiency and low false alarm rate. Finally, this proposed framework is expected to not only detect and analyze the mobile malware on android with high efficiency but to provide a means of evaluating the dangerous degree of android applications at the minimized false alarm rate as well.

The rest of this paper is organized as follows: Section 2 overviews the related work and the property of static detection method. Section 3 describes malware classification and security mechanisms on android. Section 4 presents the detailed description of proposed static detection framework. The implementation, performance and further discussions are presented in Section 5, followed by conclusions in Section 6.

2. Background and related work

2.1. Related work

The studies of detecting and preventing malware on android platform are always hot topics in recent years [5–7]. [5] investigated 1260 malware samples which are from 49 different malware family of android, and further analyzed their characteristics of the behaviors and the load of installing and running applications. [8] discussed two main vulnerabilities of content provider in android components, i.e., passive content leaks and content pollution, which are caused by application developers owing to not setting the strict access permissions with the components.

With regard to the static detection methods, [9–12] proposed the rule-based scheme, respectively. [9] defined the rules of combined permissions for various potential malware. By means of the specific combination permissions and detection results, users can choose whether to continue or to abort the installation of unsafe applications. [10] proposed a permission-based footprint-matching method to detect the known malware family or unknown malware via a heuristic filtering technique. [11] presented a fuzzy hashing algorithm to detect the repackaged applications of the third-party application stores. [12] introduced a classification method to analyze an application based on the used permissions. This method took the permissions into consideration, whereas the characteristics of android applications was not concerned about. [13] implemented an android anomaly detection system which took the system data as a measure to conduct the anomaly detection. [14,15] proposed a behavior-based malware detection system *Andromaly*, providing the testing with *CPU* workloads, the number of running processes, the number of packets sent over WiFi, application startup, and other main characteristics. Besides that, [16] proposed a malware analytic method by parsing the `AndroidManifest` files and further developed a threat degree model according to the number of dangerous permissions and dangerous intention, providing the properties of low workload and high accuracy. [17] introduced a method based on permissions and application programming interface (API) calls to detect malware. Additionally, [18] proposed a malware detection system by monitoring the network traffic.

These previous studies reviewed the theoretical and technical aspects of the static detection methods. Through the combination and improvement of existing methods, this paper proposes an integrated static detection and analysis

Table 1
Comparison with Android-based static detection systems.

Reference	System	Detection method	Description
[13]		Signature-based	Analyzing and identify android applications dangerous function calls
[19]	AASandbox	Signature-based	Using android apps sandbox to implement a static and dynamic detection
[20]	TaintDroid	Anomaly	Using stain tracking technique to monitor users' privacy in real-time
[21]		Anomaly	Cloud security services through the malware detection and analysis
[22]		Anomaly	Evaluating the effect of using machine learning classifiers to detect malware
[9]	Kirin	Rule-based	Defined potentially dangerous combination of permissions rules to detect
[10]	DroidRanger	Permission-based	Using permission behavior footprint and heuristic filtering to detect
[23]	STREAM	Permission-based	Profiling applications to obtain information used in dynamic analysis
[24]	Crowedroid	Behavior-based	Using cloud information and *k*-means clusters algorithm to classify
Our	ASE	Integrated	Combining above four static detection mechanisms

framework, which shows obvious advantages especially in efficiency, granularity, layers, and correctness. As shown in Table 1, we simply compare our system *ASE* with other typical static detection systems for android.

2.2. Static detection

In practice, the malware detection systems on android generally utilize the combination of both static and dynamic detection techniques. It is noted that these two detection methods have their own advantages respectively. In this paper, we only review the properties and advantages of the static detection, which we focus on: (1) easy to conduct the comprehensive analysis. The static detection process is not bound by a specific program execution and is appropriate for all program execution. In contrast, the dynamic detection technique mainly concern the samples with the case of matching the predefined behaviors. (2) The certainty of detection results. Before proceeding to perform the detection, the results has been given and the malicious behavior is difficult to be disguised or falsified. (3) High detection efficiency. The static detection method is easy to estimate the computation cost and execution efficiency. In contrast, the dynamic detection method is liable to be affected by the deployment environment and its workload of detection are usually higher. On the whole, however, we believe that the dynamic detection method is a necessary complement to static detection, especially in providing a comprehensive evaluations with android malware.

3. Design background

3.1. Android malware classification

In this section, the malware classification and problem formulation are presented. There exist many various malware classification [3,5,9]. Generally, the existing malware can be grouped into eight categories. We now briefly introduce the common malware classification and their features that our later design will address:

- Malicious deductions: Without the knowledge or authorization, an application can lure the user to execute malicious code and result in the loss of charges secretly.
- Privacy theft: Without the knowledge or authorization, an application can access or steal user's information involving personal privacy and sensitive data secretly.
- Remote control: Without the knowledge or authorization, an application can silently receive and perform the remote commands in the background.
- Malicious code propagation: An application can secretly propagate itself or other malicious codes via the means of replication, infection, download, etc.
- Charges consumption: Without the knowledge or authorization, an application can automatically make calls, send SMS, download repeatedly or other means to cause extra charges for users.
- Systematic damage: The malicious code can damage system functions by various means, i.e., mobile terminal, system applications, users files, network services, etc.
- Trick fraud: The malware can trick a user to forge or tamper the normal applications so as to accomplish the improper purpose.
- Hooliganism: The malware has no direct damage to system or users. However, it can reside in memory, consume CPU resources, automatic binding, pop-up advertisements, and so on.

3.2. Android permission mechanism

Android system provides developers with over 100 kinds of permissions. For instance, when a developer needs to access the network via the developed applications, the `<use-permission>` item will be used to predeclare the permission in `AndroidManifest.xml` file. Table 2 shows some malicious collections of permissions with regard to users' privacy. In addition to the permissions provided by android system, a developer can also utilize the `<use-permission>` item to declare own permission and execute forcibly.

Table 2
Android permissions involving user privacy.

Categories	Permissions	Description
Location	android.permission.ACCESS_COARSE_LOCATION	Get a rough location
	android.permission.ACCESS_FINE_LOCATION	Get precise location
Network	android.permission.INTERNET	Access network
SMS	android.permission.READ_SMS	Read SMS content
	android.permission.RECEIVE_SMS	Receive SMS
	android.permission.SEND_SMS	Send SMS
	android.permission.WRITE_SMS	Edit SMS
Contacts	android.permission.READ_CONTACTS	Read contacts
	android.permission.WRITE_CONTACTS	Edit contacts
Record	android.permission.RECORD_AUDIO	Recording audio
Camera	android.permission.CAMERA	Taking pictures

3.3. Android signature mechanism

Generally, the `META-INF` folder of android application contains three files: `MANIFEST.MF`, `*.SF`, and `*.RSA`. The android application usually uses the self-signed mechanism to ensure application's origin and integrity to a certain extent. The signature utilizes asymmetric encryption algorithms and the concrete signing steps are as follows:

- (1) As for an android application, other files, except for the `META-INF` folder, all use the `SHA-1` algorithm generate message digests. All digests need to be encoded with `BASE64` and be written into the `MANIFEST.MF` file.
- (2) Based on the `MANIFEST.MF` file, all digests need to be encrypted by the private key of the certificate. These results will further be encoded with `BASE64` and be written into the `*.SF` file.
- (3) Save the public key into the `*.RSA` file.

As seen from the above steps, we can use `*.RSA` to decompress the `*.SF` file and verify its signature. Furthermore, we can validate the integrity of the `MANIFEST.MF` file through the `*.SF` file. Finally, we achieve the validation of the integrity of all files among the whole android application. It should be noted that the permission and signature mechanisms of android system are not in any way limited to these, we do not have them discussed in this work further.

4. Integrated static detection and analyses framework

In this section, we provide the details of proposed integrated static detection and analyses framework. The framework combines a four layers of filtering mechanisms, which is appropriate to filter known malicious applications layer by layer, and also can provide an detection report on unknown application activities. In this work, to improve the detection efficiency, we perform detection with emphasis on the `AndroidManifest.XML` file instead of on the `Smali` file. In addition, we present a novel threat-degree threshold model of dangerous permissions to detect malware.

4.1. Initialization

Generally, the workload of static detection depends mainly on the size and amount of the `Smali` files as well as the `AndroidManifest.XML` files in one application. However, there are a large number of `Smali` files in each application and they distribute in various directories of an application package (APK) file. Therefore, to traverse all files in an APK and to detect each `Smali` file, it will introduce a heavy workload and reduce the detection speed obviously. To solve this problem, we perform detection mainly on the `AndroidManifest.XML` file instead of on the `Smali` file in this work. Based on this improvement, this proposed framework can significantly reduce the workload and improve the detection efficiency on APKs.

4.2. Four layers of filtering mechanisms

As the above mentioned, our integrated static detection framework involves a four layers of filtering mechanisms: the MD5 characteristic values, the combination of malicious permissions, the dangerous permissions, and the dangerous intention. Under such a framework, any android application follows the detection flow shown in Fig. 1.

(1) *MD5 blacklist database*: The MD5 blacklist database is used to achieve the detection of the message digest characteristic values. In our *ASE* system, about 3000 of 4006 samples come from a security service providers and others are open sources. We first establishes a blacklist database of MD5 values by collecting 4006 malicious samples. Second, we can extract the MD5 values from the detecting APKs, and then perform both matching and judging quickly with the established blacklist. Finally, the system will filter out some known malicious applications, whereas the rest of the detected APKs will perform other detections. It is noted that the blacklist in our system mainly includes both MD5 values and the appropriate category of malicious applications, as shown in Table 3. The focus of this detection is

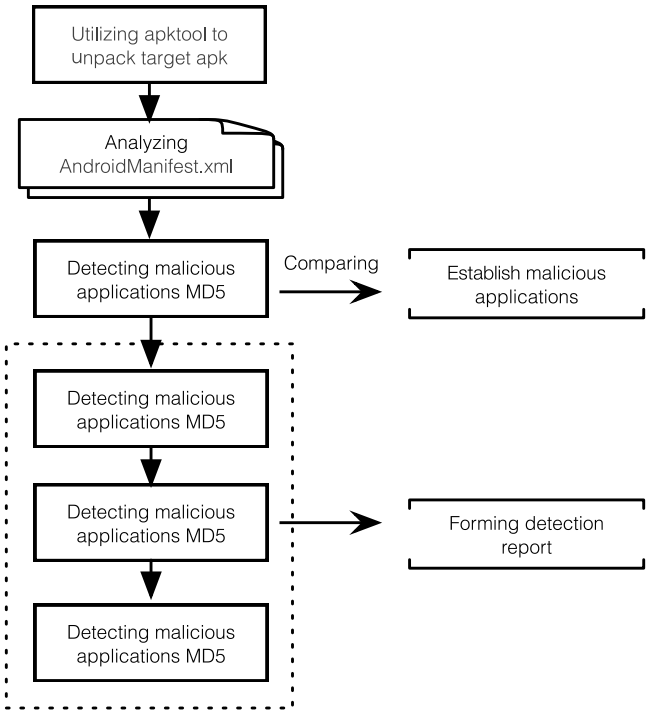


Fig. 1. Static detection flow.

Table 3
Illustration of MD5 fingerprint samples.

MD5 values	Description
a2bd62c99207956348986bf1357dea01	Android.Adware.AirAD.a
ac365eeb5595554d67975ad61003e48e	Android.Hack.i22hkt.a
f153a1ac9f8ee70f9e8db68ba62834de	Android.Troj.Kmin.A.v
ac365eeb5595554d67975ad61003e48e	Android.Hack.i22hkt.a
30f8c5d2cc445273e959b2a49fc8e937	Android.Troj.AirAD.a
540e8b5fdff054be1831cfbb4cdef7f0	Android.Troj.ACTCore.a
f14b9a0e83412e3156e79d716c96297d	Android.Troj.Blueja.a
20779151b36a15cb596e50b878f5f0bb	Android.Troj.GacBlocker.b

to extract the MD5 value and match it with the established MD5 blacklist. To achieve this detection, we can utilize the `MessageDigest.getInstance` function in `java.security.MessageDigest` package to extract the MD5 string of detected APK. During the matching process, the system needs to connect with the blacklist database and query the existing MD5 blacklist of malicious applications.

(2) *Combination of malicious permissions*: According to a previous work [5], a combination permissions including of ten types of malware families is proposed. In addition, some other combination permissions can also lead to malicious deduction and privacy leak problem, as shown in Table 4. Combined with the above combination permissions of malware families, we can obtain the appropriate APK permissions and match them with the existing combination permissions lists. Generally, if an application matches with all combined permissions of any malicious family, it means that there exists the high risks in this application, e.g., the malicious deduction or the leakage of privacy.

(3) *Dangerous permissions*: Based on the existing definition [17] and our analyzing results on 4006 malware samples, we can conclude 21 types of dangerous permissions on android malware, as shown in Table 5. In this work, we improve the detection method on dangerous permissions relative to [17]. In this detection, we first match the application permissions with the dangerous permissions, and then list the dangerous permissions that the application is applying for. Finally, we can generate a detection report and submit it to users.

Specially, we present an intuitive threat-degree threshold model especially for the detection of dangerous permissions. To achieve this goal, we define a threat-degree threshold, as shown in Eq. (1). The value δ denotes the threat-degree on dangerous permission of an application, α means the number of dangerous permissions that the current application is applying for, and β is the number of application permissions that the current application is applying for.

$$\delta = \frac{\alpha}{\beta}. \tag{1}$$

Table 4
Combination of malicious permissions.

Permissions combination		Description
android.permission.ACCESS_COARSE_LOCATION	android.permission.INTERNET	Uploading the rough location
android.permission.ACCESS_FINE_LOCATION	android.permission.INTERNET	Uploading the precise location
android.permission.READ_SMS	android.permission.INTERNET	Uploading the SMS message
android.permission.READ_CONTACTS	android.permission.INTERNET	Uploading the contact information
android.permission.SEND_SMS	android.permission.WRITE_SMS	Sending and receiving the paid SMS

Table 5
Dangerous permissions lists.

Dangerous permission	Dangerous intention	Dangerous permission	Dangerous intention
READ_SMS	Steal privacy information	CHANGE_WIFI_STATE	Access network
SEND_SMS	Malicious deductions	INTERNET	Access network
RECEIVE_SMS	Intercept communication	ACCESS_NETWORK_STATE	Access network
WRITE_SMS	Malicious deductions	RECORD_AUDIO	Steal privacy information
PROCESS_OUTGOING_CALL	Intercept communication	READ_PHONE_STATE	Intercept communication
MOUNT_UNMOUNT_FILESYSTEMS	Steal privacy information	READ_LOGS	Steal privacy information
READ_HISTORY_BOOKMARKS	Steal privacy information	CAMERA	Steal privacy information
RECEIVE_BOOT_COMPLETED	Self-starting	READ_CONTACTS	Steal privacy information
INSTALL_PACKAGES	Install malicious applications	ACCESS_FILE_LOCATION	Steal privacy information
MODIFY_PHONE_STATE	Intercept communication	CALL_PHONE	Malicious deductions
WRITE_HISTORY_BOOKMARKS	Malicious deductions		

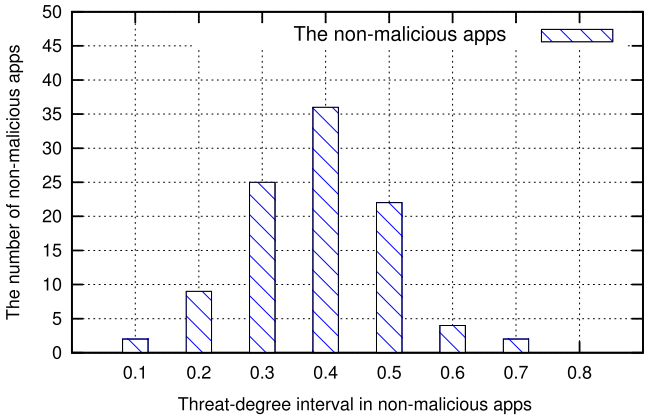


Fig. 2. Threat-degree distribution of non-malicious applications.

To validate the feasibility of the proposed threat-degree model, we perform a test on 1000 malicious samples and 100 non-malicious samples, respectively. It deserves noting that 1000 malicious samples come from a security service providers and other 100 non-malicious samples are from the top 100 applications in `Google Play` store. The specific detection results are shown in Figs. 2 and 3.

As seen from both Figs. 2 and 3, the threat-degree of malicious applications is significantly higher than non-malicious applications, especially for threat-degree exceeding 0.40. Overall, the threat-degree of malicious applications has a relatively common distribution above 0.40, whereas the threat-degree in most of non-malicious application is less than 0.50. In terms of the results of statistical analyses, it is found that the malicious applications whose threat-degree is above 0.50 is nearly 71.5%, whereas the non-malicious applications below 0.50 can reach 94%. Without loss of generality, we define 0.50 as the threshold value in this work. It deserves noting that this threshold value only means a low boundary as the judgment condition of malicious applications. In practice, we can utilize this threshold value to speed up the judgment on malicious applications to some extent. For example, when the threat-degree of a detected application is much greater than 0.50, e.g., 0.8 or greater, we can assume that it is a malicious application. In contrast, if the threat-degree of an application is much lower than 0.50, e.g., 0.3, it may be judged to be a benign application. On the other hand, owing to the uncertainty of dangerous permissions detection, we should judge an application more deeply so as to achieve a more accurate conclusion, i.e., by virtue of other detection methods and techniques.

With the respect of the number of detected samples, it should be noted that the size of 1000 malicious samples is 630 MB, whereas the size of 100 non-malicious samples reaches nearly 1.5 GB. In addition, we need not only to download non-malicious samples from `Google Play` store but also need to extract them into the mobile device. Thus the collecting of non-malicious samples has a high requirement for detection environment and storage space of real mobile devices. It is the

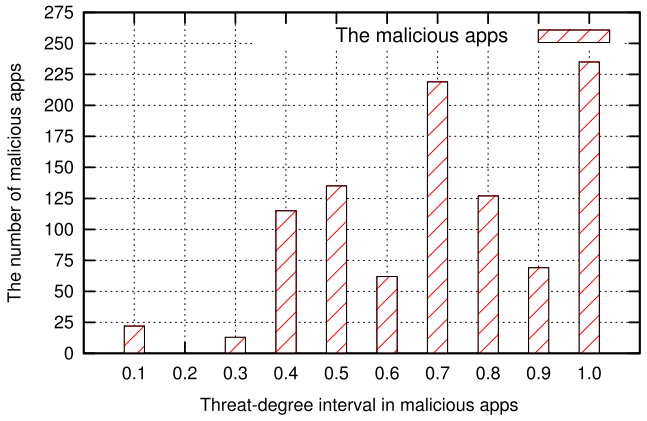


Fig. 3. Threat-degree distribution of malicious applications.

Table 6
Dangerous intention filter action list.

Dangerous permission	Dangerous intention	Dangerous permission	Dangerous intention
BOOT_COMPLETED	Self-starting, power consumption	INSTALL_SHORTCUT	Maliciously add shortcuts
SMS_RECEIVED	Intercept communication content	UNINSTALL_SHORTCUT	Maliciously delete shortcuts
CONNECTIVITY_CHANGE	Malicious deductions	NATIVE_CODE	Maliciously implant code
BATTERY_CHANGED	Maliciously change battery power	USER_PRESENT	Steal privacy information
PHONE_STATE	Intercept communication, deductions	PACKAGE_ADDED	Install malicious application
SCREEN_ON	Maliciously modify screen switch	SIG_STR	Steal phone signal strength
SCREEN_OFF	Maliciously modify screen switch	NEW_OUTGOING_CALL	Malicious deductions
TIME_TICK	Maliciously modify the system time		

Table 7
Comparison with known malware detection system.

System	Detection method	Detection object	0-day	Layers	Granularity	Working side	Usage
DriodRanger [10]	Based on permissions	Permission	Yes	2	Medium	Server	Higher
Crowdriod [19]	Based on behavior	Linux kernel calls	No	2	Coarser	Server	Medium
Kirin [9]	Based on rules	Permission	No	1	Coarser	Mobile	Lower
DroidMOSS [11]	Fuzzy hash	Hash	Yes	1	Coarser	Server	Medium
Our scheme	Permissions	Hash, Permission, Action	No	4	Finer	Mobile	Lower

main reason that only 100 non-malicious samples is used in our detection. But we believe that even more non-malicious samples in this test, the above results are also reasonable and feasible.

(4) *Dangerous intention*: In this work, we conduct a more in-depth analysis on `AndroidManifest.xml` files. That is, we statistically analyze the `action` strings of the dangerous `intent filters` as well. According to the definition of dangerous `action` in [16] and our analyzing on 4006 malicious samples, we summarize 15 types of dangerous `action` and briefly describe their dangerous intentions, as shown in Table 6.

The dangerous intent detection is a process to filter malicious applications based on the exiting dangerous intention. This detection may be repeated over and again depending on the requirements of the actual situation. Specially, we analyze and compare the performance of the proposed framework with four typical malware detection systems, including detection layer, detection method, resources usage and other five aspects. Here, the detection layer means the number of layers of malware detection system to analyze the features, the granularity refers to the size in which the detected content of android application are sub-divided, and the usage indicates that the detection system need to utilize the amount of the resource at work. Additionally, the 0-day means that whether or not the used system exists the 0-day vulnerability. All of them are divided into five grades in terms of their specific performance and requirements. As shown in Table 7, the result of this comparison show that the proposed framework has obvious advantages in detecting efficiency, granularity, layers, workload, performance and deployment.

5. System implementation and performance evaluation

In this section, we perform a comprehensive evaluation of the proposed framework and prototype system. By considering the implementation details, we conduct the component evaluation on implemented detection system. Then we apply the proposed integrated framework on our implemented prototype system and a mobile device clouds *TestIn* with realistic mobile devices settings, where we show the feasibility of the proposed framework on real mobile devices environment.

Table 8
Configuration of android AVD and real mobile devices.

Android AVD			Real mobile devices		
AVD	Version	RAM (MB)	Device	Version	RAM
MIUI2S_2.2	2.2-API level 8	512	HUAWEI_P6-C00	4.4.2	2 GB
MIUI2S_2.2	2.2-API level 8	512	HUAWEI_T8951d	4.1.1	512 MB
HUAWEI_13	3.2-API level 13	512	MOTOROLA_Droid2	2.3.4	512 MB
M9_40	4.1.2-API level 16	512	MB855	2.3.5	512 MB

Table 9
The number of samples and their success rate.

The number of samples			Success rate	
Commercial samples	Open source samples	Total	Detection method	Success rate
3000	1006	4006	MD5 detection	100%
3000	1006	4006	Combination permissions	100%
3000	1006	4006	Dangerous permissions	99%
3000	1006	4006	Dangerous intention	99%

Furthermore, there are three versions of android virtual devices (AVD), i.e., 2.2, 3.2, 4.1.2, and four types of the real phone devices, i.e., `Huawei T8951d`, `Huawei P6-C00`, `MOTOROLA Droid2 Global`, and `MOTOROLA MB855`, used in our mobile testing platform. Table 8 describes their main parameters of hardware configuration on android AVD and real mobile devices. Table 9 summarizes the number of actual samples and their success rate used in component evaluation of four layers on real mobile device.

5.1. Component evaluation

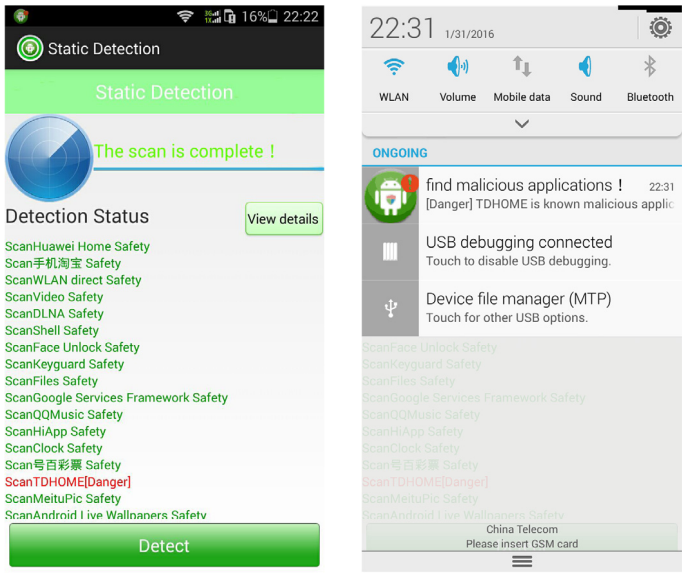
(1) *MD5 characteristic values detection*: We extract the MD5 values from the detected APKs, and then match them rapidly with the established MD5 blacklist database. Finally, the known malicious applications will be filtered out. According to our testing results, the success rate of detecting MD5 characteristic values on malicious applications is 100%. Here, we omit the illustrations of non-malicious and malicious applications that are recorded in the logs of our system.
(2) *Combination permissions detection*: After obtaining the pending permissions from APK applications, we can compare them with the each existing combination permissions. If an applications matches with all combined permission of any malicious family at the same time, we can assume that this is a suspected malicious application. With respect to detection results, this detection can accurately identify a suspected malicious application and the success rate can reach 100%.
(3) *Dangerous permissions detection*: We first classify and describe the known dangerous permissions, and then parse the permissions that the `AndroidManifest.xml` file of the application is applying for. Second, we match the parsed permissions with the existing dangerous permissions lists. Furthermore, we can calculate the threat degree of the detected application and generate a detection report for users. According to our testing and analysis results, this dangerous permissions detection can extract dangerous permissions of the applications efficiently and can record the discovered dangerous permissions into the detection report successfully. Its success rate can reach nearly 99%.
(4) *Dangerous intention detection*: Through a deep analysis on the `AndroidManifest.xml` files, we compare the performing `action` intention with the existing dangerous `action` lists and judge whether or not it is a dangerous `action` intention. With the respect of the detection results, we can find the dangerous intent based on this detecting method successfully. Additionally, approximately 99% of the success rate can be reached and it is relatively accurate as well.

5.2. Integrated evaluation on real mobile devices

Through the integrated static detect on the installed android applications, we can exclude malicious applications, generate the detection report of unknown applications, and submit it to the user. According to the detecting results, the user can decide whether to uninstall this application, to reinforce it, or to ignore it. In our prototype system, the detection process, the detection notification, the detection results with android applications, and the detection results with a safe application are shown in Fig. 4(a), Fig. 4(b), Fig. 5(c), and Fig. 5(d), respectively.

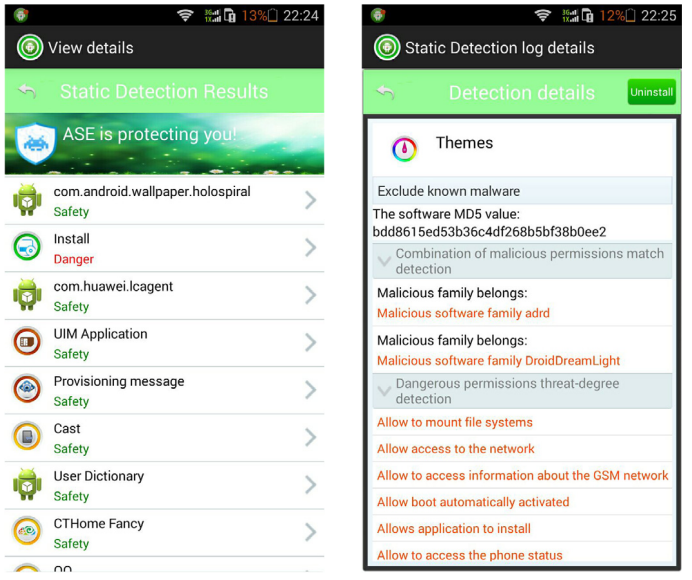
5.3. TestIn cloud evaluation

In order to evaluate the compatibility, scalability, and operating performance of our prototype system, we perform a comprehensive performance testing via the *TestIn* mobile device clouds environment. In this performance testing, there are a total of 83 types of real mobile devices, and android operating system covers eight versions from 2.3 to 5.1. The testing results show that the pass rate of our system is nearly 98.80% which covers over 61.56 million of mobile devices. The specific



(a) Detection process. (b) Detection notification.

Fig. 4. Performance evaluation on real mobile device.



(a) Detection results of android apps. (b) Detection results of a benign app.

Fig. 5. Performance evaluation on real mobile device.

performance results are shown in Table 10. In terms of the testing results, our system shows following obvious properties: a lower CPU and memory usage, a faster startup time, and little effect on the battery temperature when running. This indicated that our system provides better properties in both the compatibility and the operating performance.

5.4. The future work and discussion

For further improvements, there are some respects worth exploring: One possible concern is about the number of samples used in our testing. In our current system, there are 4006 malicious samples which are used in our experiment and analysis. However, it is not enough relative to the endless stream of new increasing malicious applications. We believe that it is necessary to further expand the sample database later. On the other hand, with regard to the threshold value of threat-degree, we use 1100 samples to conduct statistic analysis of threshold value. It is reasonable because this threshold value

Table 10
TestIn cloud test results.

Testing index	Test results
Installation time	Average installation time: 6.78 s
Start time	Average startup time: 1.89 s
CPU usage	Average CPU usage: 3.48%
Memory usage	Average memory usage: 36.72M
Traffic consumed	Average traffic consumption: 0.04 kB
Battery temperature	Average battery temperature: 38.01 °C
FPS	Average FPS: 17.08

only means a low boundary as the judgment condition on android malware. However, it is certain that, by the means of this threat-degree model, we can speed up judgment of malicious applications to some extent. Furthermore, the setting of the threshold value δ on threat-degree has a close relationship with the number and type of detection samples. And even in a certain extent, those samples will have an effect on the accuracy of threshold value. Compared with both Figs. 2 and 3, the threat-degree of malicious applications is obviously higher than non-malicious applications. It is apparently an important fact to support the reasonableness of proposed detection method. On the other hand, although the proposed system can detect most existing malicious applications with the proposed detection approaches, it does not integrate mechanisms resisting the latest malicious applications, e.g. polymorphic malware and newly appeared malware families. In addition, this proposed system is implemented on android system uniquely, without taking into consideration the deployment of PC-based environment. But we believe the efficient integration and implementation of these ideas is feasible and will be of our interest for the future work.

6. Conclusions

The studies of detecting and preventing android malware have attracted a lot of concerns in recent years. In this work, we propose an integrated static detection framework and implement a prototype system, which combines a four layers filtering mechanisms. The comprehensive evaluation on real mobile devices was conducted to show the efficiency, performance, and feasibility of the proposed framework.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 61303212, 61332019 and *U*1135004, and by Hubei Provincial Natural Science Foundation of China under Grant 2014*CFB*192.

References

[1] Gartner, Gartner says smartphone sales surpassed one billion units in 2014, 2014. http://www.gartner.com/newsroom/id/2996817.
[2] F-Secure, Mobile threat report q1 2014 by f-secure, 2014. http://www.fsecure.com/en/web/labs_global/whitepapers/reports.
[3] S. Cesare, Y. Xiang, W. Zhou, Malwise—an effective and efficient classification system for packed and polymorphic malware, IEEE Trans. Comput. 62 (6) (2013) 1193–1206.
[4] S. Cesare, Y. Xiang, W. Zhou, Control flow-based malware variant detection, IEEE Trans. Dependable Secure Comput. 11 (4) (2014) 307–317.
[5] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: IEEE Symposium on Security and Privacy, SP 2012, 21–23 May 2012, San Francisco, California, USA, 2012, pp. 95–109.
[6] Y. Wang, S. Wen, S. Cesare, W. Zhou, Y. Xiang, The microcosmic model of worm propagation, Comput. J. 54 (10) (2011) 1700–1720.
[7] W. Yu, H. Zhang, L. Ge, R.L. Hardy, On behavior-based detection of malware on android platform, in: 2013 IEEE Global Communications Conference, GLOBECOM 2013, Atlanta, GA, USA, December 9–13, 2013, 2013, pp. 814–819.
[8] Y. Zhou, X. Jiang, Detecting passive content leaks and pollution in android applications, in: 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24–27, 2013, 2013.
[9] W. Enck, M. Ongtang, P.D. McDaniel, On lightweight mobile phone application certification, in: Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9–13, 2009, 2009, pp. 235–245.
[10] Y. Zhou, Z. Wang, W. Zhou, X. Jiang, Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets, in: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5–8, 2012, 2012.
[11] W. Zhou, Y. Zhou, X. Jiang, P. Ning, Detecting repackaged smartphone applications in third-party android marketplaces, in: Second ACM Conference on Data and Application Security and Privacy, CODASPY 2012, San Antonio, TX, USA, February 7–9, 2012, 2012, pp. 317–326.
[12] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P.G. Bringas, G.Á Marañón, PUMA: permission usage to detect malware in android, in: International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions, Ostrava, Czech Republic, September 5th–7th, 2012, 2012, pp. 289–298.
[13] A. Schmidt, R. Bye, H. Schmidt, J.H. Clausen, O. Kiraz, K.A. Yüksel, S.A. Çamtepe, S. Albayrak, Static analysis of executables for collaborative malware detection on android, in: Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14–18 June 2009, 2009, pp. 1–5.
[14] A. Shabtai, Y. Elovici, Applying behavioral detection on android-based devices, in: Mobile Wireless Middleware, Operating Systems, and Applications—Third International Conference, Mobilware 2010, Chicago, IL, USA, June 30–July 2, 2010. Revised Selected Papers, 2010, pp. 235–249.
[15] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss, "Andromaly": a behavioral malware detection framework for android devices, J. Intell. Inf. Syst. 38 (1) (2012) 161–190.
[16] R. Sato, D. Chiba, S. Goto, Detecting android malware by analyzing manifest files, Proc. Asia-Pacific Adv. Netw. 36 (2013) 23–31.
[17] N. Peiravian, X. Zhu, Machine learning for android malware detection using permission and API calls, in: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4–6, 2013, 2013, pp. 300–305.
[18] J. Li, L. Zhai, X. Zhang, D. Quan, Research of android malware detection based on network traffic monitoring, in: Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on, 2014, pp. 1739–1744.

[19] T. Bläsing, L. Batyuk, A. Schmidt, S.A. Çamtepe, S. Albayrak, An android application sandbox system for suspicious software detection, in: 5th International Conference on Malicious and Unwanted Software, MALWARE 2010, Nancy, France, October 19–20, 2010, 2010, pp. 55–62.
[20] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L.P. Cox, J. Jung, P. McDaniel, A.N. Sheth, Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones, ACM Trans. Comput. Syst. 32 (2) (2014) 5:1–5:29.
[21] G. Portokalidis, P. Homburg, K. Anagnostakis, H. Bos, Paranoid android: versatile protection for smartphones, in: Twenty-Sixth Annual Computer Security Applications Conference, ACSAC 2010, Austin, Texas, USA, 6–10 December 2010, 2010, pp. 347–356.
[22] F.A. Narudin, A. Feizollah, N.B. Anuar, A. Gani, Evaluation of machine learning classifiers for mobile malware detection, Springer Soft Comput. (2014) 1–15.
[23] J.W. Brandon Amos, Hamilton A. Turner, Applying machine learning classifiers to dynamic android malware detection at scale, in: 2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013, Sardinia, Italy, July 1–5, 2013, 2013, pp. 1666–1671.
[24] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, Crowdroid: behavior-based malware detection system for android, in: SPSM'11, Proceedings of the 1st ACM Workshop Security and Privacy in Smartphones and Mobile Devices, Co-located with CCS 2011, October 17, 2011, Chicago, IL, USA, 2011, pp. 15–26.