International Journal of Security and Its Applications
Vol.9, No.8 (2015), pp.221-230
http://dx.doi.org/10.14257/ijsia.2015.9.8.19

An API Calls Monitoring-based Method for Effectively Detecting Malicious Repackaged Applications

Wenhao Fan[1, 2], Yuan'an Liu[1, 2] and Bihua Tang[1, 2]

[1]*School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*Beijing Key Laboratory of Work Safety Intelligent Monitoring, Beijing University of Posts and Telecommunications, Beijing 100876, China*
whfan@bupt.edu.cn

Abstract

The number of mobile applications on Android platform has largely increased in recent years. The security problems, as one of the outcomes induced by the popularity of Android, become more and more critical. Malicious repackaged applications (MRAs) perform malicious behaviors through malware developers embedding malicious codes into the benign origin applications (BOAs), which threat the security of users' finance and privacy. In this paper, a method based on API calls monitoring is proposed to effectively detect MRAs. We discuss the characteristics of behaviors and analyze the differences in API calls between MRAs and their BOAs. A MRA detection model is established, which builds up the super-sphere for each BOA via a SVDD algorithm. The model can detect the abnormal behaviors of MRAs which deviate the normal behaviors of corresponding BOAs. Experiments are carried out on imitated and real MRAs, where the results demonstrate the effectiveness of our method for detecting the singly and multiply contaminated BOAs.

Keywords: *Malicious repackaged applications detection; Android security; SVDD algorithm*

1. Introduction

Android is currently the most welcome smartphone operating system among manufacturers and users [13]. The popularity of Android also fertilizes the prosperity of smartphone applications. The number of applications in Google Play market [14]-[15] has reached over 1 million.

Inevitably, Android system also attracts a considerable number of malware authors due to its characteristic of openness, which allows users to install applications without security guarantee. The malware authors can download benign applications from the official Android Market or other third-party application stores, embed malicious codes into these applications, then submit the repackaged applications to public and induce users to download and install [1]. The MRAs (malicious repackaged applications) look exactly the same as BOAs (benign original applications) in appearance, but actually they quietly perform malicious acts in the background, such as absorption of communication cost by unauthorized dial, SMS and network access, steals of user's privacy, damage of data and system. Therefore, detection of MRAs becomes more and more urgent, and approaches to improve the accuracy and comprehensiveness of detection have significance both on theory and practice.

In this paper, a method based on API calls monitoring is proposed to effectively detect MRAs. The main idea is inspired by the fact that a MRA has different run-time behaviors from its corresponding BOA, since the extra components containing malicious codes are embedded in the BOA. Considering that applications having different behaviors will

ISSN: 1738-9976 IJSIA

perform different in API calls when they run in system, thus, there must be differences in API calls between MRA and its corresponding BOA. We establish a MRA detection model to obtain the law of BOA's behaviors, where a SVDD algorithm is employed to calculate the super-sphere of BOA in a multi-dimension space. Through the model, the behaviors of an application that deviate the normal behaviors of its BOA can be detected, and we can further judge whether the application is malicious repackaged or not.

The rest of this paper is organized as follows. Section II provides a brief introduction to related work on Android MRA detection. Section III details the MRA detection model we proposed. Section IV describes the experiment environments and results. Finally, the conclusion of our work is given in Section V.

2. Related Work

Malware detection is a hot area in information security technologies recently [7][12]. In this section, we mainly describe the related research on malware detection methods for smartphones.

DroidMoss [2] aims to find the malicious repackaged applications in third-party markets. It selects the hash values of authors' information and password as app-specific fingerprints, and then calculates the similarity between the original application and the application from a third-party market to judge whether one of the applications in the pair is repackaged. Although, DroidMoss fails to detect the malware which owns the same author's information as its benign application, and the applications without authors' information.

CrowDroid [3] uses the number of system calls as metrics to evaluate malicious applications. CrowDroid collects all kernel-level system calls of an application from a group of users who run the application in their phones. CrowDroid maintains two sets of data for the benign and malicious applications, then k-means clustering algorithm was applied to identify specific malicious applications by partitioning the system call collections of benign applications and their corresponding malwares into 2 clusters. CrowDroid mainly focuses on the framework which cooperates between users and the cloud, nevertheless, the detection is based on the assumptions that the data of benign applications is far greater than that of malicious ones, and the collections of benign applications are earlier than that of malicious ones. These assumptions restricts the practicality of CrowDroid.

SCSdroid [4] extracts the malicious common subsequences from the system call sequences of malwares belonging to the same family, which can be used to identify any evaluated application without requiring its original benign one. However, only unique malware can be detected, and SCSdroid cannot effectively detect the malware that does not belong to the same family.

As mentioned above, the approaches are proved to be valuable in protecting smartphones but they still have restrictions and drawbacks. In this paper, considering that applications which have different behaviors will perform differently in API calls, the proposed method is built up by training the MRA detection model with benign behaviors, once the training completes, the model can directly detect MRAs without any information of BOAs.

3. MRA Detection Model

Applications call multiple APIs provided by Android system to achieve their software functions. The MRA repackaged from a BOA is embedded malicious codes which involve the logic of user's privacy steals, communication cost consumptions, and other interests of adversaries, which result in different behaviors between the MRA and its corresponding BOA. When an application is running, there are differences in API calls

between the BOA and its corresponding MRA. Therefore, we can give a conclusion whether the application is malicious repackaged or not based on its API calls.

The architecture of our MRA detection model is shown in Fig. 1, which consists of 3 parts: API Monitoring Module, Database Module, and Data Analysis Module. API Monitoring Module is placed into the framework of Android system, by which relevant APIs are hooked. When an application is running, the module monitors and records the number of times that the APIs are called. Database Module stores the records of API Calls of BOAs, by which the features of BOAs of the monitored applications can be extracted. The module also maintains the data used by Data Analysis Module. Data Analysis Module uses the SVDD (Support Vector Data Description) algorithm to compute and describe the outlines of BOAs, which are used to distinguish whether a certain API call is abnormal or not. Detection results give the judgments that the monitored applications are malicious or benign according to the analysis of Data Analysis Module.

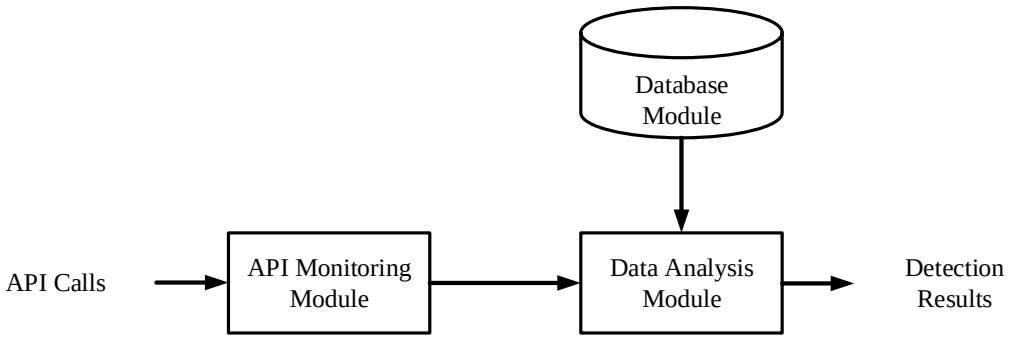

Figure 1. The Architecture of MRA Detection Model

3.1. Feature Extraction

The main work of feature extraction is to effectively extract API calls of the BOA when it runs in Android. The BOA and its corresponding MRA perform differently in API calls, for example, a BOA doesn't send SMSs, so the API sendTextMessage() will not appear in its record of API calls, while its corresponding malicious repackaged applications may send SMSs in the background without informing users, then it would call the API sendTextMessage() to implement its malicious behavior. Therefore, in order to discover the differences between BOA and MRA, effective feature selection and extraction is very important, so we need to obtain an accurate outline of the normal behavior of the BOA.

According to the analysis of MRAs, we discover that their malicious behaviors mainly include: generating communication cost, leaking privacy, getting location information, inserting advertisements, damaging system, etc. Based on above, we analyze several MRAs, especially focus on their API calls, and sum up the APIs commonly involved by MRAs. Table I shows the APIs that are summarized and adopted by us, which are divided into four categories: Cost, Privacy, Location and Network.

SVDD [5][6] is a classification model based on support vector machine theory, through which the super-sphere of normal behaviors of a certain BOA can be built up. SVDD try to find a super-sphere in the multi-dimension feature space to encase all recorded normal behaviors of the BOA. The radius of the super-sphere is maintained as small as possible, whereas, the number of training samples included in the super-sphere is kept as many as possible. We use SVDD to build the malware detection classifier. Single-class support vector machine only needs positive class samples, which are the normal behaviors of the BOA in our model.

Table 1. APIs Commonly Involved by MRAs

Category	Threat	Related APIs
Cost	Sending SMS, dialing, and other behaviors that consumes communication cost.	sendTextMessage(), sendMultipartTextMessage(), ACTION_SEND, ACTION_SENDTO, createFromPdu(), onCallStateChanged(), getOriginatingAddress(), EXTRA_EMAIL, EXTRA_STREAM, FEATURE_ENABLE_MMS, EXTRA_SUBJECT, ACITON_CALL, TelephonyManager.listen(), SMS_RECEIVED, etc.
Privacy	Getting contacts, SMS records, call logs and other private user information, which result in user information leakage	query(CallLog.Calls.CONTENT_URI), query(Phone.CONTENT_URI,PHONES_PROJECTION), managedQuery(), query(Uri.parse("content://icc/adn")), getDeviceId(), getNetworkOperator(), getSubscriberId(), getCallState(), getSimCountryIso(), getSimOperator(), getSimSerialNumber(), getLine1Number(), etc.
Location	Leaking location information	getLastKnownLocation(), onLocationChanged(), requestLocationUpdates(), etc.
Network	Consuming network flow	POST, HttpPost, Socket(), getOutputStream(), etc.

Super-sphere, which is formed by a benign sample set comprising n objects $\{x_1, x_2, x_3, \ldots, x_n\}$, is described by a center c and a radius R. The search for the optimal super-sphere in the feature space can be expressed as:

$$\min L(R, c, \varepsilon) = R^2 + C\sum_{i=1}^{n}\varepsilon_i, \quad i = 1,2,\ldots,n \tag{1}$$

s. t.

$$\|x_i - c\|^2 \le R^2 + \varepsilon_i, \quad \varepsilon_i \ge 0, \quad i = 1,2,\ldots,n \tag{2}$$

According to the Wolf duality theory, (1) can be converted to:

$$\max L = \sum_{i=1}^{n}\alpha_i(x_i, x_i) - \sum_{i,j=1}^{n}\alpha_i\alpha_j(x_i, x_j) \tag{3}$$

s. t.

$$0 \le \alpha_i \le C, \quad \sum_{i=1}^{n}\alpha_i = 1 \tag{4}$$

Assuming z is a test sample, we apply (5) to judge whether z is normal or not:

$$f(z) = \|z - c\|^2 = (z, z) - 2\sum_{i=1}^{n}\alpha_i(z, x_i) + \sum_{i,j}^{n}\alpha_i\alpha_j(x_i, x_j) \tag{5}$$

If $f(z) \le R^2$, we consider z as a normal point, otherwise z is an abnormal point. The radius R of super-sphere can be calculated according to the equation as below:

$$R^2 = (x_k, x_k) - 2\sum_{i=1}^{n}\alpha_i(x_i, x_k) + \sum_{i,j}^{n}\alpha_i\alpha_j(x_i, x_j) \tag{6}$$

where $x_k \epsilon\, SV < C$.

3.2. Behavior Description

In this part, we will present how to describe the behaviors of an application. An application call APIs to perform some specific behaviors, for example, in order to complete the actions of obtaining the phone number and accessing to the address

book, the application needs to call the API query(CallLog.Calls.CONTENT_URI) and API getLine1Number().

Given an application API call sequences $\mathbf{V}$ and an example $\mathbf{V}'$ as follows:

$$\mathbf{V} = \{A_1, A_2, A_3, \dots, A_i, \dots, A_n\}$$
$$\mathbf{V}' = \{\text{getLine1Number}(), \dots, \text{getLine1Number}() \dots, \text{query}(\text{CallLog. Calls. CONTENT\_URI}), \dots, \text{getLine1Number}()\}$$

where A_i represents the ith API call in the API call sequence. The APIs can be set in $\mathbf{V}$ are those commonly involved by MRAs, which are shown in Table I. we can compute the number of times that each API is called from $\mathbf{V}$, the result of which is denote by $\mathbf{F}$. $\mathbf{F}$ and an example $\mathbf{F}'$ of $\mathbf{F}$ are given as follows:

$$\mathbf{F} = \{f_1, f_2, f_3, \dots, f_i, \dots, f_n\}$$
$$\mathbf{F}' = \{3, \dots, 1\}$$

$\mathbf{F}$ is the feature vector to build up the database of application API calls, where f_i means the number of times that No. i API is called. Assuming API getLine1Number() is the No. 1 API, and API query(CallLog.Calls.CONTENT_URI) is the No. n API, $\mathbf{F}'$ is an example of $\mathbf{F}$, where 3 means the application called API getLine1Number() for 3 times; 1 means the application called API query(CallLog.Calls.CONTENT_URI) once.

3.3. MRA Detection Process

As shown in Figure 1, the architecture of our MRA detection model consists of 3 parts. We monitor and extract the API calls of an application during its run time. SVDD is applied to analyze API call sequence according to the BOA feature database established previously, then the discrimination value $f(z)$ of the application to be detected is calculated. Based on which the application is categorized. The detailed process is shown as below:

(1) Collect the number of times that APIs are called by the BOA, and establish feature database;
(2) Apply SVDD algorithm to find a super-sphere whose radius is maintained as small as possible and the number of training samples included in the super-sphere is kept as many as possible, calculate the center c and the radius R, then build a single classifier;
(3) For a certain application, in order to judge whether it is repackaged or not, monitor this application by API Monitoring Module, and record the API call sequences $\mathbf{V}$ of this application. Then, compute the z through $\mathbf{F}$;
(4) Use data c and R obtained from step (2) to calculate the distance $f(z)$ between z in step (3) and the center of super-sphere, then compare $f(z)$ with R^2;
(5) If $f(z) \leq R^2$, then it means z is a normal point, so we consider the application is benign. Otherwise, if $f(z) > R^2$, then it means z is an abnormal point, and the application is considered as malicious repackaged with high possibilities.

4. Experiment Results and Analysis

Experiments are conducted to evaluate the proposed method. We build up the dataset for the experiments by create several demo applications. The mobile phone used in experiments is Nexus 4, which is equipped with Android 4.2.1 system, and enables mobile network.

4.1. Imitated Malicious Repackaged Applications

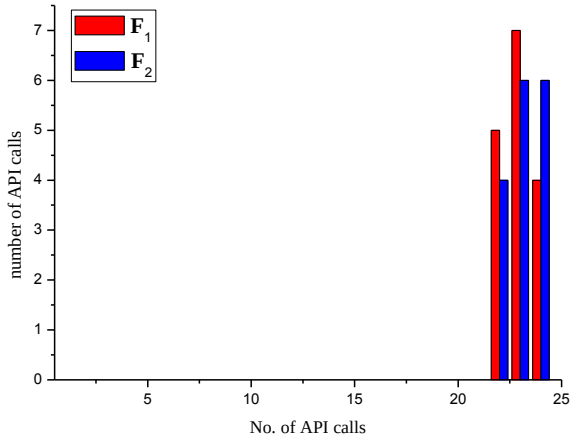

Figure 2. Results of F_1 and F_2

We write a food query application and collect the normal API call sequences of this BOA. We imitate the situation that the application is repackaged by inserting malicious codes into the BOA. The BOA mainly can search foods and beverages information around the location of users by connecting to the network. To achieve the functions of the applications, the BOA needs to call serval APIs, including POST, HttpPost and Socket(), etc. Fig.2 shows two examples of results of API calls $\mathbf{F}_1$ and $\mathbf{F}_2$ that are extracted when the BOA is running.

$$\mathbf{F}_1 = \{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 5\ 7\ 4\ 0\}$$
$$\mathbf{F}_2 = \{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 4\ 6\ 6\ 0\}$$

The classification of malicious behaviors is shown in Table I, we individually contaminate the BOA from 3 aspects: cost (represented by B1), privacy (represented by B2), location (represented by B3), as well as combinations of them. Table II shows the experiment results.

Table 2. Experiment Results for the Imitated MRA

No	Behavior	Benign Calls	Malicious Calls	Sample Data	Test Data	Accuracy
1	B1	12	3	10	5	100%
2	B2	12	3	10	5	100%
3	B1, B2	12	5	10	7	100%
4	B2, B3	12	5	10	7	100%

4.1.1. Single Contamination: In Table II, the data sets 1 and 2 are singly contaminated. Taking data set 1 as an example, in terms of cost, the BOA does not perform the malicious behaviors of generating communication cost, so there are no APIs related to cost are called during the process of the BOA. We embed the malicious behaviors of generating communication cost into the BOA, specifically enable the application to continuously send SMSs without the user's permission. The results reflecting in the API calls is that the application calls API sendTextMessage() to send text messages constantly. Figure 3 shows two examples of API calls $\mathbf{F}_3$ and $\mathbf{F}_4$ that are extracted from the singly contaminated MRA.

$$\mathbf{F}_3 = \{15\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 5\ 7\ 4\ 0\}$$
$$\mathbf{F}_4 = \{9\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 4\ 6\ 6\ 0\}$$

Comparing $\mathbf{F}_1$, $\mathbf{F}_2$, $\mathbf{F}_3$ and $\mathbf{F}_4$, we can observe that the times of API sendTextMessage() calls has an obvious distinction. After the application converting into a MRA, the frequency of callings of API sendTextMessage() largely increases. So, we can determine whether the application is malicious repackaged or not via our MRA detection model.

4.1.2. Multiple Contamination: In Table II, the data sets 3 and 4 are multiply contaminated. In the reality, one MRA always performs many kinds of malicious behaviors when it runs. Taking data set 3 as an example, the BOA does not have the malicious behaviors of generating communication cost and leaking user's privacy. We embed the malicious behaviors of generating communication cost and leaking user's privacy, specifically enable the application to continuously send SMSs which contain user's phone number. The results reflecting in the API calls is that the application calls both of API sendTextMessage() and API getLine1Number(). Figure 4 shows two examples of API calls $\mathbf{F}_5$ and $\mathbf{F}_6$ which are extracted from multiply contaminated MRA.

$$\mathbf{F}_5 = \{13\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 4\ 0\ 0\ 0\ 5\ 7\ 4\ 0\}$$
$$\mathbf{F}_6 = \{10\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 7\ 0\ 0\ 0\ 4\ 6\ 6\ 0\}$$

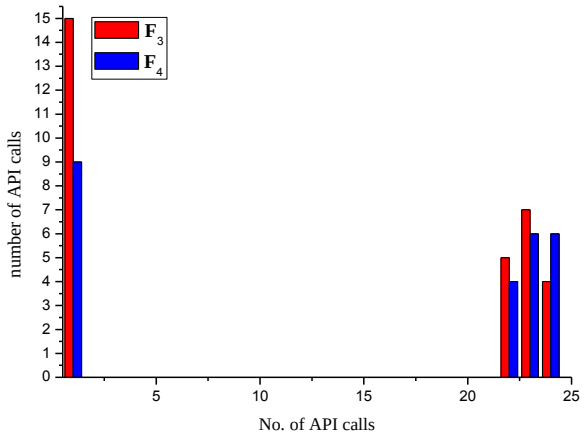

Figure 3. Results of $\mathbf{F}_3$ and $\mathbf{F}_4$

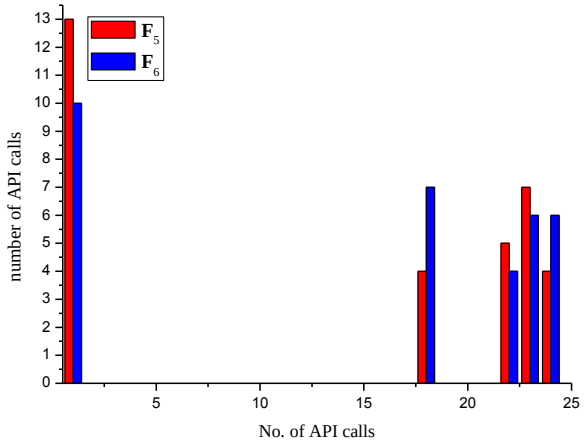

Figure 4. Results of $\mathbf{F}_5$ and $\mathbf{F}_6$

Comparing $\mathbf{F}_1$, $\mathbf{F}_2$, $\mathbf{F}_5$ and $\mathbf{F}_6$, we can find that the number of times recorded from API sendTextMessage () and API getLine1Number() has is different significantly. After the application turning out to be a MRA, the frequency of callings of API sendTextMessage() and getLine1Number() is higher than that of the BOA. So, whether the applicationlication is repackaged or not can be still determined by via our MRA detection model.

4.2. Real Malicious Repackaged Applications

We download the BOAs of two applications called SMS Multicaster and Contact Manager, and their corresponding MRAs. 45 API call sequences of the two applications are collected, respectively. Table III shows the results.

Table 3. Experiment Results for Real MRAs

Applications	Benign Calls	Malicious Calls	Sample Data	Test Data	Accuracy
SMS Multicaster	35	10	25	20	0.9
Contact Manager	35	10	25	20	0.95

The experiment results show that the detetion accuracies of the Monkey Jump 2 and Fingerprint Screensaver do not reach 100%, there are mainly two reasons. First, the malicious behaviors are not performed obviously during a few processes of the MRAs, so that the differences of API call sequences between the BOAs and MRAs may be very low, which causes failures of malware detection. Second, the number of training samples of the BOAs are not large enough, which results in that the API calls of the BOAs are not fully covered and test data of BOA is beyond the scope of the samples in the database module, so the BOAs may be detected as malware by mistake.

5. Conclusion

Maliciously repackaging benign applications is one of the tactics used by malware developers. The codes embedded perform malicious behaviors as the application runs, which severely challenge the security of users‘ finance and privacy. In this paper, we propose a API calls monitoring-based method for effectively detecting MRAs in Android system. The characteristics of behaviors and the differences in API calls between MRAs and their BOAs are analyzed and summerized. We establish a MRA detection model to detect the abnormal behaviors of MRAs. The model builds up the super-sphere for each BOA, which is trained by the records of normal behaviors of the BOA via a SVDD algorithm. Experiments are carried out on imitated and real MRAs, where the results demonstrate the high accuracy of our method for detecting the singly and multiply contaminated BOAs.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (Grant No. 61170275), Yang Fan Program of Guangdong Province, National Science and Technology Major Project (Grant No. 2012ZX03002012-002), Fundamental Research Funds for the Central Universities and Director Foundation of Beijing Key Laboratory of Work Safety Intelligent Monitoring.

References

[1] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution", Security and Privacy (SP), 2012 IEEE Symposium on, **(2012)** May; pp. 95-109.
[2] W. Zhou, Y. J. Zhou, X. X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party Android marketplaces", Proceedings of the 2nd ACM conference on data and application security and privacy, **(2012)** February; San Antonio, TX, USA, pp. 317-326.
[3] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android", Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices, **(2011)** October; Chicago, IL, USA, pp. 15-25.
[4] D. L. Ying, C. L. Yuan, H. C. Chien, and C. T. Hao, "Identifying android malicious repackaged applications by thread-grained system call sequences", Journal Computers and Security archive, vol. 39, **(2013)** November, pp. 340-350.
[5] D. Tax and R. Duin, "Data domain description using support vectors", Proceedings of European Symposium on Artificial Neural Networks, **(1999)** April 21-23; D-Facto, Brussels, pp. 251-257.
[6] A. Ben-Hur, D. Horn, H. Siegelmann and V. Vapnik, "A support vector clustering method", International Conference on Pattern Recognition, **(2000).**
[7] M. Grace, Y. Zhou, Q. Zhang, S. Zou and X. Jiang, "RiskRanker: Scalable and Accurate Zero-day Android Malware Detection", Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, **(2012)**; Low Wood Bay, Lake District, UK, pp. 281-294.
[8] D.-J. Wu, C.-H. Mao, Te-En Wei, H.-M. Lee and K.-P. Wu, "DroidMat: Android Malware Detection through Manifest and", API Calls Tracing, Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on, **(2012)** August; pp. 62-69.
[9] A. A. E. Elhadi, M. A. Maarof and A. H. Osman, "Malware detection based on hybrid signature behaviour application programming interface call graph", Am. J. Applied Sci., vol. 9, **(2012),** pp. 283-288.
[10] B. Anderson, D. Quist, J. Neil, C. Storlie and Lane, Terran. Graph-based malware detection using dynamic analysis, Journal in Computer Virology, vol. 7, no. 4, **(2011),** pp. 247-258.
[11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, "TaintDroid: An Information Flow Tracking System for Real-time Privacy Monitoring on Smartphones", Communications of the ACM, vol. 57, no. 3, **(2014),** pp. 99-106.
[12] S. Arzt, S. Rasthofer and C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps", Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, **(2014)**, pp. 259-269.
[13] Information on http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[14] Information on http://en.wikipedia.org/wiki/Google_Play
[15] Information on http://www.androidcentral.com/google-play-store

Author

Wenhao Fan, he received the B.E. and Ph.D. degree from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2008 and 2013, respectively. He is currently an assistant professor at the School of Electronic Engineering in BUPT.

His main research topics include information security for mobile terminals, mobile computing, parallel computing and transmission, and software engineering for mobile internet.