A Risk Classification Based Approach for Android Malware Detection

Yilin Ye[1], Lifa Wu[1], Zheng Hong[1], Kangyu Huang[1]
[1] Institute of Command Information System, PLA University of Science and Technology, No.1 Haifu Street, Nanjing, Jiangsu, China
[e-mail: my_etsi@163.com, wulifa@vip.163.com, hongzhengjs@139.com, huangkangyu@163.com]
*Corresponding author: Li-fa Wu

Received December 11, 2015; revised October 12, 2016; revised November 21, 2016; accepted December 23, 2016; published February 28, 2017

Abstract

Existing Android malware detection approaches mostly have concentrated on superficial features such as requested or used permissions, which can't reflect the essential differences between benign apps and malware. In this paper, we propose a quantitative calculation model of application risks based on the key observation that the essential differences between benign apps and malware actually lie in the way how permissions are used, or rather the way how their corresponding permission methods are used. Specifically, we employ a fine-grained analysis on Android application risks. We firstly classify application risks into five specific categories and then introduce comprehensive risk, which is computed based on the former five, to describe the overall risk of an application. Given that users' risk preference and risk-bearing ability are naturally fuzzy, we design and implement a fuzzy logic system to calculate the comprehensive risk. On the basis of the quantitative calculation model, we propose a risk classification based approach for Android malware detection. The experiments show that our approach can achieve high accuracy with a low false positive rate using the RandomForest algorithm.

Keywords: Android, malware detection, risk, machine learning, fuzzy logic

This work was supported by The Nature Science Foundation of Jiangsu China under Grant No.BK20131069. The authors would like to thank the anonyous reivewers for their valuable comments which improved the quality of this paper.

1. Introduction

Recently, A market report [1] released by IDC shows that Android holds about 82.8% of the mobile system market share during the last quarter. Due to its overwhelming market share and a great number of users, Android has become the most popular platform of malicious attackers. Driven by economical interest, there is a boost in the number of malware, spywares in Android mobile markets. As showed in [2], 3.26 million Android malware have been dectectd in 2014, compared with the number of Android malware a year ago, the number increased nearly 386%. While the number of affected users reached 319 million, it increased 517% compared with the number of the affected users last year.

Plenty of research has been made to alleviate the increasing threat to Android system brought by malware. One of the hot directions is Android permission analysis [3][4][5][6][7], where researchers have presented lots of work on Android system security enhancement [8][9][10][11][12][13][14] and malware detection [15][16][17]. According to the techniques employed, Android malware detection approaches can be classified into static analysis and dynamic analysis. Dynamic analysis captures what the sample has done during the evaluation, and relies on runtime behaviors to judge whether the sample is malicious or not. The down side of dynamic analysis is that it suffers from a big system overhead and inefficient path coverage, which causes false positives.

Reina et al. presented CopperDroid [18] to monitor the inner IPC and RPC communication of Android applications and capture runtime system calls to reconstruct application behaviors. VetDroid [19] concentrated on used permissions of an app, and only captured the system calls related to used permissions. Andrubis [20] applied static analysis to extract static features, and applied dynamic analysis to reconstruct runtime behaviors. Andrubis then combined static features and run-time behaviors to detect Android malware. There are other typical approaches based on dynamic analysis such as [21][22][23][24][25][26].

Compared with dynamic analysis, static analysis focuses on source code instead of runtime behavior. It is a lightweight technique that consumes relatively less system resources and achieves high path coverage, but it can be evaded by techniques such as obfuscation and dynamic loading.

Among all the schemes based on static analysis, permission-based schemes concentrate on requested permissions. Kirin [27] detected Android malware based on dangerous permission combinations or suspicious action strings. The approach presented in [28] extended Kirin by increasing the number of permissions to define more permission combinations. As there are few differences in requested or used permissions between benign apps and malware, permission-based approaches suffer the problem of low detection rate.

To overcome the shortcomings of permission-based approaches, multi-category feature based approaches [29][30][31][32][33] were proposed, which extracted many other static features besides permissions. DroidAPIMiner [34] conducted frequency analysis and data flow analysis to all APIs used in an app to acquire most frequently used APIs and their parameters. Drebin [35] conducted a broad static analysis to extract the application features containing permissions, application components, sensitive APIs, network addresses, strings and so on. Though more features are extracted to overcome the disadvantages of permission-based approaches, multi-category feature based approaches still suffered the problem that the features extracted can’t reflect the key differences between malicious apps and benign apps.

After analyzing over 30,000 samples, we found that the superficial features (e.g. requested or used permissions, sensitive APIs, filtered intents) of the two types of apps showed high similarities. Naturally, it will end in relatively high false positives and false negatives if we rely on superficial features to detect Android malware.

In this paper, we aim to overcome the shortcomings of current static analysis based methods. Based on the key observation that the essential differences between malicious apps and benign apps are rooted in the way they use the permissions granted by the users, we concentrate on how requested permissions are used in an app. As Android system maps application permissions to different permission methods in the Framework layer, we can employ permission methods to describe how the corresponding permissions are used, and we propose the notion of function call trace which represents an ordered sequence of methods corresponding to different nodes on an execution path of an app's function call graph.

Based on the above analysis, we have designed a quantitative calculation model of application risks, and proposed a novel Android malware detection approach on the basis of risk classification. Specifically, we classify application risks into five types of specific risk (money risk, privacy risk, network connection risk, highly dangerous permission risk and sensitive program behavior risk), and propose comprehensive risk to evaluate the overall risk of an Android application. The comprehensive risk is computed based on the former five. As risk preference and risk-bearing ability of the users are naturally fuzzy, we introduce fuzzy logic to calculate comprehensive risk.

In summary, the main contributions of our work are listed as follows.

- To overcome the disadvantages of existing static analysis based approaches that the used features are not effective to distinguish malware from benign Android applications, we choose the way how permissions are used as the basic distinguishing feature, and it is described by the function call traces corresponding to permission methods. We present a risk classification based approach for Android malware detection, which can achieve 93.2% detection accuracy.
- We employ a fine-grained analysis on Android application risks, and propose a quantitative calculation model of application risks.
- We implement a fuzzy logic system to compute the overall application risk, which is fuzzy due to the fact that users differ in risk preference and risk tolerance.

The remainder of this paper is organized as follows: Section 2 illustrates the motivation of this paper. Section 3 describes our method in details. After that, experiments and result analysis are presented in section 4. Related work and limitations are discussed in section 5 and section 6. Section 7 concludes the paper and proposes future work.

2. Motivation

After analyzing over 30,000 samples (discussed in section 4.1), we discovered that benign apps and malicious apps show high similarity among superficial features, such as requested and used permissions, broadcast events, sensitive program behaviors. **Fig. 1 (a)** and **Fig. 1 (b)** depict the comparison of frequency between requested and used permissions of benign apps and malicious apps respectively. As shown in both figures, the frequency of requested permissions are higher than used permissions, indicating that many Android applications are actually over-privileged, which would lower the accuracy of approaches based on requested permissions.

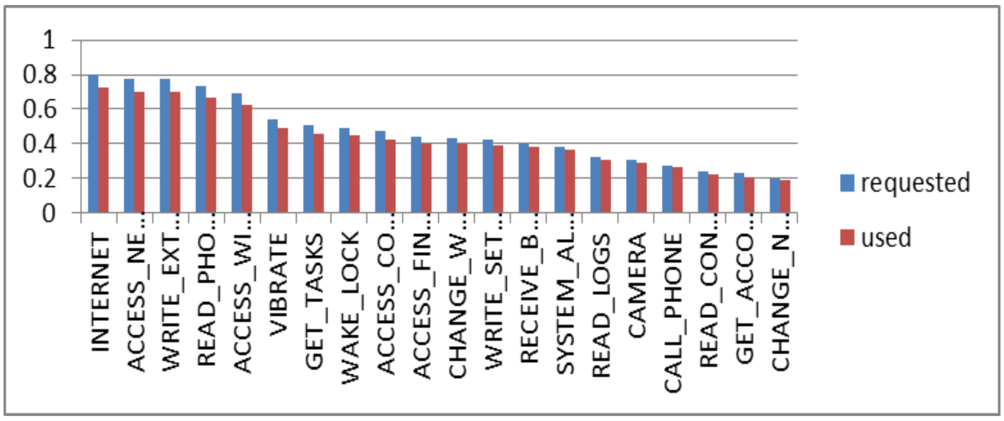


Fig. 1(a). comparison of requested and used frequency of the top 20 permissions of benign apps

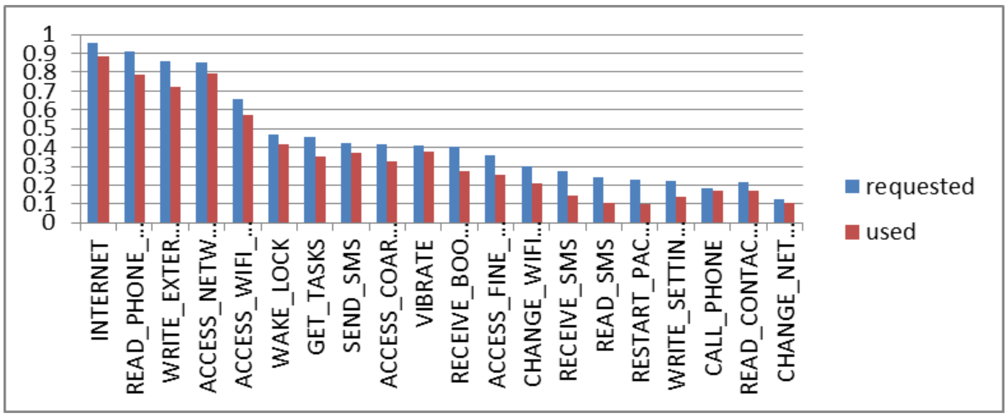


Fig. 1(b). comparison of requested and used frequency of the top 20 permissions of malware

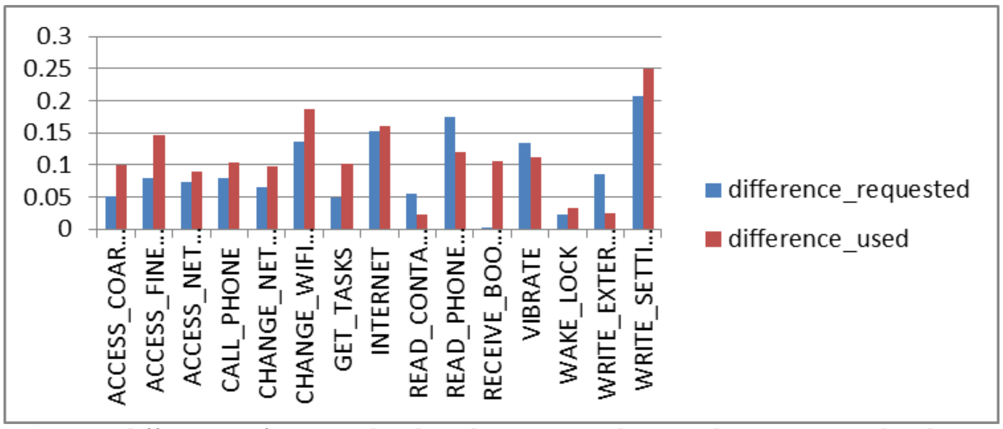


Fig. 1(c). differences of requested and used permissions between benign apps and malware

Fig. 1(c) depicts the frequency differences of requested and used permissions between benign apps and malware. The average frequency difference of requested permissions is 0.09, while that of used permissions is 0.11, indicating there are no obvious differences regarding permissions (requested or used) between the two app types. **Fig. 2 (a)** and **Fig. 2 (b)** show the top 20 registered broadcast events in benign apps and malware respectively. Among the events, the overlap ratio is nearly 0.65. For the overlapped events, as shown in Fig.2 (c), the average difference is less than 0.07. As shown in **Fig. 3**, there are few differences in frequency of sensitive program behaviors shown up in the two app types. The frequency differences of the four sensitive program behaviors are all within 0.1.

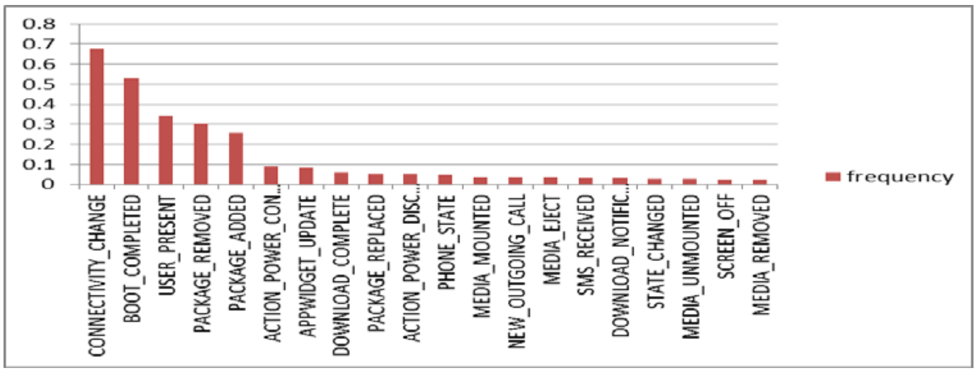


Fig. 2(a). top 20 registered broadcast events in benign apps

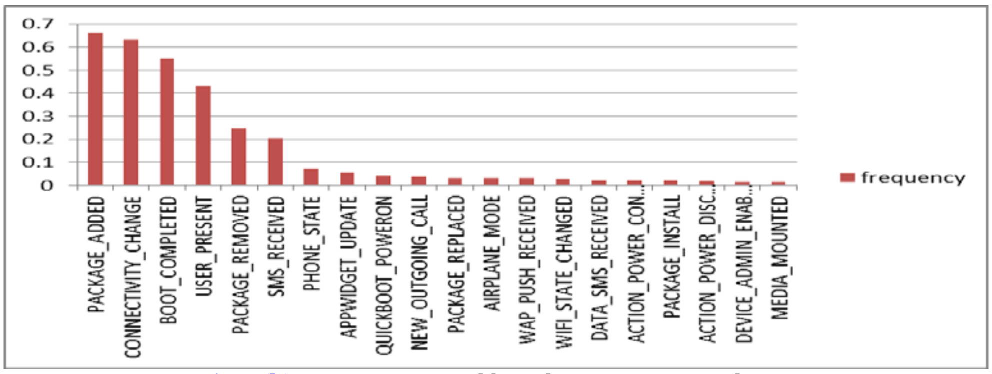


Fig. 2(b). top 20 registered broadcast events in malware

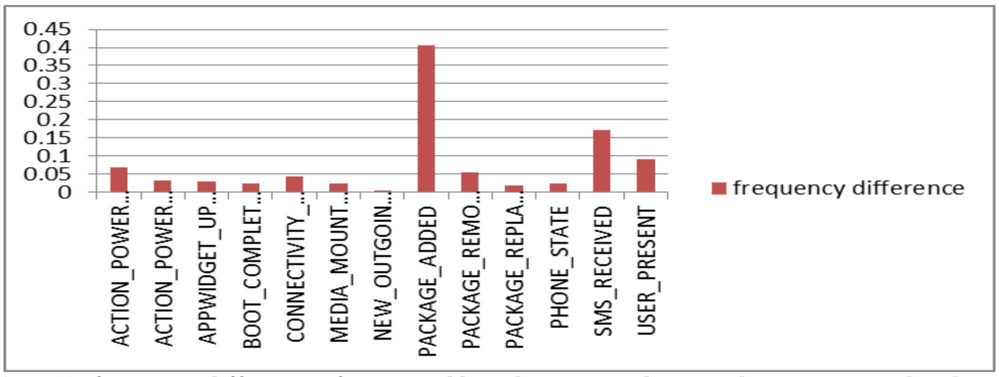


Fig. 2(c). frequency difference of registered broadcast events between benign apps and malware

As shown in the above figures, we can draw the conclusion that there is no distinguishable difference in superficial features between benign apps and malicious apps Therefore it is not desirable to employ superficial features to detect Android malware. However, benign apps and malicious apps behave quite differently. The root cause is how permissions granted by the users are used in the apps. All operations that may potentially do harm to the users' benefit (financial interest, user privacy or system security) are under the protection of application permissions in Android system. Only after being granted corresponding permissions can an application access sensitive data or carry out dangerous operations. Benign applications usually use permissions in a way that won't go against the interests of the users; On the contrary, malware usually intentionally use permissions in a way that will harm users' benefit. Thus, it is reasonable to detect Android malware by how Android application permissions are used. From such a perspective, we propose our risk classification based Android malware detection approach.

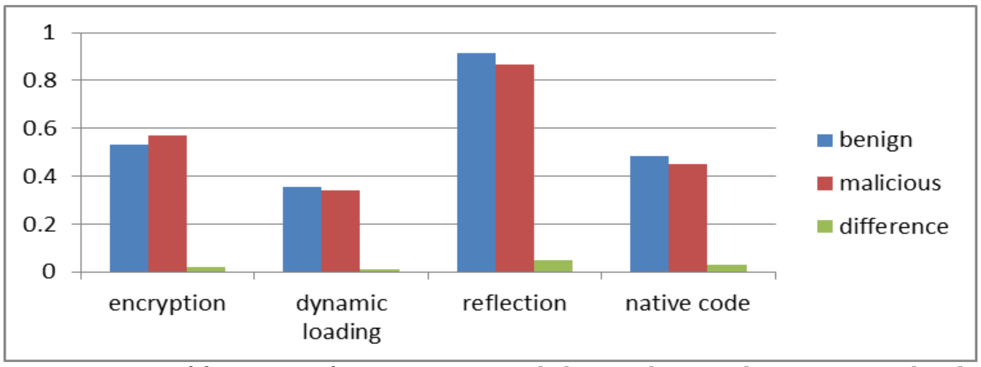


Fig. 3. comparison of frequency of sensitive program behaviors between benign apps and malware

3. Methodology

Our method tries to detect malicious Android applications by quantitative analysis. Specifically, Android application risks are classified into the following five specific categories: money risk, privacy risk, sensitive program behavior risk, network connection risk, highly dangerous permission risk. To describe the overall risk of an application, comprehensive risk is proposed and computed based on the former five. We implement a quantitative calculation model to compute application risks and employ the result as the application feature. We then apply machine learning algorithms to detect malware automatically.

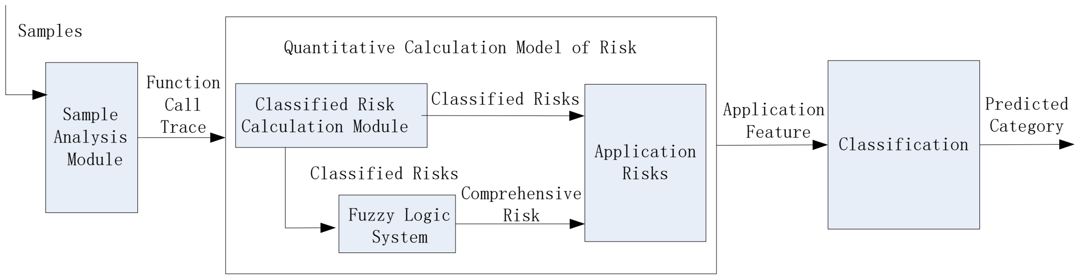


Fig. 4. workflow of our approach

As shown in **Fig. 4**, our method consists of three phases. In the beginning, we analyze all the samples to extract function call traces corresponding to permission methods that belong to used permissions. Afterwards, application risks used as application feature are computed by the quantitative calculation model. During the last phase, the application feature is fed to the selected machine learning algorithms to build the classifiers that will classify the samples.

3.1 Feature extraction

As Android application permissions are mapped to specific permission methods which can be used to describe how permissions are used. Specifically, we use function call traces of permission methods to represent how permissions are used in a certain app. Function call traces are automatically extracted through Python scripts that utilize core APIs provided by Androguard [36], and the APIs are modified to meet our needs.

Firstly, we analyze the APK file to extract global package information and filter all public available ad packages which can be assumed security unrelated. After that, the dex file in the APK file is decompiled into smail files, which are analyzed to extract used permissions and the function call graph of the application. Using the mapping relationship between permissions and permission methods provided by PScout[4], we mark all permission methods in the function call graph. Afterwards, we traverse the function call graph to extract the path that contains any marked nodes. Finally, we combine all the nodes on the path as a function call trace. Usually, one permission method has multiple function call traces, only parts of them are malicious. To filter security unrelated function call traces, we choose only the traces that satisfy either of the following two conditions.

1) Condition 1

As shown in **Table 1**, a single highly risky permission can cause security risks to users without the cooperation of any other permission. The corresponding function call traces are extracted directly.

Table 1. highly risky permissions

Permission	Risk
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed
CALL_PRIVILEGED	Allows an application to call any phone number, without going through the Dialer user interface for the user to confirm the call being placed.
INSTALL_PACKAGES	Allows an application to install packages
KILL_BACKGROUND_PROCESSES	Allows an application to kill background process
SEND_SMS_NO_CONFIRMATION	Allows an app to send SMS messages without user input or confirmation
SET_DEBUG_APP	Configures an application for debugging
PROCESS_OUTGOING_CALLS	Allows an app to intercept outgoing calls

2) Condition 2

In some cases, there are two or more permissions involved in a function call trace, i.e. there are at least two permission methods corresponding to two different permissions in a function call trace. In such cases, only when the combination of those corresponding permissions is of high risk should the function call trace be extracted. Existing permission-based approaches such as [27] [28] use permission combinations to detect Android malware. To determine whether a combination of permissions is risky, a conservative strategy is adopted to ensure that all security related function call traces are extracted. We extend the union of the permission combinations proposed in [27] and [28] (when k equals 5). The combinations used in our approach are shown in Table 2.

Table 2. risky permission combinations

Permission combination	**Risk**
ACCESS_FINE_LOCATION, INTERNT	Fine-grained location can be leaked through internet
ACCESS_FINE_LOCATION, SEND_SMS	Fine-grained location can be leaked by sending SMS messages
ACCESS_COARSE_LOCATION, INTERNT	coarse-grained location can be leaked through internet
ACCESS_COARSE_LOCATION, SEND_SMS	coarse-grained location can be leaked by sending SMS messages
INSTALL_SHORTCUT, UNINSTALL_SHORTCUT	Allows malicious app redirect its shortcut to other benign apps
READ_CONTACTS,INTERNET	Contacts can be leaked through internet
READ_CONTACTS,SEND_SMS	Contacts can be leaked by sending SMS messages
RECEIVE_SMS, WRITE_SMS	Allows the malware to remove traces of its activity
RECORD_AUDIO, INTERNET	The recorded audio can be leaked through internet
RECORD_AUDIO, SEND_MMS	The recorded audio can be leaked by sending multimedia messages.

Function call traces are converted into strings using the approach proposed in [37]. We classify all the strings into three categories: set_{benign} (set of normal function call traces that only show up in benign apps), $set_{malicious}$ (set of malicious function call traces that only show up in malware), $set_{intersection}$ (set of function call traces that show up in the two types of apps simultaneously). Due to the length of a function call trace varies with the number and size of methods in it, we calculate its MD5 for the sake of convenience.

Meanwhile, we use $risk_{benign}$, $risk_{malicious}$, and $risk_{intersection}$ to represent the risk factors of the three sets respectively. Specifically, we assign 0 to $risk_{benign}$, for the reason that it is impossible for a function call trace that only shows up in benign applications to be malicious. We assign 1 to $risk_{malicious}$, for the reason that it is of high probability that a function call trace is malicious if it only shows up in malware. When it comes to $set_{intersection}$, it is reasonable to believe that its risk factor (for simplicity, we will use *rf* to represent risk factor in the rest of the paper) should be lower than 0.5. To determine the best value that achieves the highest accuracy, we consider the following five values: 0.1, 0.2, 0.3, 0.4 and 0.5, and will analyze the influences of different $risk_{intersection}$ on detection performance in section 4.3.

$Set_{malicious}$ may contain some normal function call traces when using rule 2. To eliminate the interference of normal function call traces, we define malicious trusted probability (for convenience, we will use P_{mt} to represent malicious trusted probability in the rest of the paper) to represent the probability of a function call trace to be malicious. Specifically, we consider the following four cases: 0.6, 0.7, 0.8 and 0.9, and will analyze the influences of different *Pmt* on detection performance in section 4.3.

3.2 Quantitative calculation model

Some researchers attempt to detect Android malware from the perspective of Android application risks. Sarma et al. [38] detected Android malware based on the observation that apps in the same category request similar permissions and one app is highly suspicious if it applies the permissions barely applied by other apps of the same category. Specifically, they selected 26 permissions, and defined Rare Critical Permissions and Rare Pairs of Critical Permissions to calculate the risk of an app. In [39], Peng et al. proposed Probabilistic Generative Model for risk scoring, and they applied the Bayes model to calculate the generative probability of an app through the probabilities of each requested permission. Afterwards, the generative probability was converted to a risk score by the risk scoring function. At last, it was judged by the rank of risk score which measured whether an app was risky or not. For instance, an app will be highly risky if its risk score ranks in top 1%.

As discussed earlier, the differences of the requested permissions between benign apps and malware are tiny, while the basis of the two above approaches are requested permissions, so the two approaches are deficient inherently. Another limitation of the approaches is that as none of them analyze the categories of application risks in a detailed way, their results carry limited information, and can't tell what threat an Android application may cause (financial losses, leakage of user privacy, or damage to the system security). To solve such problems, we implement a detailed analysis of application risks and propose a risk classification based approach.

3.2.1 Risk classification

Five specific types of risk are closely associate with used permissions or sensitive program behaviors, while the comprehensive risk is associate with the five types of specific risk. In this section, we firstly describe the details how we classify application risks. Then we present the quantitative calculation model, and propose an approach to compute the comprehensive risk.

1. Money risk

Money risk refers to the potential financial losses caused by the use of money –sensitive permissions. These permissions are listed in **Table 3.**

Table 3. money risk related permissions

Permission	Risk level	Risk
CALL_PRIVILEGED	High	Allows an app to call privileged numbers
CALL_PHONE	Average	Allows an app to directly call phone numbers
INTERNET	Low	Allows an app to connect to the internet
SEND_SMS	Average	Allows an app to send SMS messages
SEND_SMS_NO_CONFIRMATION	High	Allows an app to send SMS messages without user input or confirmation
SEND_MMS	Average	Allows an app to send MMS messages
PROCESS_OUTGOING_CALLS	Average	Allows an app to intercept outgoing calls
USE_SIP	Average	Allows an app to make/ receive Internet calls

2. Privacy risk

Privacy risk refers to potential leakage of user privacy caused by the use of privacy-sensitive permissions, which are listed in **Table 4.**

Table 4. privacy risk related permissions

Permission	Risk level	Risk
ACCESS_FINE_LOCATION	High	Allows an app to access to fine-grained location
ACCESS_COARSE_LOCATION	Average	Allows an app to access to coarse-grained location

GET_ACCOUNTS	Average	Allows an app to access to all accounts of the device
READ_SMS	High	Allows an app to read SMS messages
RECEIVE_SMS	High	Allows an app to receive SMS messages
RECEIVE_MMS	High	Allows an app to receive MMS messages
READ_CONTACTS	High	Allows an app to read all the contacts on the device
READ_CALENDAR	High	Allows an app to read calendar events
READ_CALL_LOG	High	Allows an app to read the user's call log
READ_EXTERNAL_STORAGE	Average	Allows an application to read from external storage
READ_HISTORY_BOOKMARKS	Average	Allows an app to read user's Browser history and bookmarks
READ_PROFILE	High	Allows an app to read the user's personal profile data
RECORD_AUDIO	High	Allows an app to record audio
READ_USER_DICTIONARY	Low	Allows an app to read the user's dictionary
RECEIVE_WAP_PUSH	Average	Allows an app to receive WAP messages

3. Network connection risk

Complementary to privacy risk, network connection risk refers to the potential privacy risk when the device connects to other devices or networks. There are four main tunnels through which user privacy may be leaked. As shown in **Table 5**, they are Internet, Bluetooth, SMS and NFC respectively.

Table 5. network connection risk related permissions

Permission	**Risk level**	**Risk**
BLUETOOTH	Average	Allows an app to create Bluetooth connections
INTERNET	High	Allows an app to connect to the internet
NFC	Average	Allows an app to create NFC connections
SEND_SMS	High	Allows an app to send SMS messages

4. Highly dangerous permission risk

Highly dangerous permission risk refers to the potential risk caused by the use of risky permissions. Besides the permissions associated with the above three types of risk, highly dangerous permissions include all permissions whose risk level is dangerous or above. For instance, the permission WRITE_SECURE_SETTINGS is highly risky because it allows an app to modify system security configuration.

5. Sensitive program behavior risk

Android malware usually apply techniques such as native code, dynamic loading, Java reflection and code encryption to thwart static detection. Unfortunately, there are no corresponding permissions mapped to those sensitive program behaviors. Therefore, we propose sensitive program behavior risk to indicate the potential threat caused by these sensitive program behaviors.

6. Comprehensive risk

Comprehensive risk refers to the overall risk. It is a fuzzy variable and its value differs with risk preference and risk-bearing ability of the users.

3.2.2. Risk Calculation

The value of classified risk is computed by Formula (1), it equals the sum of risk values of all the used permissions of an app. The risk value of a single permission is the sum of risk values of function call traces of all permission methods corresponding to that permission. While the risk value of a single function call trace equals the product of risk factor, malicious trusted probability, the harmonic factor of function call trace and the unit risk value of its corresponding classified risk.

$$risk_{classified} = \sum_{i}\sum_{j=0}^{l-1} rf_{i,j} * c_i * p_{mt} * h_j \tag{1}$$

Where $\{i, j, l \mid i \in Perm,\ j \in Ctrace_i,\ l = len(Ctrace_i)\}$, Perm represents the set of used permissions, $Ctrace_i$ represents the set of function call traces of permission i, while l is the number of $Ctrace_i$.

In Formula (1), *rf* represents the risk factor, *c* represents the unit risk value corresponding to the permission. The unit risk value of all five types of classified risk is defined in **Table 6**. It depends on the category of risk and the risk level of the permission. *Pmt* represents the malicious trusted probability, and *h* represents the harmonic factor, which is computed by Formula (2).

$$h_i = \begin{cases} 1 & ,i \le N \\ \dfrac{1}{\sqrt{i-N}} & ,i > N \end{cases} \tag{2}$$

Where N represents the average number of function call sequences of the permission, i represents the i th sequence of the permission.

The reason that we define a harmonic factor is to weaken the correlation between the number of function call traces and the classified risk. As the value of classified risk is positive correlated to the number of function call traces, it favors large samples which results in such a phenomenon: the larger the traces, the greater value of classified risks. Without the harmonic factor, in some cases, the classified risk value of a benign app may be greater than that of a malware, which is undoubtedly unreasonable. Thus, we present the harmonic factor. As shown in Formula (2), we use the square root of i subtracting N as the denominator to compute the harmonic factor when i is greater than N. It should be noted that we have considered the logarithm (base to 2, e and 10) and the linear function but found that the quadratic function do a better job than the other two.

As illustrated in **Table 6**, there is only one risk level (and one unit risk value) defined for sensitive program behavior risk and highly dangerous permission risk. The method of calculating the risk values of the two is to multiply the unit risk value and the number of sensitive program behaviors or highly dangerous permissions.

Table 6. unit risk value of classified risk

Category	Risk level	Unit risk value
Money risk	High	5
	Average	3
	low	1
Privacy risk	High	4
	Average	2
	low	1
Network connection risk	High	3

	Average	2
	low	1
Highly dangerous risk	Average	3
Sensitive program behavior risk	Average	3

3.2.3. Fuzzy logic system

As discussed earlier, comprehensive risk is a fuzzy notion whose value varies among different users. For some users, an app is of low risk, while some others may consider it highly risky. For the users that care more about money risk, high privacy risk may be tolerable. For the users that pay more attention to privacy security, high money risk may be acceptable. Inspired by [36], we have designed and implemented a fuzzy logic system to compute the comprehensive risk.

Typically, as shown in **Fig. 5**, a fuzzy logic system consists of three phases: fuzzification, interference and defuzzification. During the fuzzification phase, a crisp set of input data is transformed to a fuzzy set using fuzzy linguistic variables, fuzzy linguistic terms, and membership functions of fuzzy input variables. During the inference phase, an inference is made according to a set of fuzzy rules. During the last phase, the resulting fuzzy output is mapped to a crisp output using the membership functions of the pre-defined output fuzzy variables.

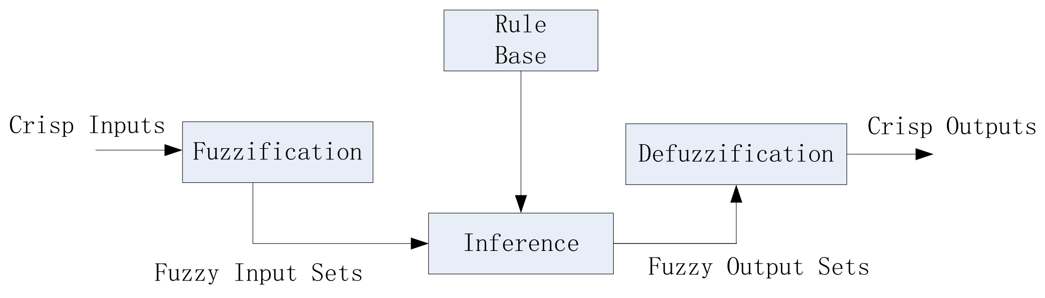


Fig. 5. overview of the fuzzy logic system

Linguistic variables: Linguistic variables include input and output variables of the fuzzy logic system whose values are words or sentences from a natural language, instead of numerical values [40]. In our system, input variables include five types of risk (Money risk, privacy risk and so on), while the output variable refers to the comprehensive risk. Usually, a linguistic variable is decomposed into a set of linguistic terms. In our system, all linguistic terms are shown in **Table 7**.

Table 7. linguistic terms

1) Money risk：	
LOW	:={(0.0, 1.0), (5.0, 1.0), (10.0, 0.0)}
AVERAGE	:= {(5.0, 0.0), (15.0, 1.0), (25.0, 0)}
HIGH	:= {(20.0, 0.0), (35.0, 1.0), (100, 1)}
2) Privacy risk	
LOW	:= {(0.0, 1.0), (10.0, 1.0), (15.0, 0.0)}
AVERAGE	:= {(10.0, 0.0), (20.0, 1.0), (30.0, 0.0)}
HIGH	:= {(25.0, 0.0), (35.0, 1.0), (100.0, 1.0)}

3) Sensitive program behavior risk	
LOW	:= {(0.0, 1.0), (2.0, 1.0), (3.0, 0.0)}
AVERAGE	:= {(2.0, 0.0), (6.0, 1.0), (8.0, 0.0)}
HIGH	:= {(6.0, 0.0), (10.0, 1.0), (20, 1.0)}
4) Network connection risk	
LOW	:= {(0.0, 0.0), (15.0, 1.0), (20.0, 0.0)}
AVERAGE	:= {(15.0, 0.0), (30.0, 1.0), (40.0, 0.0)}
HIGH	:= {(35.0, 0.0), (50.0, 1.0), (100.0, 1.0)}
5) Highly dangerous permission risk.	
LOW	:= {(0.0, 0.0), (8.0, 1.0), (12.0, 0.0)}
AVERAGE	:= {(8.0, 0.0), (20.0, 1.0), (30.0, 0.0)}
HIGH	:= {(25.0, 0.0), (30.0, 1.0), (100.0, 0.0)}
6) Comprehensive risk	
LOW	:=10
AVERAGE	:=30
HIGH	:=60

Rule base: Included in the rule base, fuzzy rules are in the form of "if-then" (for instance, if the temperature is above 86 degree, then adjust the air conditioner to cooling mode) and used to compute output fuzzy functions. The computing process is analyzed qualitatively and resistant to the change of malicious features that only affect the application feature, i.e. the five specific kinds of application risk and the comprehensive risk. While the rule base is used to calculate the comprehensive risk and can be rebuilt due to the change of users' risk preference instead of the change of the features of malware. Currently, two categories of rules are used in our fuzzy logic system. One of them is the independent rule, i.e. there is only one condition in the rule. Another one is the compound rule with more than one condition. More specifically, two compound rules are designed. Privacy risk and network connection risk make the first one, because they are closely related. Privacy risk and money risk make the second one, because the two are the most concerned types of risk. **Table 8** illustrates the results of fuzzy operation inside a compound rule.

It is worth noting that new fuzzy rules can be added to the rule base to describe the interaction of various risks in a more fine-grained level and compute fuzzy output functions in a more fine-grained way. Users can define new fuzzy rules that meet their needs in order to improve the weight of impact of some type of risk. For example, those who have low bearing capacity of money risk can increase money risk related fuzzy rules, thereby increasing the impact of money risk on the value of comprehensive risk. It will in turn improve the weight of impact of money risk on the classifier, which means money risk will have more weight in determining whether an unknown sample is benign or malicious.

Table 8. fuzzy matrixes

Risk level/result	Low	Average	High
Low	Low	Average	High
Average	Average	High	High
High	High	High	High

Table 9. one sample of fuzzy rules

```
RULE BLOCK
AND:MIN;
RULE 1: IF money is low then risk is low;
RULE 1a: IF money risk is average then risk is average;
RULE 1b: IF money risk is high then risk is high;

RULE 2: IF privacy risk is low and connection risk is low, then risk is low;
RULE 2a: IF privacy risk is average and connection risk is low risk is average
RULE 2b: IF privacy risk is average and connection risk is average, then risk is high;
...........
```

Table 9 shows one sample of fuzzy rules, where the risk represents comprehensive risk. In our fuzzy logic system, AND operator is used to evaluate fuzzy rules and combine the results of all the rules. Specifically, the Min operation is used in our system. As shown in Formula (3), COGS (Centre of Gravity for Singletons) is used to defuzzify the resulting fuzzy outputs.

$$U = \frac{\sum_{i=1}^{p} u_i v_i}{\sum_{i=1}^{p} v_i} \tag{3}$$

Where U represents the crisp output, u represents output fuzzy variable, v represents the membership function after accumulation.

3.3 Classification

We employ Weka [41] to implement machine learning algorithms to detect Android malware automatically. Our application feature consists of application risks. Five machine learning algorithms (RandomForest, J48, LibSVM, NaiveBayes, and BayesNet) from different classifiers are employed in order to select the best one according to the performance. These algorithms belong to three families, RandomForest and J48 belong to the decision tree algorithms, and LibSVM is a library for Support Vector Machine algorithm, while NaiveBayes and BayesNet come from the Bayes algorithms.

4. Evaluation

4.1 Dataset

Our dataset consists of 16,116 benign applications and 14,448 malware samples. The former were collected from three Android application markets: Appchina [42], Anzhi [43] and Google Play, the latter were downloaded from VirusShare [44]. The training dataset contains 10,000 benign applications and 5000 malware samples, both of which are randomly chosen from the raw dataset. The testing dataset includes 3000 benign applications and 3000 malware samples that are randomly chosen from the rest of the dataset.

We measure True Positive Ratio (TPR), True Negative Ratio (TRN) and accuracy to evaluate the performance. As shown in Formula (4), *TPR* represents the proportion of malware instances that are correctly classified:

$$TPR = \frac{TP}{TP + FN} \tag{4}$$

Where *TP* is the number of malware samples correctly detected and *FN* is the number of malware samples identified as benign apps. As shown in Formula (5), *TNR* represents the proportion of benign apps that are correctly classified.

$$TNR = \frac{TN}{TN + FP} \tag{5}$$

Where *TN* is the number of benign apps correctly detected and *FP* is the number of benign apps classified as malware. As shown in Formula (6), accuracy is used to evaluate the overall performance, it equals the result of the sum of benign and malware instances correctly identified divided by the whole number of the dataset instances.

$$Accuracy = \frac{TP + FN}{TP + FN + TN + FP} \tag{6}$$

4.2 Performance

As stated in section 3.3, we employ NaiveBayes, BayesNet, LibSVM, J48 and RandomForest to build classifiers. Based on Weka, 10-fold across validation is used to generate classifiers for every machine learning algorithm. Afterwards, we employ the classifiers to detect the testing set, and select the best one in each group. Firstly, to evaluate the influences of the risk factor and malicious trusted probability over each classifier's performance and determine the optimum parameters (i.e. risk factor and malicious trusted probability) of the quantitative calculation model, we conduct twenty experiments where the values of risk factor and malicious trusted probability are described in section 3.1.

4.2.1. Performance of different risk factors and malicious trusted probabilities

As shown in **Fig. 6 (a)** (for subscript x_y, x represents risk factor and y represents malicious trusted probability, i.e. *Pmt*), when risk factor remains unchanged, performance improves with the increase of *Pmt* and achieves the best (accuracy, TPR and TNR is 0.932, 0.925 and 0.939 respectively) when malicious trusted probability equals 0.8. When malicious remains unchanged, as depicted in **Fig. 6 (b)**, performance gets better with the increase of risk factor and reaches the top when risk factor equals 0.4. Therefore, for the quantitative calculation model, we set risk factor to 0.4 and malicious trusted probability to 0.8 for all the following experiments.

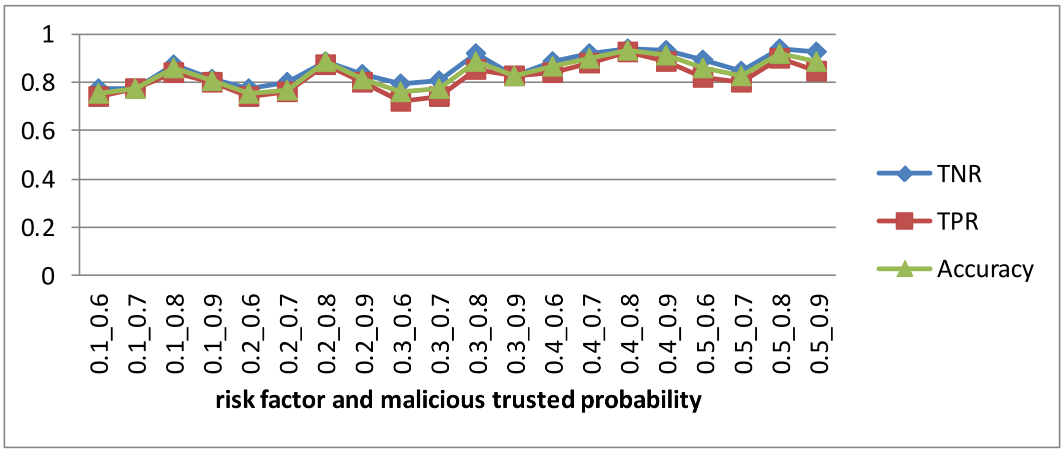


Fig. 6 (a). performance of different risk factors and malicious trusted probabilities

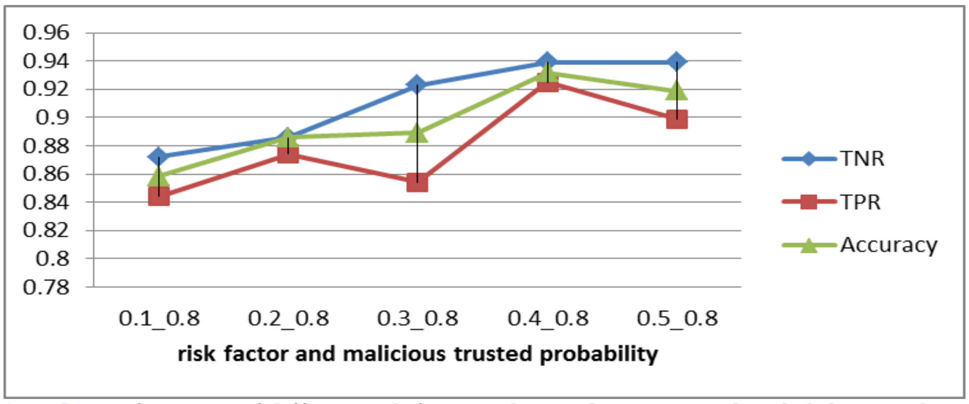


Fig. 6(b). performance of different risk factors when malicious trusted probability equals 0.8

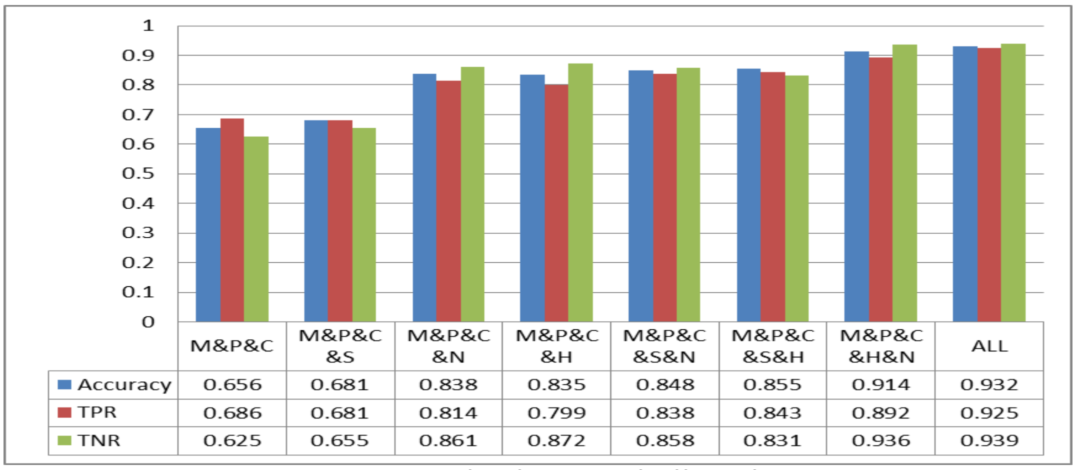
4.2.2. Performance of different features

To evaluate the influences of different features over the performance, we conducted eight experiments with different features as shown in **Table 10**. Money risk and privacy risk are the two basic types of risk, so they are included in every feature of the experiments. For the sake of convenience, we use M, P, S, N, H, and C to represent money risk, privacy risk, sensitive program behavior risk, network connection risk, highly dangerous permission risk and comprehensive risk respectively.

Table 10. application features

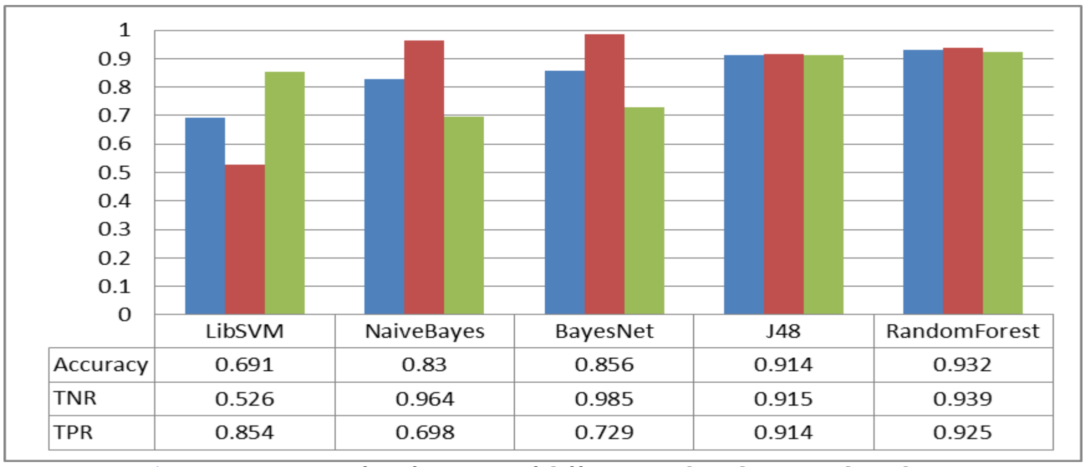
Group	Application Feature
1	M, P, C
2	M, P, C, S
3	M, P, C, N
4	M, P, C, H
5	M, P, C, N, S
6	M, P, C, N, H
7	M, P, C, H, S
8	M, P, C, N, S, H

As shown in **Fig. 7**, accuracy improves with the increase of the number of the risk types in the feature, and achieves the highest when the feature includes all the six types of risk. This happens if the number of risk types included in the evaluated feature is too few. There is a significant deviation between the real feature and the evaluated one, which misleads the classifiers and results in poor classification. When the number of risk types increases, the evaluated feature draws close to the real one, which results in sound performance.



	M&P&C	M&P&C &S	M&P&C &N	M&P&C &H	M&P&C &S&N	M&P&C &S&H	M&P&C &H&N	ALL
🟦 Accuracy	0.656	0.681	0.838	0.835	0.848	0.855	0.914	0.932
🟥 TPR	0.686	0.681	0.814	0.799	0.838	0.843	0.892	0.925
🟩 TNR	0.625	0.655	0.861	0.872	0.858	0.831	0.936	0.939

Fig. 7. comparison of performance of different features



	LibSVM	NaiveBayes	BayesNet	J48	RandomForest
Accuracy	0.691	0.83	0.856	0.914	0.932
TNR	0.526	0.964	0.985	0.915	0.939
TPR	0.854	0.698	0.729	0.914	0.925

Fig. 8. comparison of performance of different machine learning algorithms

When there are only four risk types in the evaluated feature, the performance of the combination of M&P&C&N (or M&P&C&H) outperforms the combination of M&P&C&S, and when there are five risk types in the evaluated feature, the performance of the combination of M&P&C&H&N ranks the first. Such phenomenon illustrates that there are distinct differences in network connection risk and highly dangerous permission risk between benign apps and malware and the two have high similarities in sensitive program behaviors.

4.2.3. Run-time performance

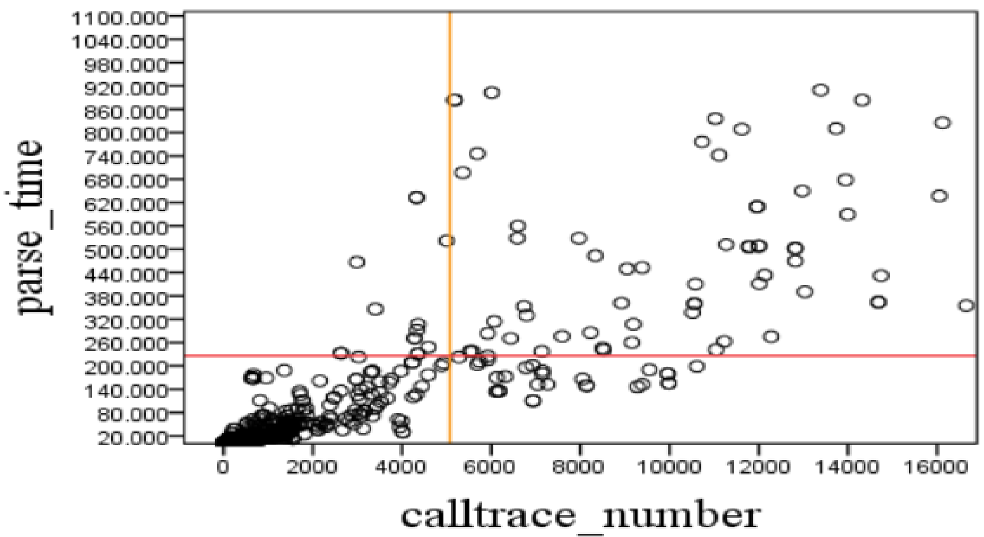


Fig. 9 (a). relationship between time overhead and the number of function call traces

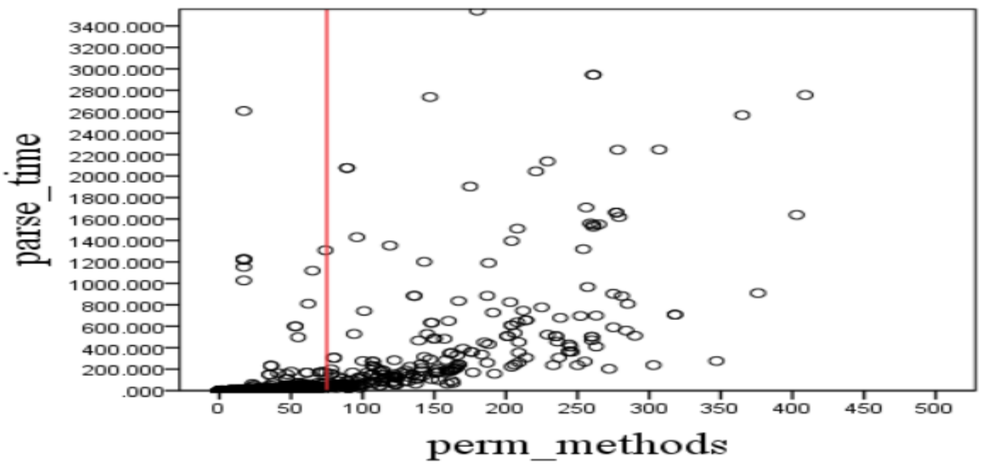


Fig. 9 (b). relationship between time overhead and the number of permission methods

All experiments are conducted on a machine equipped with Intel (R) Core (TM) i7-410MQ CPU @ 2.5 GHz processor and 16GB of physical memory. The operating system is Ubuntu 14.04 LTS (64 bit). We use SPSS to analyze the relationship between time overhead and the number of function call traces. As shown in **Fig. 9 (a)**, the time overhead is in positive correlation with the number of function call traces. The red line in **Fig. 9 (a)** represents the average time overhead (231.18s). While the orange one represents the average number of function call trace of samples in the dataset, which is 5959.05. The ratio of time overhead to the number of function call traces is 38.9ms. When the number of function call trace is under 2200, time overhead is within 100s. **Fig. 9 (b)** depicts the relation between time overhead and the number of permission methods. It is obvious that the number of permission methods of most samples ranges from 0 to 300. The red line represents the average number of permission methods, which is nearly 74.6. And the ratio of time overhead to the number of permission methods is 3.09s per method.

5. Related work

One important direction in the field of Android malware detection concentrates on the inner structural information of an app. Similar to pervious work such as [45], [46] and [47], our approach also focuses on the inner structural information: function call traces of permission methods. [45] proposed a similarity-based approach, which relied on the similarities among apps to detect malware. Similarity score between two apps was computed based on their method similarity represented by NCD distance. While the NCD distance was measured by the similarity between the control flow graphs of the methods in the two apps. [46] employed code structures to characterize Android malware families, extracted the control flow graph of every method in the sample with the help of Androguard, and then applied text mining techniques to classify different Android malware families. Most similar to our work, the approach proposed in [47] extracted the function call graph of an app and applied a labeling function to label all the nodes of the graph by a 15-bit sequence. Afterwards, it calculated the hash-value for a given node and its direct neighboring nodes using a XOR operation. The hash-value is then embedded in a vector used by Support Vector Machine to classify Android malware.

Compared with the aforementioned works, our work differs in the following three aspects. By filtering unrelated methods, our approach only targets the permission methods instead of all the methods in the APK samples. The function call trace extracted in our approach is actually a runnable trace, and it contains more context information than control flow graphs (in [45] and [46]) and hash-values used in [47]. Unlike [47], application risks are considered in our approach. We classify application risks into five categories and analyze different application risks in multi-dimensions, which helps improve the performance. More importantly, the accuracy of our method outperforms the one presented in [47]. As shown in **Table 11**, the results of our approach outperform other existing works and have the highest detection accuracy and the best AUC performance.

Table 11. comparison with existing work

Related work	**Number of samples (malicious/benign)**	**TPR**	**TNR**	**Accuracy**	**AUC**
Liang et al. [27]	1260/711	0.875	0.835	-	-
Sarma et al. [38]	121/158,062	-	-	-	0.85-0.91
Peng et al. [39]	378/ 482,514	-	-	-	0.94-0.96
Hugo et al. [47]	12,158/135,792	-	-	0.89	-
Our approach	3000/3000	0.925	0.939	0.932	0.983

6. Discussion

In reality, Android malware evolve to avoid detection by changing malicious features aggressively which results in different forms of function call traces of permission methods. Our approach outperforms feature-based methods and is resistant to such changes. Based on the fact that Android malware will always cause security risks to users, it is feasible to detect them through security risks. As long as Android malware rely on Android permission methods to conduct malicious behaviors, our method can extract the function call traces of the permission methods used in the malware and compute the security risks and then judge whether the application is benign or malicious, regardless of the implementation details of functions call traces such as adding dummy methods or deleting some unimportant methods.

As a typical static analysis approach, our approach suffers the inherent limitations of static analysis, and it can be bypassed by techniques such as Java reflection, dynamic loading, obfuscation etc. To alleviate the problem, we employ sensitive program behavior risk to indicate the risk brought by sensitive program behaviors. As our method needs to traverse the function call graph of a given sample to extract function call traces, it is more time-consuming compared with some existing work. But every approach has to achieve a balance between efficiency and performance. For our method, as an off-line detection approach, accuracy is more important and it is acceptable that the average detection time is at the level of minute.

7. Conclusion and Future work

In this paper, we have presented a risk-based approach for detecting Android malware. Based on the observation that the main differences between benign applications and Android malware root in the way how Android permissions are used, we extract the function call traces of permission methods to describe how an application utilizes user-granted permissions. After calculating application risks, machine learning algorithms are applied to detect Android malware.

We classify Android application risks into money risk, privacy risk, network connection risk, highly dangerous permission risk, sensitive behavior risk and introduce comprehensive risk to describe the overall risk of an application based on the former five, and present a quantitative calculation mode and a fuzzy logic system to calculate the comprehensive risk. All risk categories are embedded in a feature vector, and five machine learning algorithms are employed to automatically detect Android malware. The results show that our scheme is capable of detecting Android malware at a satisfying accuracy rate.

For future work, we plan to optimize the algorithm of extracting function call trace to reduce the time overhead. Furthermore, to improve detection rate, we are going to conduct a hybrid Android malware analysis, i.e. combine static analysis and dynamic analysis, to solve the problems entangled with static approaches.

References

[1] IDC: Smartphone OS Market share 2015 ,http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[2] 360: mobile phone security situation report 2014, http://www.199it.com/archives/325900.html
[3] Wei X, Gomez L, Neamtiu I. and Faloutsos M., "Permission evolution in the Android ecosystem," in *Proc. of Computer Security Applications Conference*, 31-40, 2012. Article(CrossRefLink)
[4] Au K W Y, Zhou Y F, Huang Z, Lie D., "PScout: analyzing the Android permission specification," in *Proc. of the 2012 ACM conference on Computer and communications security*. ACM, 217-228, 2012. Article(CrossRefLink)
[5] Barrera D, Kayacik, H. G, Van Oorschot P C and Somayaji A., "A methodology for empirical analysis of permission-based security models and its application to android," in *Proc. of ACM Conference on Computer and Communications Security*, CCS 2010, Chicago, Illinois, USA, October. 73-84, 2010. Article(CrossRefLink)
[6] Johnson R, Wang Z, Gagnon C and Stavrou, "A. Analysis of Android Applications' Permissions," in *Proc. of IEEE Sixth International Conference on Software Security and Reliability Companion*. 45-46, 2012. Article(CrossRefLink)
[7] Felt A P, Chin E, Hanna S, Song D and Wagner D., "Android permissions demystified," in *Proc. of ACM Conference on Computer and Communications Security,* CCS 2011, Chicago, Illinois, USA, October. 627-638, 2011. Article(CrossRefLink)
[8] Nauman M, Khan S, Zhang X., "Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints," in *Proc. of ACM Symposium on Information, Computer and Communications Security*, *ASIACCS* 2010, Beijing, China, April. 328-332, 2010. Article(CrossRefLink)

[9] Ongtang M, Mclaughlin S, Enck W and McDaniel P., "Semantically Rich Application-Centric Security in Android," *Security & Communication Networks*, 5(6):658-673, 2009. Article(CrossRefLink)
[10] Felt A P, Wang H J, Moshchuk A, Hanna S and Chin E., "Permission re-delegation: attacks and defenses," *Usenix Conference on Security.* USENIX Association, 22-22, 2011. Article(CrossRefLink)
[11] Dietz M, Shekhar S, Pisetsky Y, Shu A and Wallach DS., "Quire: lightweight provenance for smart phone operating systems," *Dissertations & Theses*, 23-23, 2011. Article(CrossRefLink)
[12] Bugiel S, Davi L, Dmitrienko A, Fischer T and Sadeghi AR., "XManAndroid: A new Android evolution to mitigate privilege escalation attacks," *Technical Report*, Technische Universität Darmstadt, TR-2011-04, 2011. Article(CrossRefLink)
[13] Conti M, Nguyen V T N, Crispo B., "CRePE: Context-Related Policy Enforcement for Android," in *Proc. of Information Security, International Conference*, ISC 2010, Boca Raton, Fl, Usa, October 25-28, 2010, Revised Selected Papers. 331-345, 2010. Article(CrossRefLink)
[14] Zhou Y, Zhang X, Jiang X and Freeh W V., "Taming Information-Stealing Smartphone Applications (on Android)," in *Proc. of Trust and Trustworthy Computing International Conference*, Trust 2011, Pittsburgh, Pa, Usa, June 22-24, 2011. Proceedings. 93-107, 2011. Article(CrossRefLink)
[15] Sakamoto S, Okuda K, Nakatsuka R and Yamauchi T., "DroidTrack: tracking and visualizing information diffusion for preventing information leakage on Android," *Journal of Internet Services and Information Security (JISIS)* 4.2, 55-69, 2014. Article(CrossRefLink)
[16] Nauman M, Khan S, Zhang X and Seifert JP, "Beyond Kernel-Level Integrity Measurement: Enabling Remote Attestation for the Android Platform," in *Proc. of Trust and Trustworthy Computing, Third International Conference*, TRUST 2010, Berlin, Germany, June 21-23, 2010. Article(CrossRefLink)
[17] Song F, Touili T., "Model-Checking for Android Malware Detection," *Programming Languages and Systems*. Springer International Publishing, 216-235, 2014. Article(CrossRefLink)
[18] Reina A, Fattori A, Cavallaro L., "A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors," *Eurosec*, 2014. Article(CrossRefLink)
[19] Zhang Y, Yang M, Xu B, Yang Z and Gu G., "Vetting undesirable behaviors in android apps with permission use analysis," *Computer and Communications Security*, 9:611-622, 2013. Article(CrossRefLink)
[20] Lindorfer M, Neugschwandtner M, Weichselbaum L and Fratantonio Y, "ANDRUBIS -- 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in *Proc. of Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, IEEE Computer Society, 3-17, 2014. Article(CrossRefLink)
[21] Enck W, Gilbert P, Chun B-G, McDaniel P, Sheth A., "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Transactions on Computer Systems*, 32(2):393-407, 2014. Article(CrossRefLink)
[22] Droidbox, http:code.google.com/p/droidbox.
[23] Yan, Lok-Kwong, and Heng Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," *USENIX security symposium*. 2012. Article(CrossRefLink)
[24] Rastogi V, Chen Y, Enck W., "AppsPlayground: automatic security analysis of smartphone applications," in *Proc. of ACM Conference on Data and Application Security and Privacy*. 209-220, 2013. Article(CrossRefLink)
[25] Sun, Mingshen, J. C. S. Lui and X. Jiang, "Design and implementation of an Android host-based intrusion prevention system," in *Proc. of the 30th Annual Computer Security Applications Conference*. ACM, pp.226-235, 2014. Article(CrossRefLink)
[26] Bläsing T, Batyuk L, Schmidt A D, Camtepe SA., "An Android Application Sandbox system for suspicious software detection," in *Proc. of International Conference on Malicious and Unwanted Software. IEEE*, S166, 2010. Article(CrossRefLink)

[27] Enck W, Ongtang M and Mcdaniel P, "On lightweight mobile phone application certification," *Computer and Communications Security*, 2009. Article(CrossRefLink)
[28] Liang, Shuang, and Xiaojiang Du, "Permission-combination-based scheme for Android mobile malware detection," in *Proc. of the 2014 IEEE International Conference on Communications, Sidney, Australia*, pp. 2301-2306, June 2014. Article(CrossRefLink)
[29] Zhou, W., Zhou, Y., Jiang X. and Ning, P., "DroidMoss: Detecting repackaged smartphone applications in third-party Android marketplaces," in *Proc. of the second ACM conference on Data and Application Security and Privacy, CODASPY'12*, 2012. Article(CrossRefLink)
[30] Feng Y, Anand S, Dillig I, Aiken A., "Apposcopy: semantics-based detection of Android malware through static analysis," *The ACM Sigsoft International Symposium*, 576-587, 2014. Article(CrossRefLink)
[31] Grace M, Zhou Y, Zhang Q, Zou S and Jiang X., "RiskRanker: scalable and accurate zero-day android malware detection," in *Proc. of International Conference on Mobile Systems, Applications, and Services. ACM*, 281-294, 2012. Article(CrossRefLink)
[32] Zhou Y, Wang Z, Zhou W and Jiang X., "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proc. of Annual Network & Distributed System Security Symposium*, 2012. Article(CrossRefLink)
[33] Yuan Z, Lu Y, Wang Z, Xue Y., "Droid-Sec: deep learning in android malware detection," *ACM Sigcomm Computer Communication Review*, 44(4):371-372, 2014. Article(CrossRefLink)
[34] Aafer Y, Du W and Yin H., "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," *Security and Privacy in Communication Networks. Springer International Publishing*, 86-103, 2013. Article(CrossRefLink)
[35] Arp D, Gascon H, Rieck K, Spreitzenbarth M and Hübner M., "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," *Network and Distributed System Security Symposium*. 2014. Article(CrossRefLink)
[36] Androguard. https://code.google.com/p/androguard/.
[37] Cesare S, Xiang Y., "Classification of malware using structured control flow," *Eighth Australasian Symposium on Parallel and Distributed Computing. Australian Computer Society, Inc*. 61-70, 2010. Article(CrossRefLink)
[38] Sarma B P, Li N, Gates C, Potharaju R and Nita-Rotaru C., "Android permissions: A perspective combining risks and benefits," in *Proc. of Acm Symposium on Access Control Models & Technologies Ser Sacmat '*, 13-22, 2012. Article(CrossRefLink)
[39] Peng H, Gates C, Sarma B, Li N and Qi Y., "Using probabilistic generative models for ranking risks of Android apps," in *Proc. of Conference on Computer and Communications Security*. 241-252, 2012. Article(CrossRefLink)
[40] Driankov, Dimiter, Hans Hellendoorn, and Michael Reinfrank, "An introduction to fuzzy control," *Springer Science & Business Media*, 2013. Article(CrossRefLink)
[41] Weka, http://www.cs.waikato.ac.nz/ml/weka.
[42] Appchina, http://www.appchina.com.
[43] Anzhi, http://www.anzhi.com.
[44] Virus share, http://www.virusshare.com .
[45] Hassana, Doaa, Matthew Might, and Vivek Srikumar, "A Similarity-Based Machine Learning Approach for Detecting Adversarial Android Malware," *Technical report UUCS-14-002, School of Computing, University of Utah*, 2014. Article(CrossRefLink)
[46] Suarez-Tangil G, Tapiador J E, Peris-Lopez P, Blasco J., "Dendroid : A text mining approach to analyzing and classifying code structures in Android malware families," *Expert Systems with Applications*, 41(4):1104-1117, 2013. Article(CrossRefLink)
[47] Gascon H, Yamaguchi F, Arp D and Rieck K., "Structural detection of android malware using embedded call graphs," in *Proc. of ACM Workshop on Security and Artificial Intelligence*. 45-54, 2013. Article(CrossRefLink)

Yi-lin Ye was born in Jiujiang, Jiangxi, China in 1987. Now he is a Ph.D. candidate in the university. His research interests include information security and cloud security.

Li-fa Wu was born in Qichun, Hubei, China in 1968. He received his Ph.D. from Nanjing University in 1998. He is currently a professor in PLA University of Science and Technology. His research fields concern network security, protocol engineering and satellite communication.

Zheng Hong was born in Yingtan, Jiangxi, China in 1979. He received his Ph.D. from PLA University of Science and Technology in 2007. Now he is an associate professor in the university. His research fields concern network security and protocol reverse engineering.

Kang-yu Huang was born in Yichuan, Jiangxi, Chian in 1985. He received his received M.S degree in network engineering from PLA University of Science and Technology in 2009. His research fields concern network security.