Neural Comput & Applic (2018) 30:3353–3361
https://doi.org/10.1007/s00521-017-2914-y

ORIGINAL ARTICLE

HEMD: a highly efficient random forest-based malware detection framework for Android

Hui-Juan Zhu[1,2,3] · Tong-Hai Jiang[1,3] · Bo Ma[1,3] · Zhu-Hong You[1,3] · Wei-Lei Shi[1] · Li Cheng[1,3]

Received: 7 November 2016 / Accepted: 13 February 2017 / Published online: 10 March 2017

Abstract Mobile phones are rapidly becoming the most widespread and popular form of communication; thus, they are also the most important attack target of malware. The amount of malware in mobile phones is increasing exponentially and poses a serious security threat. Google's Android is the most popular smart phone platforms in the world and the mechanisms of permission declaration access control cannot identify the malware. In this paper, we proposed an ensemble machine learning system for the detection of malware on Android devices. More specifically, four groups of features including permissions, monitoring system events, sensitive API and permission rate are extracted to characterize each Android application (app). Then an ensemble random forest classifier is learned to detect whether an app is potentially malicious or not. The performance of our proposed method is evaluated on the actual data set using tenfold cross-validation. The experimental results demonstrate that the proposed method can achieve a highly accuracy of 89.91%. For further assessing the performance of our method, we compared it with the state-of-the-art support vector machine classifier. Comparison results demonstrate that the proposed method is extremely promising and could provide a cost-effective alternative for Android malware detection.

✉ Zhu-Hong You
zhuhongyou@ms.xjb.ac.cn

✉ Wei-Lei Shi
shiwl@ms.xjb.ac.cn

[1] The Xinjiang Technical Institute of Physics and Chemistry, Chinese Academy of Sciences, Ürümqi 830011, China

[2] University of Chinese Academy of Sciences, Beijing 100049, China

[3] Xinjiang Laboratory of Minority Speech and Language Information Processing, Ürümqi 830011, China

Keywords Random forest · Malware detection · Android · Support vector machine · Requested permissions

1 Introduction

Mobile phones or smart phones are accelerating the progress of mobile industry. The number of smart phone users in recent years is also increasing exponentially. Modern people enjoy a variety of convenient services, such as mobile banking service, mobile client mall, network search, social network service, through mobile phones anywhere, anytime as long as they can access the network. With the great convenience and swiftness provided by smart phones, significant threats of security vulnerability have also increasingly highlighted, which is mainly caused by the ongoing emergence of malicious software (also known as malware) on the mobile platforms [1]. Recently, hackers are expanding their attacks from existing PCs to smart phone terminals. As various types of significant user information are scattered through smart mobile phone, such as user preferences, user phone number and user's current location, there is a possibility of enduring parlous damages by the threat of hacker attacks [2, 3].

A report [4] from Alcatel-Lucent's Motive Security Lab stated the percent of infected mobile phones observed on a monthly basis since December 2012 which using data was averaged from actual mobile deployments. This report showed that the world's mobile devices infected by malware had 16 million units in 2014, accounting for 0.68% of all mobile devices. The malware infection rate in mobile devices increased 25% in 2014, and the growth was 20% in comparison with 2013, while the infection rate fell to 0.5% in 2015, but at the end of the first quarter it rose again to the 0.75%; in the first half of 2015, the amount of Android malware samples increased more than doubled. It can be seen from these reports that we are facing a huge challenge in terms of malware detection for mobile terminal, especially for the Google Android platform, which has been dominant in the market [5]. The main reason for the growth of malware on Android platform is its open-source policy as well as its tolerance for market app verification. Besides, unofficial repositories are also allowed, where software developers can upload apps, including cracked apps, Trojan horses or malware repackaged into a normal app. Unfortunately, recent studies have indicated that the existing popular solutions for detecting malware are far behind the growing popularity of mobile apps.

Up to now, most of the malware detection methods are based on the traditional content signatures [6–9], and their work patterns are to compare each app against the known malware signatures in database. The major weakness of this kind of approaches is that they can only detect the identified instances of malware precisely, but it unable to identify the metamorphic or unseen instances of malware [10]. This traditional approaches never keep up with the speed in which malware is created and evolved. These methods are commonly referred to as static analysis [11, 12]. Static analysis, as the name indicates, it is the process of examining malware without executing apps. In addition to signatures, permissions were also often adopted to detect Android malware [13]. Generally speaking, static analysis is an economical as well as a fast approach, but it produces less information, thus limiting the extraction of possible features from malware activities. Moreover, the attackers have developed a variety of methods, such as code confusion techniques, to escape inspection by static analysis [14, 15].

Instead of using predefined signatures or requested permissions and so on from static analysis for malware detection, dynamic analysis [7, 16] also provides some effective ways to detect malware by observing the dynamic behavior and features of apps, such as to use characteristic and behavior-based method [17–21]. Dynamic analysis involves running the sample in order to analyze its execution traces to extract useful information and enrich the feature set. Some behavioral traits, such as system calls [22], permissions app requested, battery consumption and premium SMS, can be captured using dynamic analysis exist. Dynamic analysis is usually more complex than the static analysis, and it often suffers from low detection rates and requires more resources to reduce the number of false positives reported. And analyzing the dynamic behavior of app only reveals information about what the malware was doing at that time [23]; besides, its cost and time consumption are also amazing [24]. Egele et al. [25] offers a complete overview of automated dynamic malware analysis technique. They are usually inclined to high false positives because of lack of sufficient training. Thus, we here introduce machine learning method to detect malware in Android platform.

Application of machine learning algorithms for Android malware classification is an emerging area [24, 26–30]. Recently, a number of machine learning algorithms provide some effective ways to dynamically extract malware patterns, and the experimental results from previous studies indicate that they can achieve outstanding detection rates [12, 31]. In this paper, we propose HEMD, a highly efficient method for detecting Android malware using the machine learning method RF [32]. HEMD performs a broad static analysis and gathers malware-related features from Android Package (Apk) file. These features include permissions, monitoring system events, sensitive APIs and permission rate. For instance, a great percentage of current malwares send premium SMS messages and thus request the Android.permission.SEND_SMS permission and so on. Consequently, the key features are extracted from such malwares, which can be mapped to a specific set of features associated with the corresponding permissions, monitoring system events, sensitive APIs and permission rate. This working mechanism enables HEMD to identify automatically combinations and patterns of features which are the indicatives for malwares by means of machine learning methods. When performed on an actual data set using tenfold cross-validation, HEMD can achieve a high accuracy of 89.91% which exhibits it is a credible method to protect against malicious destructive activities. For further assessing the performance of the proposed method, we compared it with the state-of-the-art SVM classifier under the same experimental conditions. Comparison results prove that the proposed method is a glaring candidate for detecting malicious activities.

The remainder of this paper is organized as follows: Sect. 2 briefly describes the proposed method's framework. Section 3 introduces our materials and methodology. Section 4 presents RF Classifier used in this paper. In Sect. 5, we present experimental results from RF classifier and also compare those with that of state-of-the-art SVM classifier. Finally, we conclude our work in Sect. 6.

2 The proposed model's framework

The framework of the HEMD proposed by this paper is illustrated in Fig. 1. Our framework consists of several components which offer adequate resources and mechanisms to recognize Android malware. Specifically, the first step is to execute the de-compilation module, in which all Apk files are reversed to Androidmanifest.xml and some Smali files through apktool [21, 33]. The second step is to execute the feature extraction module. Some key features

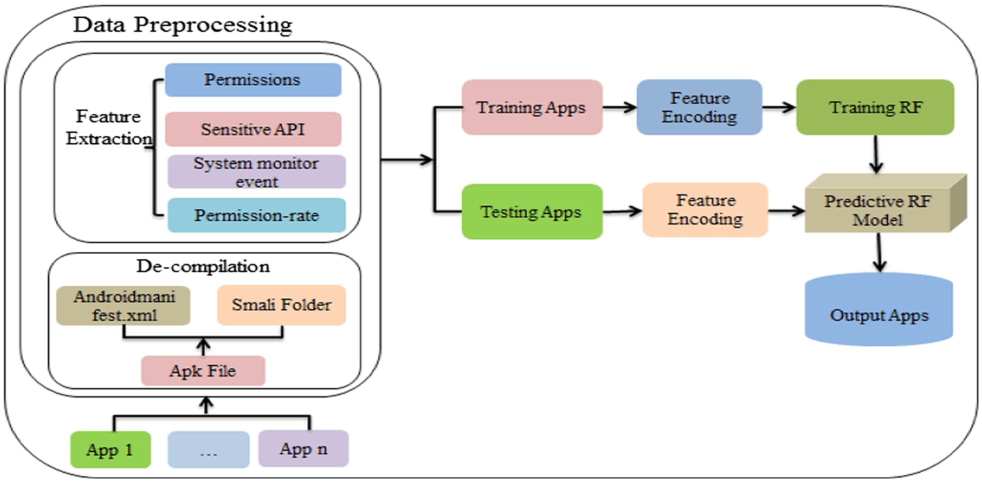

Fig. 1 Framework of the proposed model

are extracted in this module in accordance with some importantly and widely accepted measures, such as TF-IDF and cosine similarity [34]. The third step is to carry out the machine learning module, in which RF model is employed to build the classifier.

In this study, 600 benign and 600 malware apps are adopted as training data set and the other apps are used as testing data set.

3 Materials and methodology

The most concerned features, as mentioned above, are permissions, monitoring system events, sensitive API and permission rate, and these four groups of features are employed to characterize the behaviors of Android apps in this paper.

3.1 Data sources

The data set we used consists of 2130 Android apps, of which 1065 apps are benign and 1065 apps are malware. The benign apps are from official Android market and the malware apps are from http://virusshare.com/, and we named this data set as *Hemdds*. We perform our method on *Hemdds* data set and compare our method with SVM based on this data set.

3.2 Key features representation

General speaking, the collected data by monitoring resources in an Android environment play a very important role to detect malware. This part explains the key features used in the proposed method for detecting Android malware.

3.2.1 Permissions

The permission system is one of the most crucial security mechanisms introduced in the Android platform. To perform certain tasks on the mobile, such as mobile maps, each app has to explicitly request permissions from the user during the installation stage [7]. In order to better reflect the characteristics of the Android malware, the permissions requested by the training data set all were collected. A list of top 9 permissions are selected as the dangerous permissions in the proposed model in conformity with the ratio r (r = frequency in malware/frequency in benign software), which are shown in Table 1.

Table 1 Extracted permission features

Nos.	Permission name
1	android.permission.WRITE
2	android.permission.UPDATE_DEVICE_STATS
3	com.android.alarm.permission.SET_ALARM
4	android.permission.INSTALL_PACKAGES
5	com.android.browser.permission.WRITE_ HISTORY_BOOKMARKS
6	android.permission.WRITE_ SECURE_SETTINGS
7	com.android.browser.permission.READ_ HISTORY_BOOKMARKS
8	android.permission.RECEIVE_SMS
9	android.permission.SEND_SMS

3.2.2 Monitoring system events

The Android app consists of four components: activity, service, BroadcastReceiver, and content provider. These components work individually, and each component delivers messages (also known as intent) to other components to allow cooperation. The BroadcastReceiver is a class that does not have a user interface and which is able to run silently in the background, and its responsibility is to monitor broadcast events (e.g., intent) from the system to wake up the appropriate activities. For this

Table 2 Extracted monitoring system events

Nos.	Monitoring system events name
1	android.intent.action.DATA_SMS_RECEIVED
2	android.intent.action.BATTERY_CHANGED
3	android.intent.action.AIRPLANE_MODE
4	android.provider.Telephony.SMS_RECEIVED
5	com.google.android.c2dm.intent.RECEIVE
6	android.intent.action.QUICKBOOT_POWERON

reason, malware monitoring system events by BroadcastReceiver is a quite popular solution for malware developers [33, 35]. A typical example of a monitoring system event involved in malware [37, 38] is BOOT_COMPLETED, which is used to trigger malware directly after rebooting the smart phone. In order to better characterize the malware, we collected all monitored system events in malware and benign training set and counted separately the frequencies they appeared in these two sets of samples. After sorting the ratio r ($r =$ frequency in malware/frequency in benign software, if the denominator is 0, it will be replaced by 1), the top of 6 events are selected as key feature subsets to identify whether one app is belong to malware or benign, which are shown in Table 2.

3.2.3 Sensitive APIs

The Android platform offers a framework application programming interface (API) that apps can use it to interact with the Android system bottom. APIs have been demonstrated to play an important role in malware detection by previous studies [24, 36, 39, 40], and the API calls are contained in the .dex classes. The tool apktool [33] is used as the de-compiler in this research which parses .dex file to Smali files, and the information of API calls can be obtained in these Smali files. Combined with the points that users more concerned about (e.g., SMS operation, equipment information, contacts operation), the top of 12 sensitive API are selected and part of them is shown in Table 3.

Table 3 Part of selected sensitive API and URL

Type	API and URL
SMS-related API calls	sendTextMessage()
SMS-related API calls	getMessageBody()
Get the international mobile station identity (code) of mobile	getSubscriberId()
Get the cell phone number	getLine1Number()
Get the location information	getLastKnownLocation()
The operation of mobile phone contacts	content://com.android.contacts

3.2.4 Permission rate

In this paper, the permission rate (*prate*) is defined as formula (1):

$$prate = \frac{pnum}{ssize} \tag{1}$$

where *pnum* represents the total amount of permissions requested by one app and *ssize* is the size of Smali file (unit: MB) which generated by decompiled the app. The use of permission rate as one feature to detect malware is based on following reasons: First, the abuse of permissions is common in malware; second, normally, the more permissions a benign app requests, the more functions it provides, which mean the size of its Smali file will be also big. Therefore, the permission rate of the malware is usually greater than the benign one, and the permission rate can also become an effective feature of malware detection.

4 Random forest classifier

RF is a new classification and regression algorithm developed by Leo Breiman [41] which uses decision tree as base classifier. Each tree is built employing a bootstrap sample of data, and the candidate features set in each division is a random subset of the global features. Accordingly, RF takes advantage of two powerful machine learning techniques: bagging (bootstrap aggregation) [42] and random features selection for tree building. Each of the decision trees has a characteristic of low bias because it is un-pruned and grown fully; meanwhile, the correlation of individual trees is low due to the characteristics of bagging and random features selection. Therefore, RF produces an ensemble with low bias and low variance.

The final prediction result is derived from a set of prediction results through combination strategy, and the combination strategies commonly adopted include averaging (simple averaging, weighted average) and voting (majority voting, plurality voting and weighted voting). Compared with the single decision tree classifier (e.g., CART, C4.5) [34, 43–45], RF takes a significant performance increase. Although RF is not widely used in the Android malware detection now, it takes following characteristics that make it ideal for this target:

1. It is available when there are many more features than observations.
2. It takes great estimated performance, in spite of a lot of predictor variables may contain noise.
3. It is not prone to over-fitting.
4. It can handle a mixture of categorical and continuous predictors.

5. It has the characteristics of high quality and free implementations.
6. There is almost no need to continuously fine-tune parameters to achieve superior performance. Commonly, L (the number of decision trees) and *mtry* (the number of input features tried at each split) are the most significant parameters, and these two parameters will be seriously and carefully investigated in this research.

The RF classifier used in this paper, as well as the majority voting, is summarized as follows:

The RF classifier algorithm

1. Take L bootstrap samples from the *Hemdds* data set;
2. For each of the bootstrap samples, grow an un-pruned classification tree, with the following modification: at each node, select the optimal split from randomly sample *mtry* of the features, instead of from all features. (Note: when $mtry = tn$, it is equivalent to Bagging algorithm, where *tn* is the total amount of features.)
3. The new app will be predicted by gathering the predictions of the L trees with the majority voting method.

It is assumed that the training model including classifiers $\{c_1, c_2, \ldots, c_T\}$ and $c_i(x)$ is the output of classifier c_i on the sample x. In the classification task, every classifier c_i will predict an output from the class label set $\{o_1, o_2, \ldots, o_N\}$ (in this paper, the class label set is $\{\text{malware, benign}\}$); then, the majority voting method is introduced to combine these outputs to get the final predict class. In order to facilitate the discussion, the predict outputs of classifier c_i on the sample x are expressed as N-dimensional vector $\{c_i^1(x), c_i^2(x), \ldots, c_i^N(x)\}$, where $c_i^k(x)$ is the predict output of classifier c_i on the class label o_k. The final prediction classification result $C(x)$ is defined as gformula (2):

$$C(x) = \begin{cases} o_k, & \text{if } \sum_{i=1}^{T} c_i^k(x) > 0.5 \sum_{l=1}^{N} \sum_{i=1}^{T} c_i^l(x) \\ \text{reject}, & \text{otherwise} \end{cases} \quad (2)$$

5 Results and discussion

5.1 Select the parameters *mtry* and L in RF

In our method, the amount of features randomly sampled as candidates at each split (*mtry*) and the number of decision trees L in the RF will affect the performance of our model. Since a large number of trees and an appropriate *mtry* will lead to considerable computational cost and will affect the accuracy of the proposed method, it is crucial to find the suitable parameters. Figure 2 shows the prediction results of different parameters. According to Fig. 2a, keeping L to 500 and tuning *mtry* from 0 to 28 at intervals of 1, we can find that when setting the $mtry = 11$, an excellent result with accuracy of 90.05% can be obtained. When we set *mtry* to 11 and increase the value of L from 50 to 1000 at intervals of 50, the results are shown in Fig. 2b. Figure 2b demonstrates that with the increase in L, the accuracy rate gentle rise at the beginning, but it soon tends to be stable. Considering the time cost and accuracy of the algorithm, we eventually choose the most appropriate parameters of $mtry = 11$ and $L = 500$.

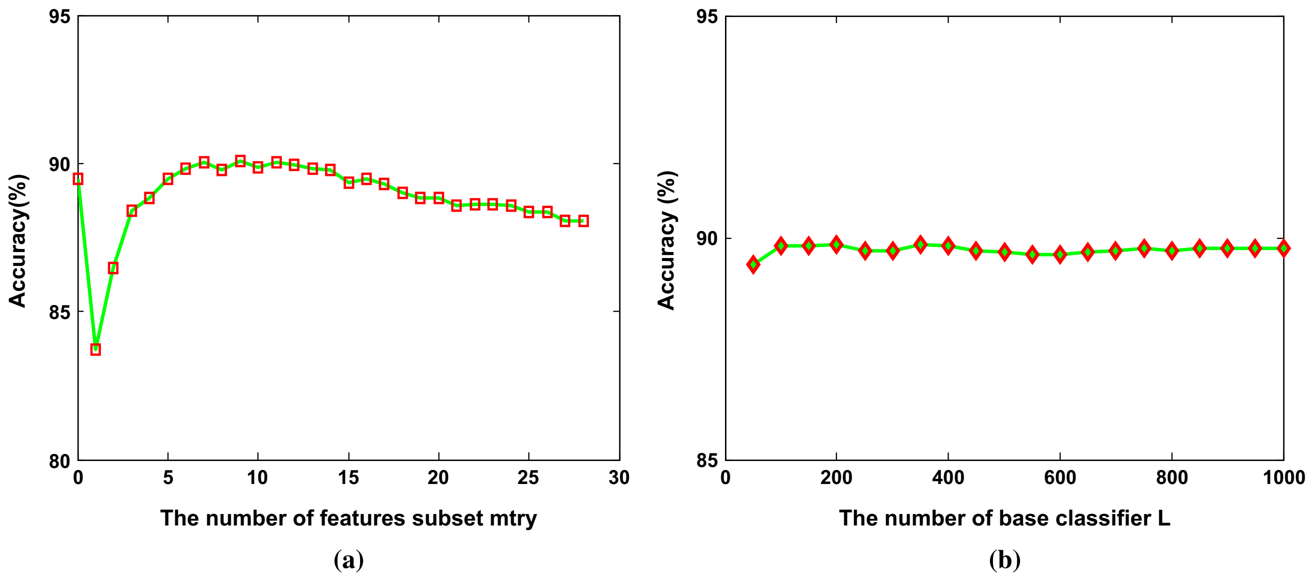

Fig. 2 **a** Percentage of prediction accuracy with increasing *mtry*; **b** percentage of prediction accuracy with increasing L

Table 4 Prediction results achieved by the proposed method with using tenfold cross-validation

Test set	Sensitivity (%)	Precision (%)	Accuracy (%)	MCC (%)	AUC (%)
1	90.35	89.57	89.2	78.29	89.88
2	88.78	84.47	87.32	74.66	86.3
3	86.92	87.74	87.32	74.65	88.12
4	90.29	86.92	88.73	77.52	88.23
5	90.29	87.74	89.2	78.43	91.32
6	93.52	89.38	91.08	82.23	90.66
7	90.48	88.79	89.67	79.35	88.84
8	92.52	93.4	92.96	85.92	93.51
9	97.27	87.7	91.55	83.55	92.02
10	91.82	92.66	92.02	84.02	94.21
Average	91.22 ± 2.67	88.84 ± 2.5	89.91 ± 1.84	79.86 ± 3.71	90.31 ± 2.39

5.2 Evaluation criteria

For the purpose of measuring the predictive performance of the HEMD method proposed in this paper, some evaluation criteria such as sensitivity, precision, accuracy, Matthews correlation coefficient (MCC), area under curve (AUC) and receiver operating characteristic curve (ROC) are introduced. Their definitions are shown as the formula (3–6):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{3}$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FN}) \times (\text{TN} + \text{FP}) \times (\text{TP} + \text{FP}) \times (\text{TN} + \text{FN})}} \tag{6}$$

where true positive (TP) is the amount of positive testing samples correctly predicted as positive, false positive (FP) is the amount of negative testing samples incorrectly predicted as positive, true negative (TN) is the amount of negatives testing samples correctly predicted as negative, false negative (FN) is the amount of negatives testing samples wrongly predicted as positive and MCC is a correlation coefficient that evaluate the quality of binary classifications in machine learning. Besides, AUC and ROC are often used together to evaluate the merits of binary classifier. ROC is a comprehensive index reflecting the sensitivity and specificity, and AUC is adopted together because sometimes the ROC curve does not clearly indicate which classifier is better.

5.3 Assessment of prediction ability

In the experiment, in order to test the stability of the proposed model and avoid the over-fitting, the tenfold cross-validation is utilized to measure the performance of HEMD. Specifically, all samples are randomly divided into ten disjoint subsets of approximately equal size; each takes one for the test set, the other nine copies for the training set to form a model, thus formed ten groups of test model.

The experimental results are shown in Table 4. Focused on the *Hemdds* data, it can be noticed that the average of accuracy and its standard deviation are 89.91 and 1.84%; the average of precision and its standard deviation are 88.84 and 2.5%; the average of sensitivity and its standard deviation are 91.22 and 2.67%. In particular, in the best case, the accuracy and the precision is up to 92.96 and 93.4%. We yield the average of MCC and its standard deviation which are 79.86 and 3.71%. The ROC curves perform on *Hemdds* data set is shown in Fig. 3, and the average of AUC and its standard deviation achieved by the proposed method are 90.31 and 2.39%.

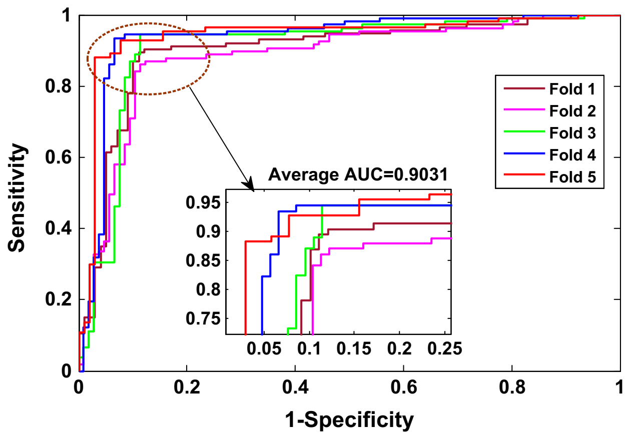

Fig. 3 Receiver operating characteristic (ROC) curves for RF classifier

Table 5 Prediction results achieved by SVM with using tenfold cross-validation

Test set	Sensitivity (%)	Precision (%)	Accuracy (%)	MCC (%)	AUC (%)
1	87.27	85.71	85.91	71.80	85.28
2	83.48	87.27	84.51	69	86.68
3	83.02	81.48	82.16	64.33	83.23
4	85.71	77.06	81.69	63.79	82.84
5	85.98	87.62	86.85	73.72	87.68
6	88.24	81.82	84.98	70.19	85.59
7	89.42	82.30	85.45	71.18	88.89
8	87.62	84.40	85.92	71.89	88.30
9	83.19	88.39	84.51	68.98	85.03
10	87.88	85.29	87.32	74.60	86.1
Average	86.18 ± 2.18	84.13 ± 3.31	84.93 ± 1.74	69.95 ± 3.41	85.96 ± 1.9
Proposed method	91.22 ± 2.67	88.84 ± 2.5	89.91 ± 1.84	79.86 ± 3.71	90.31 ± 2.39

5.4 Comparison between random forest and support vector machine

To evaluate the performance of the proposed method, we compared it with the state-of-the-art SVM classifier under the same experimental conditions. Table 5 shows the prediction results achieved by the proposed method and SVM classifier. It can be observed that the averages of sensitivity, precision, accuracy, MCC, AUC and their standard deviations for RF model are 91.22 ± 2.67, 88.84 ± 2.5, 89.91 ± 1.84, 79.86 ± 3.71 and 90.31 ± 2.39%, respectively. The averages of sensitivity, precision, accuracy, MCC, AUC and their standard deviations for SVM classifier are 86.18 ± 2.18, 84.13 ± 1.74, 84.93 ± 1.74, 69.95 ± 3.41 and 85.96 ± 1.9%, respectively.

Table 5 shows that the performance of the proposed method can achieve significant improvement over the SVM classifier. Specifically, the average accuracy achieved by the proposed method is improved by 4.98% compared to SVM. According to Fig. 3, it can be noticed that the result of average AUC achieved by the proposed method is 90.31%. Figure 4 shows that the result of average AUC achieved by SVM is 85.96%. This is due to the fact that there may contain a great quantity of information in the feature set, of course, noise data are common among them, which will affect the accuracy of the classifier, so the SVM did not perform well on this kind of feature set. Conversely, decision tree provides an explicit model describing the relationship between features and predictions, thus easing model interpretation. RF, as an ensemble of trees, inherits the ability to select 'important' features. Additionally, a large number of decision trees and the use of the majority voting that RF identified are extremely crucial for classifying Android apps coincided with expectations. It also can be seen that RF yields generalization error rate that compares favorably to SVM algorithm, yet is more robust to noise.

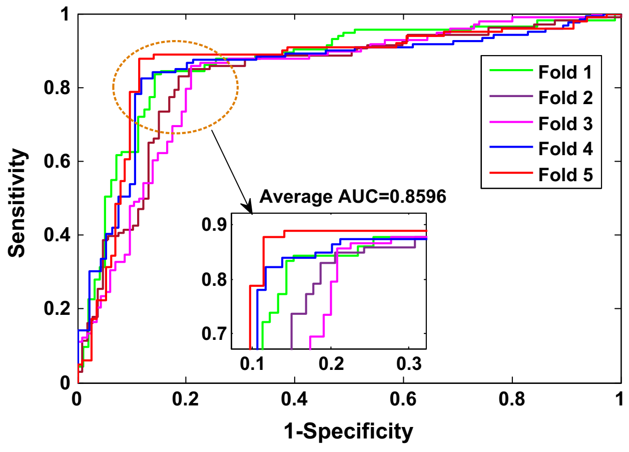

Fig. 4 Receiver operating characteristic (ROC) curves for SVM classifier

6 Conclusion

In this work, we propose a RF-based malware detection scheme for Android platform, and use permissions, monitoring system events, sensitive API and permission rate combinations as features to build a RF classifier, which can automatically distinguish Android malicious or benign apps. A significant advantage of our approach is that it can acquire key features involved in each app through simple and rapid static analysis method, and do not need to involve any dynamical tracing (e.g., system calls and battery consumption). Moreover, because these four groups of features are always available for each app, our approach can be generalized to all mobile apps. Excellent experimental results we achieved in actual data set demonstrate that the proposed model can accurately predict the Android malware. At the same time, we believe that our method has broad application prospects in the user information security area.

Acknowledgements This work is supported in part by the West Light Foundation of The Chinese Academy of Sciences, under Grants XBBS201313, and in part by the Thousand Talents Plan, under Grant Y32H251201. The authors would like to thank all anonymous reviewers for their constructive advices.

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K (2014) DREBIN: effective and explainable detection of android malware in your pocket. In: Network and distributed system security symposium
2. Werth D, Emrich A, Chapko A (2012) An ecosystem for user-generated mobile services. J Converg 3(4):35–40
3. Gnanaraj JWK, Ezra K, Rajsingh EB (2013) Smart card based time efficient authentication scheme for global grid computing. Hum Cent Comput Inf Sci 3(1):1–14
4. Motive Security Labs. Malware report—H1 2015 (2015) http://resources.alcatel-lucent.com/asset/189669
5. Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: behavior-based malware detection system for android. In: ACM workshop on security and privacy in smartphones and mobile devices, pp 15–26
6. Schmidt AD, Camtepe SA, Albayrak S (2010) Static smartphone malware detection. In: The 5th security research conference (Future Security 2010). Berlin, p 146
7. Sharma A, Dash SK (2014) Mining API calls and permissions for android malware detection. In: International conference on cryptology and network security. Springer, pp 191–205
8. Kou X, Wen Q (2011) Intrusion detection model based on android. In: 2011 4th IEEE international conference on broadband network and multimedia technology, pp 624–628
9. Bose A, Hu X, Shin KG, Park T (2008) Behavioral detection of malware on mobile handsets. In: ACM proceedings of the 6th international conference on mobile systems, applications, and services, pp 225–238
10. More SS, Gaikwad PP (2016) Trust-based voting method for efficient malware detection. Proced Comput Sci 79:657–667
11. Shabtai A, Moskovitch R, Elovici Y, Glezer C (2009) Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. Inf Secur Tech Rep 14(1):16–29
12. Chandramohan M, Tan HBK (2012) Detection of mobile malware in the wild. Computer 45(9):65–71
13. Huang CY, Tsai YT, Hsu CH (2013) Performance evaluation on permission-based detection for android malware. Advances in intelligent systems and applications-volume 2. Springer, Berlin, pp 111–120
14. Gupta BB, Tewari A, Jain AK, Agrawal DP (2016) Fighting against phishing attacks: state of the art and future challenges. Neural Comput Appl. doi:10.1007/s00521-016-2275-y
15. Moser A, Kruegel C, Kirda E (2007) Limits of static analysis for malware detection. In: IEEE computer security applications conference, 2007. Twenty-third annual, pp 421–430
16. Li Y, Li S, Song Q, Liu H, Meng QH (2014) Fast and robust data association using posterior based approximate joint compatibility test. IEEE Trans Indus Inf 10(1):331–339
17. Schmidt AD, Schmidt HG, Clausen J, Camtepe A, Albayrak S (2008) Enhancing security of linux-based android devices. In: 15th international Linux Kongress. Lehmann
18. Cheng J, Wong SHY, Yang H, Lu S (2007) Smartsiren: virus detection and alert for smartphones. In: Proceedings of the 5th international conference on mobile systems, applications and services. ACM, pp 258–271
19. Liu L, Yan G, Zhang X, Chen S (2009) Virusmeter: preventing your cellphone from spies. International workshop on recent advances in intrusion detection. Springer, Berlin, pp 244–264
20. Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices. ACM, pp 15–26
21. Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y (2012) "Andromaly": a behavioral malware detection framework for android devices. J Intell Inf Syst 38(1):161–190
22. Dini G, Martinelli F, Saracino A, Sgamdirra D (2012) MADAM: a multi-level anomaly detector for android malware. International conference on mathematical methods, models, and architectures for computer network security. Springer, Berlin, pp 240–253
23. Kapoor A, Dhavale S (2016) Control flow graph based multiclass malware detection using bi-normal separation. Def Sci J 66(2):138–145
24. Peiravian N, Zhu X (2013) Machine learning for android malware detection using permission and API calls. In: 2013 IEEE 25th international conference on tools with artificial intelligence, pp 300–305
25. Egele M, Scholte T, Kirda E, Kruegel C (2012) A survey on automated dynamic malware-analysis techniques and tools. ACM Comput Surv (CSUR) 44(2):1–42
26. Zhao M, Ge F, Zhang T, Yuan Z (2011) AntiMalDroid: an efficient SVM-based malware detection framework for android. International conference on information computing and applications. Springer, Berlin, pp 158–166
27. Yerima SY, Sezer S, Mcwilliams G (2014) Analysis of Bayesian classification-based approaches for android malware detection. IET Inf Secur 8(1):25–36
28. Narudin FA, Feizollah A, Anuar NB, Gani A (2016) Evaluation of machine learning classifiers for mobile malware detection. Soft Comput 20(1):343–357
29. Santos I, Devesa J, Brezo F, Nieves J, Bringas PG (2013) Opem: a static-dynamic approach for machine-learning-based malware detection. International joint conference CISIS'12-ICEUTÉ 12-SOCÓ 12 special sessions. Springer, Berlin, pp 271–280
30. Allix K, Bissyandé TF, Jérome Q, Klein J, State R (2016) Empirical assessment of machine learning-based malware detectors for Android. Empir Softw Eng 21(1):183–211
31. Ham HS, Kim HH, Kim MS, Choi MJ (2014) Linear SVM-based android malware detection. In: Frontier and innovation in future computing and communications, vol 301. Springer, pp 575–585
32. Elyan E, Gaber MM (2016) A fine-grained random forests using class decomposition: an application to medical diagnosis. Neural Comput Appl 27(8):2279–2288
33. Jang J, Kang H, Woo J, Mohaisen A, Kim HK (2015) Andro-autopsy: anti-malware system based on similarity matching of malware and malware creator-centric information. Digital Investig 14:17–35
34. Li W, Ge J, Dai G (2015) Detecting malware for android platform: an SVM-based approach. In: Cyber security and cloud computing (CSCloud), 2015 IEEE 2nd international conference, pp 464–469
35. Oulehla M, Malanik D (2016) Techniques that allow hidden activity based malware on android mobile devices. Int J Sci Eng Appl Sci (IJSEAS) 2(3):409–419
36. Chan PPK, Song WK (2014) Static detection of android malware by using permissions and API calls. In: IEEE 2014 international conference on machine learning and cybernetics, vol 1, pp 82–87
37. Wolfe B, Elish KO, Yao D (2014) Comprehensive behavior profiling for proactive android malware detection. In: International conference on information security. Springer, pp 328–344

38. Idrees F, Rajarajan M (2014) Investigating the android intents and permissions for malware detection. In: 2014 IEEE 10th international conference on wireless and mobile computing, networking and communications (WiMob). IEEE, pp 354–358
39. Aafer Y, Du W, Yin H (2013) DroidAPIMiner: mining API-level features for robust malware detection in android. In: International conference on security and privacy in communication systems. Springer, pp 86–103
40. Wu D J, Mao C H, Lee H M, Wu KP (2012) Droidmat: Android malware detection through manifest and API calls tracing. In: Information security (Asia JCIS), 2012 seventh Asia joint conference on. IEEE, pp 62–69
41. Ellis K, Kerr J, Godbole S, Lanckriet G, Wing D, Marshall S (2014) A random forest classifier for the prediction of energy expenditure and type of physical activity from wrist and hip accelerometers. Physiol Meas 35(11):2191
42. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
43. Ham HS, Choi MJ (2013) Analysis of android malware detection performance using machine learning classifiers. In: IEEE 2013 international conference on ICT convergence (ICTC), pp 490–495
44. Kim T, Choi Y, Han S, Chung J Y (2012) Monitoring and detecting abnormal behavior in mobile cloud infrastructure. In: 2012 IEEE network operations and management symposium, pp 1303–1310
45. Sahs J, Khan L (2012) A machine learning approach to android malware detection. In: IEEE intelligence and security informatics conference (EISIC), 2012 European, pp 141–147