HFA-MD: An Efficient Hybrid Features Analysis Based Android Malware Detection Method

Yang Zhao, Guangquan Xu(✉), and Yao Zhang

Tianjin Key Laboratory of Advanced Networking (TANK),
School of Computer Science and Technology,
Tianjin University, Tianjin 300350, China
{zhaoyang6621, losin, zzyy}@tju.edu.cn

Abstract. Lack of supervision and management of many Android third-party application markets has led to a growing number of malware on android platforms. This causes a serious privacy threat to the user's sensitive information. To solve this problem, in this paper, a new hybrid features analysis method aiming at Android malware detection is proposed, which obtains a hybrid feature vector by extracting the information of permission requests, API calls and runtime behaviors. The characteristic of this work is the use of machine learning classification algorithms to detect malicious software. In addition, the feature selection algorithm is used to further optimize the extracted information to remove some useless features. Our experiments are based on real-world Apps, and use five different classification algorithms to detect the malware. The experiment results show that our proposed hybrid feature extraction method can improve the accuracy rate of Android malware detection compared with using static methods alone.

Keywords: Android malware detection · Machine learning · Static analysis
Dynamic analysis · Feature selection

1 Introduction

With Internet-centric mobile applications becoming more and more popular, the varieties and quantities of mobile applications have been increasing rapidly. Due to the open-source nature and openness of the Android system and the fact that many Android third-party application markets do not have a rigorous application review mechanism, resulting in hackers and malware developer are more inclined to Android operating system as the preferred target of attack [1]. Reports from Symantec show that the number of malicious apps increased by 152% in 2015 and increased by 105% in 2016 [2]. The user's mobile phone privacy data has become an ideal target for malicious software to steal. Besides the threat to user privacy, malware may severely threat the underlying infrastructure since it may open a gate to the legal access if the core network is vulnerable in for example fog/edge computing or mobile edge computing. Mobile-edge Computing provides IT and cloud-computing capabilities within the

L. Wang et al. (Eds.): QShine 2017, LNICST 234, pp. 248–257, 2018.
https://doi.org/10.1007/978-3-319-78078-8_25

Radio Access Network (RAN) in close proximity to mobile subscribers, which are mostly mobile phones for users.

In recent years, some researchers have begun to introduce data mining and machine learning methods into Android malware detection [3]. The Machine learning-based detection method of extracting features for each Android APP is divided into static analysis or dynamic analysis. The static analysis method [4] has the advantages of fast detection and high efficiency. However, in some cases, the static analysis method may cause false positives, which reduce the overall accuracy of static detection. Moreover, utilizing code obfuscation techniques can bypass the use of static analysis method of detection. Therefore, it can be combined with dynamic analysis to improve the accuracy rate. To some extent, the dynamic analysis method [5] can bypass code obfuscation and other code protection mechanism, but the speed of detection is relatively slow.

In order to improve the shortcomings of the existing research methods, we propose a hybrid analysis method for the detection of the Android malware that integrates the advantages of static and dynamic analysis methods. In this study, the hybrid features vector is extracted using a hybrid feature analysis method. We train the five different machine learning classifiers with vectors, respectively, in order to find a more efficient detection method to deal with Android malware threats. Experimental results show that compared with single analysis method, the feature set extracted by hybrid analysis method is more efficient in training classifier.

The rest of this paper is organized as follows. Related works are discussed in Sect. 2. Section 3 briefly describes the Android malware detection model for we proposed approach and analysis the various feature extraction. Our research methodology is introduced in Sect. 3.4 including feature selection algorithms and machine learning classifier. Section 4 presents the experimental results. Finally, in Sect. 5, we conclude our work and proposal for future work.

2 Related Work

In recent years, there have been a lot of related research works applying machine learning methods in Android malware detection field. In the static analysis method, Chan et al. [6] proposed a static Android malware detection method that extracts the permissions and API calls characteristics of each APP as the feature vector set for classifier training. Drebin [7] makes a static analysis of the Android APK file, mainly from the manifest file to extract the permissions of the application request, the contained components and other information, while also analyzing the application of the sensitive API calls and some network addresses from the Dex file. Wu et al. [8] proposed a method of malicious behavior analysis based on static behavior characteristics. In the dynamic analysis method, Amos et al. [9] proposed STREAM, a feature vector collection framework, which accelerated the large-scale verification of machine learning classification of Android malware. STREAM is a distributed mobile malware detection framework that can automatically train and evaluate malware classifiers. Dash et al. [10] proposed DroidScribe, a method of automatic classification of Android malware based on dynamic runtime behavior analysis. Rieck et al. [11] proposed a

framework for automatically analysing the malware behavior using machine learning methods that perform behavioral analysis in an incremental manner, avoiding the run-time and memory overhead of previous methods.

3 The Proposed Detection Method

3.1 Architecture of the Proposed Approach

An overview of the methods we presented is shown in Fig. 1. The methods of using machine learning to detect malware are mainly divided into the following parts: data collection, feature extraction, feature data preprocessing, classifier model training and classification results. The entire malware detection process can be divided into two phases: the training phase and the testing phase. In the training phase, firstly, we extract feature vectors from benign software and malicious software respectively; secondly, the feature vectors are selected to remove the feature which have no effect on the classification results, and the optimized feature vectors are obtained; finally, a hybrid feature vector is formed as input of the classifier model, and then different classifier models are selected to train, and the classifier models are obtained through continuous training. In the detection phase, the unknown samples are detected by the obtained classifier model. Since the classifier models are obtained by means of training the hybrid feature vectors, the classifier models will output the detection results when the unknown samples are inputted into the classifier models in the detection.

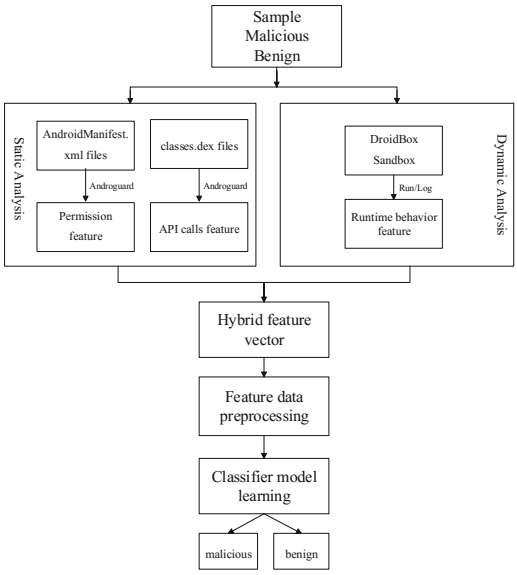

Fig. 1. Architecture of the proposed detection method

3.2 Feature Extraction

3.2.1 Static Analysis and Static Features

(1) Permission Extraction. Some applications that want to make malicious behavior must request appropriate sensitive permissions. The differences on these permissions information provide the theoretical feasibility of the permissions as a feature of the Android malware detection. In this paper, we use the open source tool Androguard [12] to extract the permission features from AndroidManifest.xml file in the APK package. We use the androlyze.py tools to extract sensitive permission features from normal samples and malicious samples respectively. By analyzing the results extracted from a large number of application samples, those applications that have malicious behavior often requires many sensitive permissions, such as malicious fee-absorbing applications often frequently apply for SMS-related permissions. In this case, excluding individual permissions that rarely appear, we counted the top 10 permissions that occurred most frequently, the permissions and their functions as shown in Table 1. In this step, we optimized the initial extracted permission feature sets and got 45 the highest relevance permissions as features. Each APP can be represented by a 45-dimensional vector $[Per]_{1\times 45}$, and each dimension corresponds to a permission. If an APP's AndroidManifest.xml file contains this permission, the value is 1, otherwise it is 0.

Table 1. Permissions and their functions

Permission	Functional description
INTERNET	Allow accessing to network connections
READ_PHONE_STATE	Allow reading only access to phone state
ACCESS_NETWORK_STATE	Allow accessing to network information
WRITE_EXTERNAL_STORAGE	Allow writing to external storage
READ_SMS	Allow reading of SMS messages
RECEIVE_BOOT_COMPLETED	Allow applications to boot up
RECEIVE_SMS	Allow to receive SMS messages
SEND_SMS	Allow to send SMS messages
CHANGE_WIFI_STATE	Allow to change Wi-Fi connectivity state
READ_CONTACTS	Allow accessing to user's contact information

(2) API Calls Extraction. The APIs studied in this paper refers to the function provided by the Android system itself. It may also trigger high-risk behaviors such as secretly connecting the network and sending SMS message for malicious deducting expenses. These APIs, which are related to sensitive data and high-risk behaviors, are referred to as sensitive APIs in this paper. As with the permissions information, there are significant differences in the use of these sensitive APIs due to the difference between benign software and malicious software. The malicious application of the number of calls to sensitive APIs is far more than the benign application, which can reflect the real behavior characteristics of an application to

some extent, and therefore can be used as a feature of the application to identify malicious behavior. We use the open source tools baksmali [13] and Androguard to reverse the analysis of classes.dex files, from which to extract the relevant sensitive APIs. In this step, we extracted the API calls features from a large number of sample sets, and then we used the filter feature selection algorithm Relief [14] to optimize it, and we count the number of times each API is called as the initial value of the relevant statistic vector component. After the feature selection process, we obtain an optimal set of features with 22 API calls, each of which can be represented by a 22-dimensional vector $[API]_{1\times 22}$, with each dimension corresponding to an API. Table 2 shows the 22 selected API calls.

Table 2. Sensitive API calls

API calls	
getDeviceID()	sendTextMessage()
getCellLocation()	sendDataMessage()
getLinelNumber()	getConnectionInfo()
getNetworkOperator()	getWifiState()
getSimSerialNumber()	setWifiEnabled()
getOutputStream()	getSubscriberId()
getInputStream()	addCopletedDownload()
getNetworkInfo()	AudioRecord.read()
startService()	AudioRecord.getRecordingState()
getLatitude()	MediaRecorder.setCamera()
getLogitude()	MediaRecorder.setOutputFile()

3.2.2 Dynamic Analysis and Dynamic Features

In the dynamic analysis phase, the main work of dynamic behavior acquisition is to collect the runtime behavior features of each application. In order to collect the runtime behavior features of the unknown sample as much as possible in the behavioral detection of the application, when the application installed in the simulator is running, we use the automated test tool monkey [15] to simulate the event flow to run all the components of the application. It can automatically test unknown samples and trigger the relevant malicious code, so that the monitoring program can record its malicious behavior.

We used the open source tools DroidBox [16] to monitor the runtime behavior of the application. We install and run each APP on DroidBox, and then use automated test techniques to monitor whether each APP has malicious behavior such as automatic connection to the network, sending malicious SMS messages, and obtaining privacy information and so on. In this step, we count the number of occurrences of each runtime behavior feature as the initial value of the relevant statistic vector component. After the feature selection process, we collect a total of 20 features (i.e., runtime behavior features) for each monitored APP from a large variety of aspects such as the battery, binder, network, user activity. Among them, behavior_sentSMS represents the behavior of

sending SMS messages, behavior_openingKeyboard is the behavior that opens keyboard input, and behavior_packetsWiFi represents the behavior of sending packets over a WiFi. As a result, we obtain a set of features containing 20 runtime behaviors. Each APP can be represented by a 20-dimensional vector $[Runbehavior]_{1\times 20}$, and each dimension corresponds to a runtime behavior.

3.2.3 The Integrated Feature

After the feature extraction of the above 2 sections, three feature vectors of three kinds of features are formed, each APP can obtain a set of permission feature vector $[Per]_{1\times 45}$, a set of API calls feature vector $[API]_{1\times 22}$, a set of runtime behaviors feature vector $[Runbehavior]_{1\times 20}$. Combining these three feature vectors sets, each APP can be represented by an 87-dimensional hybrid feature vector $[Pre, API, Runbehavior]_{1\times 87}$. Each feature in the hybrid feature vectors is binary, indicating that if an APP contains this feature, the value of the feature is 1, and if not, the value is 0. The combination of the hybrid feature vectors can better representation the characteristics of the application to distinguish between malware and benign software, and further improve the detection accuracy.

3.3 Feature Selection

Feature selection is an important process of data preprocessing. In the feature extraction of this paper, a greater number of features are extracted, but some of which have no effect on the results of classification. In order to improve the efficiency and accuracy of the classifier, it is necessary to remove the features which have no effect on the classification. At the same time, too many irrelevant features have an influence on the effect of classification. This paper assumes that the initial feature set contains all the important information.

In this paper, we use the filter feature selection algorithm to select the data sets firstly, and then training the classifier. The feature selection process is independent of the subsequent classifier. Kira et al. [14] proposed Relief is a highly efficient filter feature selection algorithm, which designs a "relevant statistic vector" to measure the importance of features. The algorithm is mainly aimed at solving two classification problems. The key of the Relief is how to determine the value of the "relevant statistic vector". Assume that the training set D is $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, for each sample x_i, its feature j corresponds to the relevant statistic vector is as follows:

$$\delta^j = \sum_i -diff(x_i^j, x_{i,nh}^j)^2 + diff(x_i^j, x_{i,nm}^j)^2 \tag{1}$$

Where the greater the value of the formula (1) is, the stronger the classification ability of the feature is. From the formula (1), the evaluation value of each feature is obtained, and the relevant statistic vector component of the feature is obtained by averaging the evaluation value of all the samples to the same feature, the greater the vector component value, the stronger the classification ability.

3.4 The Machine Learning Classifier

Android malware detection belongs to the two-classification problems, and we choose use different classifier algorithms to detect malicious software. In this paper, the following five classifier algorithms are used include Support Vector Machine (SVM) [17], k-Nearest Neighbor [18], Naive Bayes [19], Decision Tree (J48) [20] and Random Forest [21]. Among them, the J48 decision tree algorithm we used in our experiment is the implementation of C4.5 algorithm in WEKA [22]. Selecting different classifier algorithms brings different detection effects, so it is very important to select the appropriate classifier algorithm. The comparison and analysis of different classifier algorithms is a key point in this paper.

4 Experiments and Result

In this section, we use the machine learning tool WEKA [22] to train the classification model for the features obtained from the experimental samples. All experiments were carried out on a computer with a CPU of 3.20 GHz Intel (R) Core (TM) i5 and 8 GB of memory. We collected a total of 359 malicious apps and 500 benign apps as experimental samples. The malicious samples were derived from third-party sample collection platform VirusShare [23]. The Benign samples used in this paper are mainly downloaded from the Google Play store to ensure the availability of the experimental data. In this experiment, we randomly selected 150 malicious apps and 150 benign apps from the experimental samples, and then mixed them together as a training set. Similarly, we get a test set in the same way. The following experiments are carried out on these two data sets.

4.1 Performance Metrics

The following three performance measures that calculate and evaluate the performance of a classification algorithm.

$$True\ positive\ rate = Recall = \frac{TP}{TP + FN} \tag{2}$$

$$False\ positive\ rate = \frac{FP}{TN + FP} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

where True positive rate (TPR), or Recall rate, is the proportion of malware being correctly predicted by the classifier. False positive rate (FPR), is the proportion of malware being incorrectly predicted by the classifier as benign software. Accuracy is the proportion of all samples being correctly classified to all samples, which is used to measure system errors, and the larger the value, the smaller the system error. The higher the value of Accuracy and TPR, the lower the value of FPR, the better the classification effect.

4.2 Experiment Results

In Table 3, we firstly list the classification results for five different classifiers when using only static methods to extract features (i.e., Permission, API calls). The results show that the Random Forest algorithm has the highest accuracy rate and the accuracy rate reaches 92.07%, and its classification effect is the best. In contrast, Table 3 also shows the classification results of five different classifier algorithms when the feature extraction uses the hybrid analysis method (i.e., Permission, API calls, Runtime behavior). As we can see from Table 3, the performance of the Random Forest algorithm is still the best, with an accuracy rate of 94.89% and 2.81% higher accuracy rate than using only static methods. The classification accuracy of SVM algorithm is improved by nearly 2.4%, which achieves 91.27%.

Figure 2 shows more visual and intuitive the classification effects of different classifier algorithms in Android malware detection. We can clearly see that the Random Forest algorithm has the best classification effect, followed by the SVM algorithm. All in all, the experimental results show that the feature extraction method of hybrid

Table 3. Classification results from only static methods, and hybrid methods

Classifier model	Measure metrics (%)					
	Only static methods			Static & dynamic methods		
	TPR	FPR	Accuracy	TPR	FPR	Accuracy
SVM	92.47	9.74	91.27	95.17	7.74	93.66
J48	91.97	14.72	88.19	93.38	14.02	89.34
Naive Bayes	93.49	19.77	85.74	89.39	19.05	84.52
KNN	90.01	19.41	84.56	92.31	17.65	86.71
Random Forest	92.57	8.55	92.07	95.30	5.30	94.89

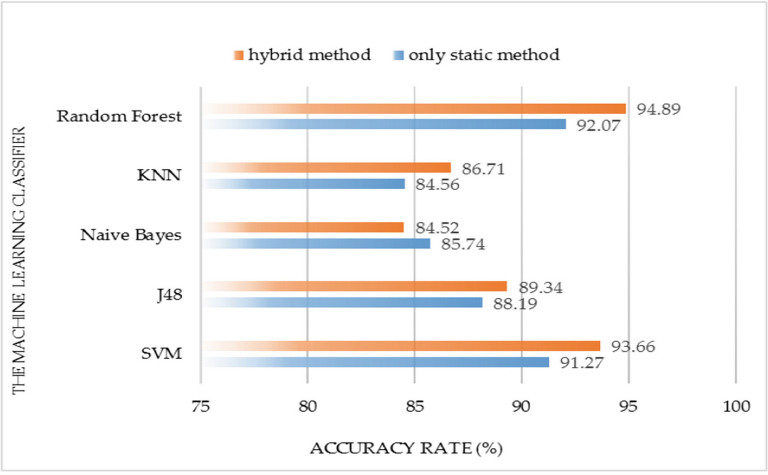

Fig. 2. Accuracy analysis: Using different classifiers

analysis can improve the accuracy of classification results in Android malware detection. At the same time, through the comparison of the performance of five different classifiers, we can know that the Random Forest algorithm has the best detection effect.

5 Conclusions and Future Works

In this paper, we propose a hybrid features analysis method for detection of Android malwares by extracting permissions, API calls and runtime behavior as feature set. We validate the method proposed in this paper through simulation experiments. The experimental results show that this method can effectively detect and classify Android malware, and obtain higher detection rate. Generally speaking, due to the diversity of malicious behavior in malicious applications, the features extracted by the hybrid analysis method can more comprehensively and effectively show the characteristics of Android applications. We demonstrate that the hybrid analysis method combined with static and dynamic methods can improve the accuracy of Android malware detection compared to single static feature extraction methods. Additionally, after reducing the dimension of the extracted hybrid feature vectors, some useless features are removed, which makes the classification accuracy become higher and achieves better detection effect. Finally, we choose the different classification algorithm to bring the classification effect is different, and we find through the analysis that the Random Forest algorithm and SVM algorithm are higher accuracy rate.

For future work, we consider the approach of semantics learning into feature extraction to analyze the behavior of malware. In this way, we can further mine the association rules between features select better feature selection algorithms to reduce the redundancy of features, and further improve the efficiency of classification.

Acknowledgments. This work has partially been sponsored by the National Science Foundation of China (No. 61572355) and Tianjin Research Program of Application Foundation and Advanced Technology under grant No. 15JCYBJC15700, and Fundamental Research of Xinjiang Corps under grant No. 2016AC015.

References

1. Malhotra, A., Bajaj, K.: A survey on various malware detection techniques on mobile platform. Int. J. Comput. Appl. **139**(5), 15–20 (2016)
2. Symantec: Internet Security Threat Report 2017. https://www.symantec.com/security-center/threat-report
3. Tan, D.J., Chua, T.W., Thing, V.L.: Securing Android: a survey, taxonomy, and challenges. ACM Comput. Surv. (CSUR) **47**(4), 58 (2015)
4. Shabtai, A., Moskovitch, R., Elovici, Y.: 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 (2009)
5. Tam, K., Khan, S.J., Fattori, A.: CopperDroid: automatic reconstruction of Android malware behaviors. In: NDSS (2015)

6. Chan, P.P., Song, W.K.: Static detection of Android malware by using permissions and API calls. In: 2014 International Conference on Machine Learning and Cybernetics (ICMLC), vol. 1, pp. 82–87. IEEE (2014)
7. Arp, D., Spreitzenbarth, M., Hubner, M.: DREBIN: effective and explainable detection of Android malware in your pocket. In: NDSS (2014)
8. Wu, D.J., Mao, C.H., Wei, T.E.: Droidmat: Android malware detection through manifest and API calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security (Asia JCIS), pp. 62–69. IEEE (2012)
9. Amos, B., Turner, H., White, J.: Applying machine learning classifiers to dynamic Android malware detection at scale. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 1666–1671. IEEE (2013)
10. Dash, S.K., Suarez-Tangil, G., Khan, S.: Droidscribe: classifying Android malware based on runtime behavior. In: 2016 IEEE Security and Privacy Workshops (SPW), pp. 252–261. IEEE (2016)
11. Rieck, K., Trinius, P., Willems, C.: Automatic analysis of malware behavior using machine learning. J. Comput. Secur. **19**(4), 639–668 (2011)
12. Androguard. https://code.google.com/archive/p/androguard
13. Baksmali. https://github.com/JesusFreke/smali
14. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Comput. Electr. Eng. **40**(1), 16–28 (2014)
15. Monkey. https://developer.android.com/studio/test/monkey.html
16. DroidBox: An Android Application Sandbox for Dynamic Analysis. http://code.google.com/p/droidbox
17. Gu, B., Sheng, V.S., Wang, Z.: Incremental learning for ν-support vector regression. Neural Netw. **67**, 140–150 (2015)
18. Liao, Y., Vemuri, V.R.: Use of k-nearest neighbor classifier for intrusion detection. Comput. Secur. **21**(5), 439–448 (2002)
19. Buntine, W.: Learning classification rules using Bayes. In: Proceedings of the Sixth International Workshop on Machine Learning, pp. 94–98 (2016)
20. Bhargava, N., Sharma, G., Bhargava, R.: Decision tree analysis on J48 algorithm for data mining. Proc. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **3**(6) (2013)
21. Chutia, D., Bhattacharyya, D.K., Sarma, J.: An effective ensemble classification framework using random forests and a correlation based feature selection technique. Trans. GIS **21**(6), 1165–1178 (2017)
22. Hall, M., Frank, E., Holmes, G.: The WEKA data mining software: an update. ACM SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
23. VirusShare Malware dataset. https://virusshare.com