Android Malware Detection Based on Sensitive Permissions and APIs

Chunhui Zhao[1,2](✉), Chundong Wang[1,2], and Wenbai Zheng[1,2]

[1] Key Laboratory of Computer Vision and System, Ministry of Education, Tianjin University of Technology, Tianjin 300384, China
574878671@qq.com

[2] Tianjin Key Laboratory of Intelligence Computing and Novel Software Technology, Ministry of Education, Tianjin University of Technology, Tianjin 300384, China

Abstract. With the widespread use of the Android operating system, the number of applications based on the Android platform is growing. How to effectively identify malware is critical to the security of phones. This paper proposes an Android malware detection method based on the combination of sensitive permissions and API features. This method extracts the permission features and API features by decompiling the APK file, and then uses the mutual information to select sensitive permissions and APIs as feature sets. On this basis, an ensemble learning model based on decision tree classifier and KNN classifier is used to quickly and accurately detect unknown APKs. The experimental results show that the discriminative accuracy of the proposed method is higher than that of the permission set or the API set alone, and the accuracy rate can reach up to 95.5%.

Keywords: Permissions and APIs · Android malware detection · Mutual information · Ensemble learning algorithm

1 Introduction

With the continuous development of mobile phone hardware performance, smart phones have become more and more popular in people's daily life, and the corresponding Android applications are also growing. Since Android apps can earn revenue through advertising, etc., many independent developers may pursue benefits and steal the privacy, property, etc. of the downloaded users, which results in the creation of a large number of malicious applications. In addition to the Google Play market, there are many other third-party application download

Our work is supported by NSFC: The United Foundation of General Technology and Fundamental Research (No. U1536122), the General Project of Tianjin Municipal Science and Technology Commission under Grant (No. 15JCYBJC15600), and the Major Project of Tianjin Municipal Science and Technology Commission under Grant (No. 15ZXDSGX00030).

platforms on the market [1]. However, the supervision of relevant departments is often limited, resulting in some malicious applications flowing into the market, and the crazy spreading of people's downloads has caused huge losses to people. According to the 2017 Android malware special report released by 360 Fire Lab [2], the total number of malware samples on the Android platform intercepted in 2017 was 7.573 million, an average of 21,000 per day. Therefore, how to effectively detect malicious applications from a large number of programs to protect the security and interests of Android mobile phone users is a necessary and urgent challenge for researchers.

The paper proposes an Android malicious application detection method based on sensitive permissions and APIs. The method is mainly divided into two parts: training phase and detection phase. In the training phase, the permissions and API features in the APK file are extracted in batches by decompilation, and then use the mutual information model to generate feature sets with 10 sensitive permissions and 20 sensitive APIs ranking from high to low. The results of the two classifiers of the tree classifier and the KNN classifier are linearly correlated to generate the final result, which is used as an ensemble learning model. In the detection phase, using the above set learning model to quickly classify a large number of APKs. Experiments show that the detection method that combines sensitive permissions and APIs is more accurate than the detection by using permission or API alone.

2 Related Work

Malware detection and classification are challenging problems, especially on mobile platforms. Researchers have made great efforts to address these problems in various ways. In this section, the previous work addressing malware problems is discussed.

2.1 Dynamic Analysis Approaches

Android application dynamic analysis is to trace the relevant memory, such as register contents, function execution results, memory usage, etc., to analyze function functions, clarify code logic, and mine possible loopholes in the case of running code. The advantage of the behavior-based detection method is that it can handle the obfuscated encryption of the code very well. Many researchers have deeply analyzed the application from a dynamic perspective. Cai and Chen [3] proposed that there are some unique advantages of behavior-based dynamic detection techniques. The feature databases are small and do not require frequent updates. TaintDroid [4] identified sensitive information at a taint source, and tracked, dynamically, the impact of labeled data to other data that might leak the original sensitive information. The impacted data were identified before they left the system at a taint sink. DroidScope [5] collected detailed native and Dalvik instruction traces to track information leakage through both Java and native components. These dynamic methods all aim to conduct taint analysis to detect suspicious behaviors during runtime. However, behavior-based detection technology needs to be monitored in real time during the running of the program. It requires high automation and real-time, and requires more time and memory resources.

2.2 Static Analysis Approaches

Static code analysis is an analysis of code correctness and compliance that can be performed without executing a program. The advantage of static code analysis is the high speed of detection. Because there is no need to run, the detection speed is fast. Apposcopy [6] proposed a high-level language to capture the signatures describing semantic characteristics of malware families. Based on the extracted signatures, a static analysis was conducted to detect certain malware families. The literature [7] proposed a model based on API calls, and used the permissions available in various Android applications to capture the functions related to malware behavior, but there is a problem of high false positive rate in this model, which needs to be solved in the future. Literature [8] proposed a tool called Stowaway. With the help of this tool, people can identify whether the programmer has excessive permission to apply for permission during the development of Apps, because of this seemingly inconspicuous behavior there will be many security risks for the application. Wang et al. [9] considered each permission as a feature to establish a feature vector and distinguish between malicious programs and normal programs through classification algorithms. However, there are limitations to only having permissions as features, because this does not fully describe the characteristics of malware.

3 System Design and Implementation

The malware detection process framework designed in this paper is shown in Fig. 1. The main process is divided into two parts: feature generation and integrated learning model.

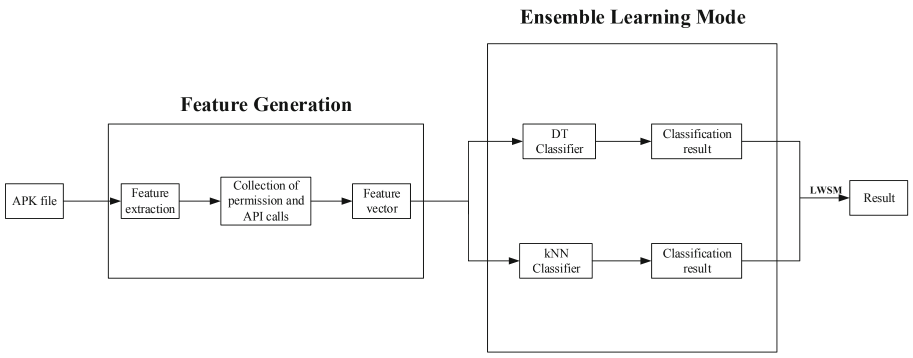

Fig. 1. Flow chart of malware detection

3.1 Feature Generation

The feature generation phase mainly includes three modules: feature extraction, generation permission application and collection of API calls, and feature vector generation.

Feature extraction: APK is the Android application package file, which is an application installation file format on the Android operating system. Structurally, APK is a file based on the zip file format, which is similar to the way jar files are constructed [10]. The permission information and API information used in this article are respectively stored in the manifest.xml file and the smali file. This article uses the python script file provided by the Androguard tool to decompile static analysis. For this purpose, a python script program is written to extract and output the application permissions and API calls information in the APK file to the specified file in batches in order to complete the first step of the experiment to extract features.

Generating a collection of permission requests and API calls: through the above, the permissions and API information of the application have been collected, but there are tens of thousands of known permissions and APIs, and malicious applications and benign applications are different in the permission applications and API calls. The extracted information features need to be filtered. This paper uses the mutual information method to filter out the top 10 sensitive permissions and the 20 top sensitive APIs (see Table 1). Mutual information can measure the relevance of specific permissions, APIs, with applications. A collection of permissions and APIs are choosed based on the relevance. The formula for mutual information is as follows:

$$I(X,Y) = \sum_{x_i} \sum_{y_j} p(X = x_i, Y = y_j) \times log \frac{p(X = x_i, Y = y_j)}{p(X = x_i) \times p(Y = y_j)} \tag{1}$$

Among them, the variable X indicates whether the permission or API appears in an application, the variable Y represents the category of the application (belonging to normal software or malware), and $p(X = x_i)$ indicates the probability that the variable X is x_i, $p(Y = y_j)$ represents the probability that the value of the variable Y is y_j. According to the mutual information formula, the correlation value $I(X,Y)$ of each permission or API with software is obtained. The value ranges from 0 to 1. The larger the value, the higher the correlation between the two. The value of 0 means there is no correlation between the two, and the value of 1 means there must be a correlation.

Feature vector generation: for each application, create a 30-dimensional vector $[feature]_{1*30}$, including the 10 permission features and 20 API features previously filtered. Then this vector is uniformly formatted, that is, processed into CSV format. If the feature of the corresponding dimension of the vector appears in the file output in the feature extraction step, the dimension is set to 1, otherwise it is set to 0.

Table 1. The 10 most sensitive permissions and 20 most sensitive API calls

Permissions		API calls			
1	READ_SMS	1	sendMultipartTextMessage()	11	getSimOperator()
2	SEND_SMS	2	getNETWORKCountryIso()	12	getAccountsByType()
3	READ_PHONE STATE	3	openConnection()	13	getDisplayMessageBody()
4	READ_CONTACTS	4	chmod()	14	com.android.contacts()
5	RECEIVE_SMS	5	abortBroadcast()	15	getOutputStream()
6	ACCESS_NETWORK_STATE	6	writeTextMessage()	16	getDeviceId()
7	INTERNET	7	writeExternalStorageState()	17	getInputStream()
8	CALL_PHONE	8	sendTextMessage()	18	startService()
9	WRITE_SMS	9	getLine1Number()	19	getRunningTasks()
10	INSTALL_PACKAGES	10	getLastKnownLocation()	20	updateConfigurationLocked()

3.2 Ensemble Learning Model

The classification algorithm is trained by using the feature vector [11] of the collected samples, and then discriminates the unknown samples. Different classification algorithms are trained and tested for the same batch of samples, and the resulting classification results will be different. Therefore, using the ensemble learning method to classify the training of samples. This study uses the ensemble learning method based on kNN and decision tree to train and classify samples. In the ensemble learning model, a single weak classifier performs training prediction on the sample data, and then combines the prediction results of these weak classifiers to vote for the final prediction result. This approach reduces the variance of the base class classification by introducing randomness into the model building process [12].

Weak classifier: first of all, the kNN classification algorithm is easy to implement and understand, and it has high classification accuracy in the classification algorithm. In addition, KNN is an online technology, new data can be directly added to the data set without retraining, so the kNN classifier is more suitable than other classifiers. The "information gain" approach is used in the attribute selection of the decision tree algorithm, which is much the same as the mutual information method used in this study. Therefore, the decision tree classifier is used.

Weighted voting: better classification results can be obtained by combining multiple individual classifiers into one strong classifier. This paper assigns different weights to KNN and decision tree classifier. Finally, the weight of 0.4 is assigned to KNN, and the weight of 0.6 is assigned to the decision tree. If the detected application is benign, then result-DT and result-kNN are set to 1. Otherwise, set it to 0. In addition, our detection model has a threshold set to 0.5. This is because the probability that an unknown application is considered malicious is theoretically the same as a benign probability. Specifically, using the Linear Weighted Weights Method (LWSM) to calculate the probability that an unknown application is classified as a vicious or benign program. The linear weighted sum calculation is shown in Eq. 2:

$$Result = \frac{1}{2}(R_1 * P_1 + R_2 * P_2) \tag{2}$$

Where R_i represents the result of the classifier, its value is 0 or 1, which means that the application is a malicious application or a benign application. P_i represents the weight of the two classifiers, and $P_1 + P_2 = 1$. Based on this, the following four results are obtained:

- when result-DT = 1 & result-kNN = 1:
  Result = $\frac{1}{2}(1 * 0.6 + 1 * 0.4) = 0.5$
- when result-DT = 1 & result-kNN = 0:
  Result = $\frac{1}{2}(1 * 0.6 + 0 * 0.4) < 0.5$
- when result-DT = 0 & result-kNN = 1:
  Result = $\frac{1}{2}(0 * 0.6 + 1 * 0.4) < 0.5$
- when result-DT = 0 & result-kNN = 0:
  Result = $\frac{1}{2}(0 * 0.6 + 0 * 0.4) < 0.5$

If the result is equal to 0.5, the application will be judged as a benign application. Otherwise, it will be a malicious application.

4 Experimental Results and Analysis

4.1 Experimental Environment

In this experiment, 2474 normal applications and 3526 malicious applications are selected as experimental data sets. Among them, the normal Android applications are collected from third-party application market and Google Android Market [13] by using web crawler programs, and the malicious applications are provided by the malicious sample set of the virusShare.com [14]. The experimental environment is: operating system of Windows 10, processor: Intel Core i5, 4 GB of memory, Python 2.7 scripting languages.

4.2 Results and Analysis

In order to evaluate the detection model, testing a large number of samples for experiments and conducted multiple sets of comparative experiments to demonstrate the superior performance of proposed test models. The results of the experiment are evaluated by TPR, FPR, Precision and Accuracy.

Overall performance: as shown in Fig. 2, experiments are performed on 5,000 samples by selecting different features. The distribution ratio of test set and training set is 1:3. As can be seen from the figure, the overall detection accuracy of the model can reach 95.5%. Our proposed method of combining permissions and API as a feature has a better classification effect than the detection of a feature alone. It can be clearly seen that the single feature detection in detection

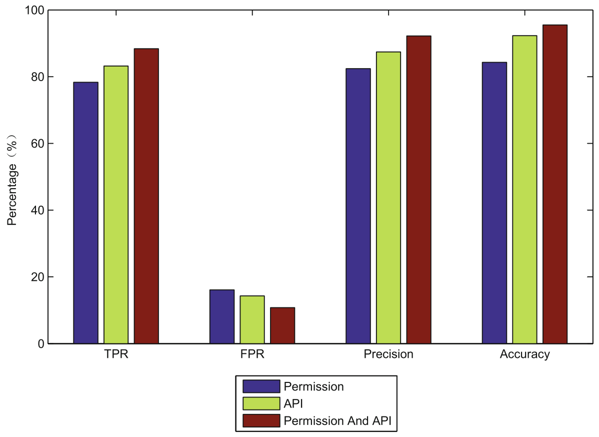

Fig. 2. Overall performance

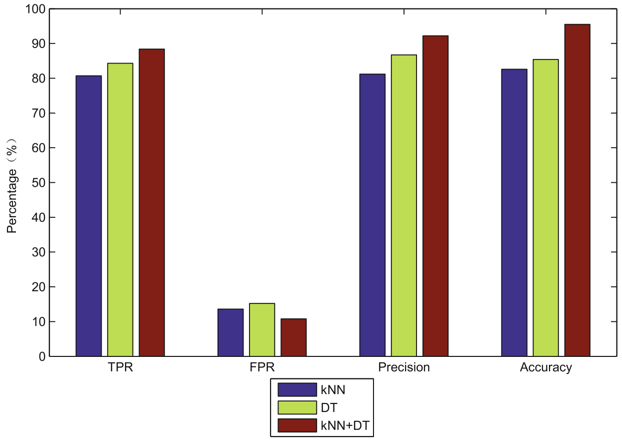

Fig. 3. Ensemble learning model performance

accuracy is lower than the permission and API combination detection. And our method has a significant reduction in the false positive rate, but the false positive rate is still high, which is a problem needed to be solved in the future. In short, the malware detection method proposed based on sensitive permissions and API has better detection results.

Ensemble learning model effect: the ensemble learning model is evaluated by comparing the combined classification method of k-nearest neighbors and decision trees with the classification method using only one of them. The effect diagram is shown in Fig. 3. Through Fig. 3, all aspects of using the ensemble learning model are better than using any of the classifiers alone, so using the ensemble learning model to detect and classify is effective.

Ensemble learning model weight selection: the ensemble learning model needs to assign a weighting ratio to the classification results of each classifier to obtain

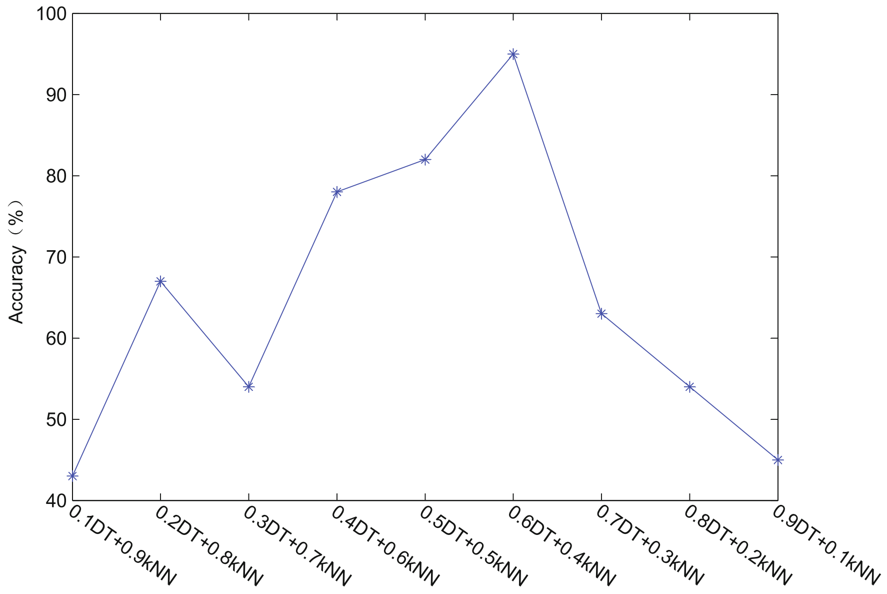

Fig. 4. The effect of weight on accuracy

the final classification result. Different weights have a significant impact on the final classification result. For this reason, selecting the test weighting interval to be 0.1 to obtain the detection accuracy of different weight ratios as shown in Fig. 4. It can be seen from the figure that different weight selection has a great influence on the accuracy of the whole model. According to the figure, the final choice is to assign a weight of 0.6 to the decision tree classifier and assign 0.4 to the kNN classifier. This is also because the kNN algorithm itself is relatively simple, and it has a certain influence on the experimental results when facing the unbalanced data set.

5 Conclusion

Starting from the two characteristics of the Android application's permissions and API, we collect the feature set of the application software sample through decompilation, and then extract the high-risk API and permission features using the mutual information model, and combine the two to generate the permission-API feature. The vector is then used to implement classification detection of Android malicious applications through an ensemble learning model. The simulation experiments on 3526 malicious applications and 2474 normal applications show that the proposed method has good effects on accuracy and true positive rate. This method can effectively improve the accuracy of Android malicious application detection, more comprehensively reflect the characteristics of Android applications, but the false positive rate of this method is still slightly higher, and we need further improvement in subsequent research.

References

1. Wu, D.J., Mao, C.H., Lee, H.M., Wu, K.P.: DroidMat: Android malware detection through manifest and API calls tracing. In: 7th Asia Joint Conference on Information Security, Tokyo, Japan, pp. 62–69 (2012)
2. 360 Campfire Lab: 2017 Android malware special report. http://blogs.360.cn/post/review_android_malware_of_2017-2.html. Accessed 3 Jan 2018
3. Cai, L., Chen, T.: Research review and outlook on Android mobile malware detection. In: Netinfo Security 2016, vol. 9, pp. 218–222 (2016)
4. Enck, W., et al.: TaintDroid: an information-flow tracking system for real time privacy monitoring on smartphones. ACM Trans. Comput. Syst **32**(2), 5 (2014)
5. Yan, L.K., Yin, H.: Droidscope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis. In: Proceedings of 21st USENIX Security Symposium, pp. 569–584 (2012)
6. Feng, Y., Anand, S., Dillig, I., Aiken, A.: Apposcopy: semantics-based detection of Android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 576–587 (2014)
7. Sharma, A., Dash, S.K.: Mining API calls and permissions for Android malware detection. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 191–205. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_13
8. Felt, A.P., Chin, E., Hanna, S., et al.: Android permissions demystified. In: Proceedings of 18th ACM Conference on Computer and Communications Security, pp. 627–638 (2011)
9. Wang, W., Wang, X., Feng, D.W., et al.: Exploring permission-induced risk in Android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**(11), 1869–1882 (2014)
10. APK. http://zh.wikipedia.org/wiki/APK
11. Xiang, C., Yang, P., Tian, C., Liu, Y.: Calibrate without calibrating: an iterative approach in participatory sensing network. IEEE Trans. Parallel Distrib. Syst. **26**(2), 351–356 (2015)
12. Yang, Z., Wu, C., Zhou, Z., Zhang, X., Wang, X., Liu, Y.: Mobility increases localizability: a survey on wireless indoor localization using inertial sensors. ACM Comput. Surv. **47**(3), 1–34 (2015)
13. Google Android Market. http://play.google.com/store/apps?feature=corpusselector. Accessed 30 Jan 2017
14. Virusshare. http://virusshare.com. Accessed 30 Sept 2017