Journal of Computers Vol. 29 No. 6, 2018, pp. 243-253
doi:10.3966/199115992018122906024

Android Malware Detection Based on Static Behavior Feature Analysis

Chen Chen[1], Yun Liu[1*], Bo Shen[1], Jun-Jun Cheng[2]

[1] Department of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing, China
{15120201, liuyun, bshen}@bjtu.edu.cn

[2] China Information Technology Security Evaluation Center, Beijing, China
chengjj@itsec.gov.cn

Received 9 January 2018; Revised 22 January 2018; Accepted 22 January 2018

Abstract. As an open source operating system, Android has groups of users and developers, which result in a great deal of Android application in app markets. But at the same time, app's benign and malicious behavior cannot be screened, the number of new malware sample for Android platform and infected mobile phone are still soaring, Android malware detection is faced with a tough challenge. Thus this paper proposes an approach that analyze Android malware by extracting app's static behavior, and information are captured include the use of permission, android's components and API calls. Then we take the advantage of PCA to extract app's principle behavior features. Finally, classifiers are trained by Android app dataset to validate the performance. Experiments show that the proposed method is capable of 97.4% true positive rate and 99.8% area under ROC, our lightweight classifiers can detect Android's unknown malware effectively and accurately.

Keywords: Android, feature extraction, feature selection, malware, static behavior

1 Introduction

Figures released by International Data Corporation (IDC) show that smartphone companies shipped a total of 1.47 billion smartphones worldwide by 2016, Android dominated the smartphone market with a share of 86.8%, moreover, it is expected to continue rising in 2017 [1]. With the popularity of smartphone, the malicious software in desktop computer began to spread over mobile devices. 360 Internet security center pointed out in the report that 14.033 million new malicious Android app samples were intercepted by their researchers in 2016. The report also indicated, according to the intercepting record, the accumulative total of malicious attacks have reached over 253 million times in a year, which means an average of nearly 700 thousand infection each day [2]. A variety of third-party Android app stores are emerging one after another and have no guarantee for the security of users. The growth rate of Android malicious app is being accelerated, the diversity and complexity is being enhanced, thus making the security of Android smartphone in a worse state. There are grave challenges facing the management of Android app store and the detection of Android malware. In order to tackle the constantly changing and continuously updated security threats, it is thus important to detect Android malware effectively and classify Android app correctly.

The paper is organized as follows: Section 2 gives background information on the methods and analyses of Android malware detection. Next, section 3 illustrates our static behavior analysis system and malware detection model, also explains the machine learning techniques and main ideas behind our analysis. Section 4 presents experiments, results and analyses. We then conclude the paper with a brief discussion and outline future work.

[*] Corresponding Author

2 Related Work

There are a variety of Android malicious apps which can be changed and updated rapidly. ANVA classify mobile threats into the eight classes of charges consumption, privacy theft, malicious cost, fraud, scamp, system destroy, remote control and malicious propagation [3]. Overall, Android malware has been formed complete gray industrial chain, efficient and automatic analysis method have to be approached to fight against these threats.

Methods of Android malware detection can be categorized into two classes: signature-based and behavior-based methods. The signature-based methods on Android malware are a malicious code matching by extracting signatures from Android app. Signature-based methods have the advantage of quick, accurate and low false positive rate, whereas they cannot detect the unknown signature, and are relatively backward. Due to the increasing and changing of malware, malicious code signature database needs to be updated frequently. The behavior-based detection methods match the apps' behavior pattern with known malicious'. Unknown malware can be detected through these methods, but with high false positive rate. Behavior-based have two categories: static analysis, by using reverse engineering to extract local feature; and dynamic analysis, by running the application on virtual machine or mobile phone to monitor and intercept the app's behavior on runtime. In contrast to dynamic analysis, static extends to app's every execution step without specific execution process restriction, dynamic can only detect app's behavior accord with specific pattern. Meanwhile, dynamic has high real-time requirement, needs to continuously monitor the runtime state, has an effect on mobile phone's performance and takes time and resources.

Kirin is a representative permission-based static analysis of early security service for Android, it defines malicious behaviors by comparing application's permission requirement with static, conjunctive rule set, which consist of common dangerous permissions using by malware. Since the study's small number of rules are not representative, the detection rate is low. Moreover it cannot identify collusion attack that carried out by using a combination of multiple malicious or infected apps [4]. In respect to the issue of malicious code flooding, Hu et al. proposes a novel statistical analysis system that combine static and dynamic analysis technologies. Then the malicious code detection system is released online and free to use [5]. But dynamic analysis will greatly increase cost, degrade performance, the detecting speed and efficiency of this system cannot meet the social Android malware detection demand. Moreover, the accuracy will decrease with unknown malware.

With the rise of machine learning, a large number of scholars attempt to get a new Android malware detection methodology through a combination of machine learning algorithms and static analysis method, make the detecting target transition from known malware to unknown malware. By analyzing app's static behavior, the classifier formed by the machine learning can identify the distinction between benign and malicious applications more automatically and efficiently than traditional classification.

Glodek et al. extract permissions, components, native code as training features for random decision forest classification of malicious and benign applications [6]. Zhou et al. improve the Aswini by extracting permissions features when the privilege escalation vulnerability is considered, then they train and test samples by five different machine learning algorithms. Although the accuracy of malware detection is improved, the complexity is increased too [7]. After extract a number of permission and control flow graph features, Sahs et al. train a One-Class Support Vector Machine with benign Android app and test their system against a collection of benign and malicious apps with different kernels, but the result is not ideal [8]. Yerima et al. apply Bayesian classification with API (Application Programming Interface) call, permission and system command features extracted through reverse engineering of the Android apps, results of their experiments with different feature sets and training samples show that this model significantly improve the detection rate compare to those signature-based detection model [9].

For Android malware detection, common machine learning algorithms include Decision Tree, Random Forest, K-means, Logical Regression, Adaboost, Naïve Bayes, Support Vector Machine, etc. And general feature extraction algorithms are Principal component Analysis, Linear Discriminant Analysis and Independent Component Analysis, feature selection is broadly separated into filter method and wrapper method. No one is completely certain which machine learning algorithm to use, which feature to extract and how to extract. In this paper, we apply a static behavior analysis system for Android app and use feature extraction and selection to get a feature set which is composed of the use of permission, android's components and API calls in java code information, then use different machine learning algorithms to construct classifiers. The dimension of our final feature sets are quit low but those classifiers are capable of high detection rate.

3 Model Construction

Through the investigation of large quantities of malicious application samples, researchers concluded Android static behaviors and gave us a lot of constructive suggestions. Based on attack types, Arslan et al. divide mobile threats into four classes: Malicious Applications (Riskware, Adware, Malware), Trojan, derivatives and working structure, Viruses and Worms. And there is a summary of each application's attack, development, detection, and prevention methodology [10]. 360 Internet security center also indicates that new malware on Android platform are mainly charges consumption apps, accounts for 74.2% in 2016, 67.4% of privacy theft apps will steal SMS information, 34.8% even steal mobile banking information and 10.0% steal phone's contacts.

The report on Chinese Android mobile phone's privacy and security in 2016, published by DCCI, rate the privacy permission on a four-point scale for permission crossing the line [11]: High-risk permission: root permission.

- Core privacy permission: access to user's contacts data, SMS data, call log, location information, and phone number.
- Crucial privacy permission, send SMS messages, make calls, monitor ongoing calls, record audio, access the camera device and mobile network,
- Normal privacy permission, get all application packages that are installed on the device, change Wi-Fi and Bluetooth connectivity state, access to phone state.

The static code analysis for classifying android applications applied by Shabtai et al. is based on Android's Java byte-code. Features are extracted from game and tool apps' apk, XML, and dex files for the classification, which give us some great tips on Android app's function and API calls [12].

In this paper, we apply reverse engineering on Android malware detection to get app's source code and resource file, and then get the usage of permission, android's components, API calls information and so on. Through the static analysis and observation of lots of malware samples, app's features are captured for training the classifiers. Next, measure the correlation between of classes and features and remove the frivolous behavior feature. Finally, employ various machine learning algorithms for the classification of apps' static behavior to detect the Android malware. To implement feature engineering and machine learning work, we take full advantage of the data mining tool WEKA [13].

3.1 Static Behavior Analysis

Each Android application contains an AndroidManifest.xml at the root of the project hierarchy, and this file provides information that Android operating system needs in order to run the app. It describes the permissions that the app itself must have in order to access protected parts of the API and interact with other apps, permissions that other apps are required to have in order to interact with the app's components. It also declares the components of the app, which include the activities, services, broadcast receivers, and content providers [14]. Source code file helps us detect common charges consumption, privacy theft, malicious cost and so on behaviors in Android app form the coding point of view. Then we can set up filtering rules through the API calls of relevant sensitive operations.

Based on the research and analysis above, we designed a system to analysis Android app's static behavior automatically. The steps are outlined in Fig. 1.

First, the app's installed file is decompiled using Apktool [15], resource files are decoded to nearly original forms. By this, there is an AndroidManifest.xml that contains permissions and components information at project's root directory. Second, the Dalvik executable format file can be extracted from APK file, read the Dalvik executable (.dex) file and convert it to class (.class) files with the help of Dex2jar [16]. A class file is a compiled .Java file created by the Java compiler, but it can be decompiled by JAD [17]. Then decompile all class file to java code (.java) file to analysis potential runtime behavior in java code, and look for the relevant function and system API calls based on specific keywords.

Follow the above steps, Android app's static behavior feature can be extracted from those resource and source code files we got.

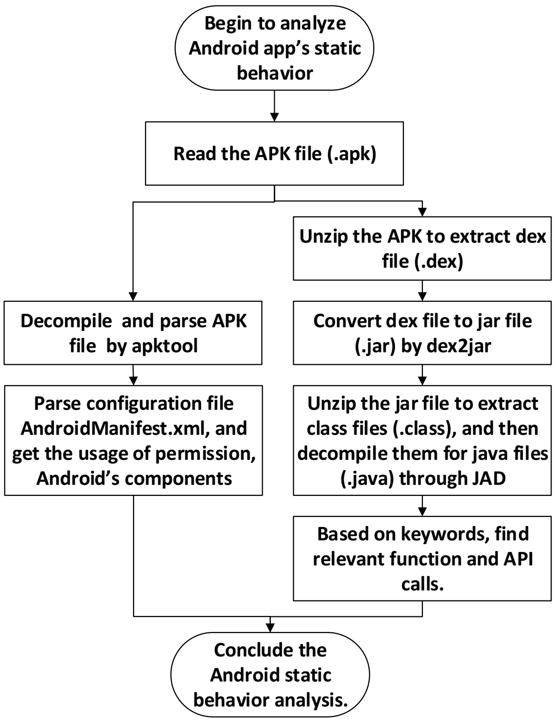
Begin to analyze Android app's static behavior
Read the APK file (.apk)
Unzip the APK to extract dex file (.dex)
Decompile and parse APK file by apktool
Convert dex file to jar file (.jar) by dex2jar
Parse configuration file AndroidManifest.xml, and get the usage of permission, Android's components
Unzip the jar file to extract class files (.class), and then decompile them for java files (.java) through JAD
Based on keywords, find relevant function and API calls.
Conclude the Android static behavior analysis.

Fig. 1. Static behavior analysis system

The test dataset consists of 387 Android apps, 196 benign apps are the most popular apps downloaded from the official Android Market [18-20], and 191 malicious apps were randomly downloaded from a shared database which contains 83822 Android malware samples [21].

3.2 Feature Engineering

Features are raw materials of machine learning system, and they immediately affect the prediction model and classification results. The input data will be too much if we model the static behavior data directly. To better represent practical problems that processed by the prediction model, and have algorithms and models maximum use the features that extracted from raw data, the next section is feature engineering, which extract, prune and combine features in an automatic way.

Feature description. As an important part of Android security, the use of permissions and their importance to static behavior analysis are mentioned earlier, here, the permission information will be extracted for the detection of malicious applications.

Activity is the only "screenful" component, it serves as the entry point for a user's interaction with app, and every screen in an app is an activity by itself. For an application to be able to use activities, the activities and certain of attributes must be declared in the manifest. An Android service is a component that is designed to perform long-running operations in the background, and does not provide a user interface. Broadcast Receivers simply receive and respond to broadcast announcements from other applications or from the system, and it does not provide any UI. Content providers can help an app manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps. It can be seen, as the four components of Android, their use and app's true intentions are closely linked, and play a significant role in the analysis of Android static behavior.

"SmsManager" manages SMS operations such as sending data, text, and pdu SMS messages. "SENT_SMS_ACTION" is for sending an SMS and monitoring the success of its transmission and delivery. "ACTION_DIAL" is invoked when system makes a call, then the built-in dialer will be launched and a UI with the number being dialed will be shown to allow the user to explicitly initiate the call. "android.intent.action.CALL" can implement phone calls via program directly without pop-up or UI. "Settings.System" contains miscellaneous system preferences, allow users to modify app features and behaviors to access individual settings entries in all aspects. Therefore, calls to those five APIs describe app's potential malicious behavior from a static perspective.

To capture app's static behavior through the static behavior analysis system designed by this paper, 387 samples were analyzed based on analyzing methods and perspective discussed above. Then 20 app's permissions, 4 Android components and 5 API calls information were obtained to describe app's static behavior.

As shown in Fig. 2, within 387 apps, only 2 malware don't request INTERNET permission, all the other malware and benign application have the request, and it turns out, accessing to network is the basic needs of achieving a functional app. Beyond this, the request of crucial privacy permission SEND_SMS in malware is far more than benign application, which reveals potential charges consumption, privacy theft, malicious cost and so on threat.

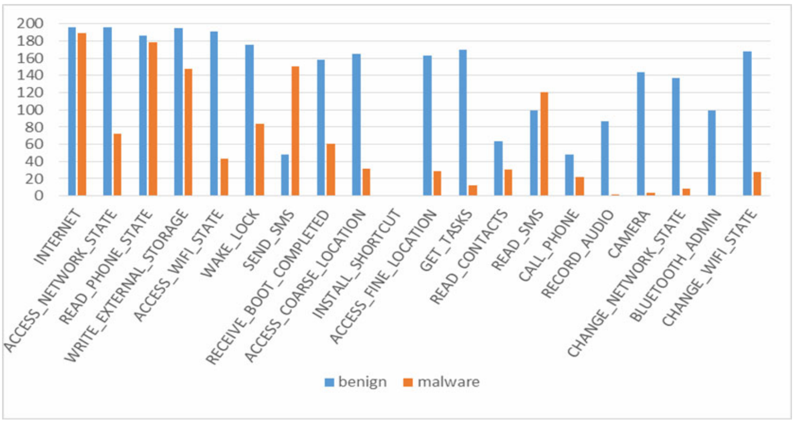

Fig. 2. Permission requirement of malware vs. permission requirement of benign app

Because most of malware has simplex objectives and motivation, there is less request of Android components as part of their design and development. In Fig. 3, even the Activity, only "screenful" component, on average the usage in malware is 3.6% of benign application. And the use of Broadcast Receivers in malware is not significantly lower than benign. When there are malware and benign application in user's mobile simultaneously, it is prone to threaten the data security of components, then the benign app will be used by malware and cause sensitive information leakage.

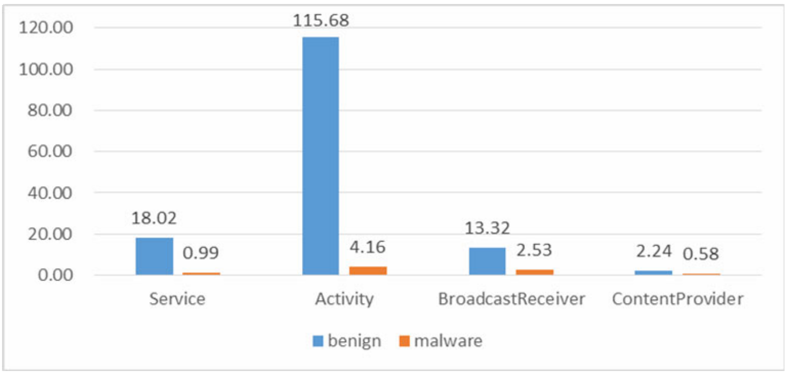

Fig. 3. Component requirement of malware vs. component requirement of benign app

In Fig. 4, we show the average number of API calls in app's source code. The state of the demand for sending, reading and writing SMS in permission analysis, together with this "SmsManager" API calls, give us confidence that most of malware steal user's personal information, send SMS and customize expansive plan. Those threats correspond to malicious behavior, such as intercept user's SMS and forward it to criminals silently, if the SMS for confirming banking trade or other sensitive information, there will be a big risk.

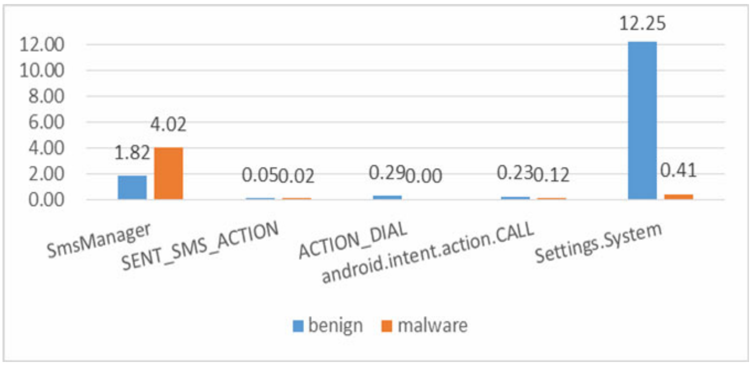

Fig. 4. API requirement of malware vs. API requirement of benign app

Feature extraction. In machine learning, feature extraction is a set of methods that map high-dimensional input features to new low-dimensional output features which are more informative and non-redundant. In order to achieve the purpose of dimensionality reduction, this paper use Principal Component Analysis (PCA), an unsupervised feature selection strategy, to decrease the dimension of feature vector without incurring excessive information loss, and maximize variance of the app data.

A training dataset consists of m app samples, and each app sample is described with respect to n features, equivalently, a $m \times n$ matrix $\mathbf{X}$ is formed. Then the training dataset matrix $\mathbf{X}$ is processed by PCA, and its covariance matrix can be written as

$$\mathbf{C_X} = \frac{1}{m}\mathbf{B^T B} \tag{1}$$

$\mathbf{B}$ for the i-th sample ($\mathbf{B_i}$) is calculated as:

$$\mathbf{B_i} = \mathbf{X_i} - \frac{1}{m}\sum_{i=1}^{m}\mathbf{X_i} \tag{2}$$

where $\mathbf{X_i}$ is the i-th sample app. Then calculate $\lambda_1, \lambda_2, \cdots, \lambda_n \left(\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n\right)$, which is the eigenvalues of covariance matrix $\mathbf{C_X}$, and corresponding eigenvectors $\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_n}$. Cumulative explained variance ratio $\varphi(d)$ associated to each d-th sample is defined as:

$$\varphi(d) = \frac{\sum_{i=1}^{d}\lambda_i}{\sum_{i=1}^{m}\lambda_i} \tag{3}$$

Set cumulative explained variance $\varphi(d) \geq 0.95$, and base on the first d eigenvalues, corresponding eigenvectors are chosen to construct the low-dimension subspace $\mathbf{U_d} = [\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_d}]$, a $m \times d$ matrix. Then project the app sample $\mathbf{X_i}$ in training dataset into $\mathbf{U_d}$,

$$\mathbf{Y_i} = \mathbf{X_i U_d}, i = 1, 2, \cdots, m \tag{4}$$

Finally, app sample $\mathbf{X_i}$ in high-dimension space can be expressed as $\mathbf{Y_i}$ with lower dimension. In PCA, the amount of information are only measured by variance, and not affected by factors outside of the dataset. Besides, each of the primary components is mutually orthogonal, which can eliminate the interaction of raw data.

In this paper, the training dataset consists of 387 app samples, and each app sample is described with respect to 29 features. With feature extraction by PCA, primary components, where cumulative explained variance is larger than 95%, are used for subsequent training, the training dataset matrix is 387×21 now.

Feature selection. Feature selection includes four parts: generation procedure, evaluation function, stopping criterion, and validation procedure. First, generate a feature subset from the complete set. Second, evaluate the feature subset with evaluation function. Third, compare the result with stopping criterion. The whole procedure will stop if the result is better than criterion, otherwise continue to generate another feature subset to select feature. Finally, validate the subset that has been selected.

Generation procedure search feature subset of Android app, the search algorithm is divided into complete, heuristic and random. The best first search, which is a greedy complete algorithm, is chosen for its backtracking. Pick N features with highest score from feature set as subset, add it into an unlimited length priority queue, then take out the subset with highest score from the queue, and enumerate all the feature sets produced by adding a feature to the subset and add them to the queue.

Evaluation function includes filter and wrapper. The first measures the feature subset by analyze the interior characteristics regardless of classification selection. Therefore, a wrapper evaluation method called “WrapperSubsetEval” is used with three subsequent classification algorithms. Then calculate prediction performance of this algorithm on the feature subset, and evaluate the feature subset that provided by generation procedure [22].

The most common stopping criterion are execution time, evaluate times and threshold setting. We set standard deviation to 0.01, if the evaluation procedure exceeds this value, it will stop searching. Because the accuracy doesn’t differentiate classes and treats malware and benign apps equally, it is set as the measure used to evaluate the performance of attributes combinations.

3.3 Machine Learning Algorithm

For Android malware detection, commonly used machine learning algorithms include decision tree, Bayesian network, naïve Bayes, random tree, random forest, support vector machine and so on. In order to obtain a faster classification model with higher detecting efficiency, it is important to choose an appropriate algorithm.

Random forest. Alam [23] and Glodek [6] et al. get good accuracy and classification results by proposing malware detection approaches using Random Forest, give us insights into data processing and parameter selection. After using “WrapperSubsetEval” with Random Forest classification algorithm and 25 iterations, we got a feature set containing 6 attributes for classification analysis.

Random Forest is constructed by a multitude of classification and regression tree (CART), and output the classification result of each sample by taking a vote. When sample the training set with replacement for the current tree, about one-third of the available data are left out of the sample. So every tree in forest is built on the other two-third, and the out-of-bag (OOB) data is left for running internal classification error as trees are added to the forest. Breiman [24] gives experimental results that the OOB error is unbiased.

Support vector machine. On the basis of statistical learning theory, Vapnik Chervonenkis (VC) dimension and structural risk minimization (SRM), nonlinear support vector machine (SVM) maps original finite-dimensional space into a much higher-dimensional space nonlinearly with proper kernel function. By constructing hyperplanes, it allows sample data to be analyzed with linear discriminant function in higher-dimensional space, minimizes the empirical risk and confidence interval simultaneously.

In order to map low-dimensional sample data to a high-dimensional space, the Radial Basis Function (RBF) is used to get an optimal performance because the less parameter requirement and the ability to reduce complexity of the model.

$$\mathrm{k}\left(x, x_i\right) = e^{-\gamma|x-x_i|^2} \tag{5}$$

After test analysis on probability detection, it shows that good statistical regularities can be got with small sample size when $\gamma = 0.06$. And a feature set that contains 8 attributes for classification analysis was got by using “WrapperSubsetEval” with SVM classification algorithm.

Naïve bayes. Naïve Bayes algorithm is widely employed in many fields, it calculates the priori probability of sample’s classes and the conditional probability vector of feature vector for every class based on some features. Then it computes the posterior probability of samples in the test dataset and classifies the sample with maximum posterior probability.

Given an Android app dataset $\mathbf{E}$, $\mathbf{E} = \{X_1, X_2, \cdots, X_n, \mathbf{C}\}$, where $X_1, X_2, \cdots, X_n$ are feature variables, and x_i is the value of X_i, $\mathbf{C}$ is the class variable, $\mathbf{C} = \{c_1, c_2, \cdots, c_m\}$. Assume that each feature is conditionally independent of every other feature, based on Bayes’ theorem, the probability that sample

$\mathbf{I_i} = \{x_1, x_2, \cdots, x_n\}$ belong to c_j is

$$P(c_j \mid x_1, x_2, \cdots, x_n) = P(x_1, x_2, \cdots, x_n \mid c_j) \times P(c_j) \tag{6}$$

and $P(c_j)$ is the prior probability of class c_j, $P(x_1, x_2, \cdots, x_n \mid c_j)$ is the likelihood which is the probability of predictor given class, and $P(c_j \mid x_1, x_2, \cdots, x_n)$ is the posterior probability of class c_j. Then the class c_j that has maximum posterior probability will be chosen for this sample $\mathbf{I_i}$. After combining the "WrapperSubsetEval" and Naïve Bayes classification algorithm without substituting Kernel density estimation for normal distribution, we got a feature set containing 9 attributes for the next classification.

4 Ensemble Machine Learning

In this section, to verify the proposed static behavior analysis system and the Android malware detecting model, we implemented static analysis and feature engineering on the dataset that contains 191 malware and 196 benign Android app. Then the result data was loaded into WEKA to apply machine learning with different classification.

4.1 Experiment Results

In the machine learning process, 10-fold cross-validation was used for the above Random Forest, SVM and Naïve Bayes classification. Android static behavior data was randomly partitioned into 10 equal size subsets, a single subset was retained as the validation data for testing the model, and the remaining 9 subsets were used as training data in turn.

For evaluating the predictive accuracy of classifiers, true positive rate (TPR), false positive rate (FPR) and area under curve (AUC) were used to describe the classifier. TPR is the rate of detecting malware instance as malware correctly, FPR is the rate that classifier classify benign app as malware falsely, AUC is the area under the ROC curve, also the composite indicator of TPR and FPR. The nearer an AUC value approximates to one, the higher its malware detection rate is.

In Fig. 4, we show the direct classifying result with different classification algorithms after implement the Android static behavior analysis without the feature engineering. It can be seen, static feature captured by the proposed static behavior analysis system can describe app samples very well. For algorithms RF, SVM and NB, the value of TPR are 0.974, 0.974 and 0.953 respectively. Apparently, the classifiers of this paper can obtain good classified effect on app dataset, and has high sensitivity and specificity. RF minimizes correlation between each tree and increases the classification accuracy by randomly selecting features at every node to branch. Moreover, the introduction of randomness avoids over fitting problem of the RF and improve the anti-noise ability. SVM can better solve the smaller samples, nonlinear, high dimension, local minimum and so on practical problems on limited training dataset, which optimizes the generalization capability of classifier. NB classifier has high learning and classification efficiency on small-scale dataset.

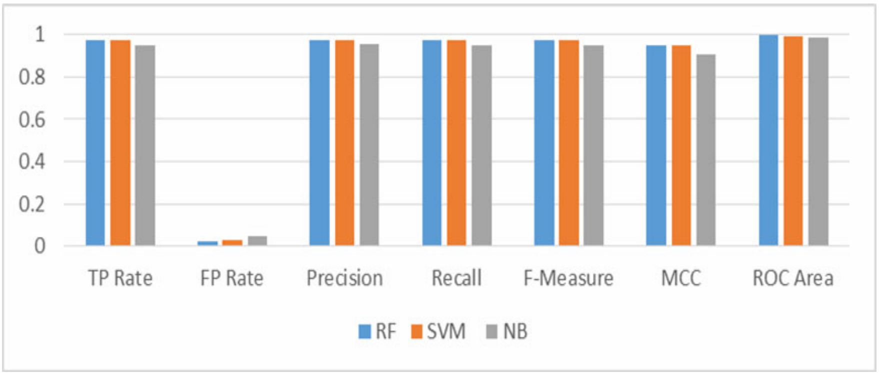

Fig. 5. Classification result without feature project

Then the feature extraction and selection were applied with PCA and "WrapperSubsetEval" to test each classification algorithm. Results are shown in Table 1, where RF0, SVM0 and NB0 are classifiers on raw feature set created by static behavior analysis system, RF1, SVM1 and NB1 are tested on dataset resulted by feature extracting process, RF2, SVM2 and NB2 are trained on dataset resulted by feature engineering.

Table 1. Classification result of different classifiers

Classifier	Dimension	TPR	FPR	Precision	Recall	F-Measure	MCC	AUC
RF0	29	0.974	0.025	0.975	0.974	0.974	0.949	0.998
SVM0	29	0.974	0.026	0.974	0.974	0.974	0.948	0.995
NB0	29	0.953	0.046	0.954	0.953	0.953	0.908	0.988
RF1	21	0.951	0.049	0.951	0.951	0.951	0.902	0.992
SVM1	21	0.964	0.036	0.964	0.964	0.964	0.928	0.994
NB1	21	0.925	0.076	0.926	0.925	0.925	0.851	0.966
RF2	6	0.961	0.039	0.961	0.961	0.961	0.923	0.994
SVM2	8	0.974	0.026	0.974	0.974	0.974	0.948	0.974
NB2	9	0.948	0.052	0.948	0.948	0.948	0.897	0.978

From Table 1 we can see, dataset created by PCA fails to get a good classification results compared to two other datasets. For the results of RF1 and NB1, various indicators are falling apparently. According to our analysis, PCA is an unsupervised learning method, doesn't take into account the categorization information which are not available for PCA in the training data. Besides, noise samples influence the classification effect, but the dimension reducing goal is achieved after feature extraction. Subset that displays a substantial correlation with class and small correlation between each feature is selected from high-dimension feature set automatically by feature selection. Not only the dimension of new subset is decreased significantly, but also the classification of RF2, SVM2 and NB2 are improved obviously. The classification results of SVM2 are elevated to SVM0's, which means we get the same high classification accuracy on dataset with 8 features as 29 features. For NB2, feature engineering makes a negative impact on it because Naïve Bayes algorithm assume each feature is conditionally independent. And RF2 obtains a high classification efficiency on sample dataset in lowest dimensional feature space.

4.2 Experiment Analysis

From the above experiment it can be concluded that the Android static behavior analysis system we proposed can describe features of sample app adequately and effectively. Classification results of feature extraction by PCA and feature selection by Wrapper indicate that too many feature could cause the curse of dimensionality, complicate the model and decrease the accuracy and extend capability. Feature extraction searches feature subsets by testing repeatedly, constructs and evaluates the model automatically to obtain an objective subset with best predication ability. Although feature engineering has an impact on the accuracy of classifier, it greatly reduces the dimension of dataset and improve the efficiency of classification. Whereas the raw dataset without feature extraction and selection has more information to describe sample and has an edge in classification. And classifiers constructed with Random Forest algorithm not only have high TPR but also low FPR, and can detect benign app accurately, barely classify malicious app as benign, it has good sensitivity and specificity.

There are significant performance differences between different classification algorithms. We also tested other classifications in the experiment, but parameters, configurations and usages presented in this paper are the most typical ones and have fine classifying ability, then other algorithms will not enumerate here.

5 Conclusion

In this paper, we proposed a static behavior analysis system to capture raw features of Android application, and processed it with the help of feature extraction and selection, then there was a feature subset used for machine learning, finally, performance of three classification algorithms are tested. Experiments showed that this analysis can detect Android's unknown malicious applications effectively and accurately, and the classifiers are also lightweight. Dynamic behavior analysis solve the obfuscation, encryption code and other problem that only can be detected at runtime in the process of static analysis, and multi-class features of Android malware need to be taken into account in reality. Hence the next step is to add dynamic behavior analysis to our work for a better detecting accuracy and performance. In addition, we used the classic classification methods to do malware detection test, so we will optimize the classification algorithm to construct a more complete Android malware detecting model.

Acknowledgements

This research is supported by the Fundamental Research Funds for the Central Universities, No. W17JB00060.

References

[1] IDC, Smartphone OS Market Share. <http://www.idc.com/promo/smartphone-market-share/os> , 2016.

[2] 360 Internet Security Center, China Internet Security Report in 2016. <http://bbs.360.cn/thread-14837467-1-1.html>, 2016.

[3] Smartphone OS market share 2017 q1. <htt/ps://www.idc.com/promo/smartphone-market-share/os>, 2017.

[4] W. Enck, M. Ongtang, P. Mcda-Niel, On lightweight mobile phone application certification, in: Proc. the 16th ACM Conference on Computer and Communications Security, 2009.

[5] H.U. Wenjun, S. Zhao, J. Tao, X. Ma, L. Chen, A detection method and system implementation for android malware, Journal of Xi'an Jiaotong University 47(10)(2013) 37-43.

[6] W. Glodek, R. Harang, Rapid permissions-based detection and analysis of mobile malware using random decision forests, in: Proc. IEEE Military Communications Conference, 2013.

[7] Y. Zhou, H. Zhang, X. Zhang, P. Li, Malware detection based on Android permission information, Application Research of Computers 32(10)(2015) 3036-3040.

[8] J. Sahs, L. Khan, A machine learning approach to Android malware detection, in: Proc. European Intelligence and Security Informatics Conference, 2012.

[9] S.Y. Yerimas, S. Sezer, G. McWillrams, I. Muttik, A new Android malware detection approach using bayesian classification. <https://arxiv.org/abs/1608.00848>, 2013.

[10] B. Arslan, S. Gunduz, S. Sagiroglu, A review on mobile threats and machine learning based detection approaches, in: Proc. International Symposium on Digital Forensic and Security, 2016.

[11] DCCI, The report on Chinese Android mobile phone's privacy and security in 2016. <http://www.dcci.com.cn/media/download/2716b7369ac4a744c842683ecla89e718c74.pdf>, 2016.

[12] A. Shabtai, Y. Fledel, Y. Elovici, Automated static code analysis for classifying Android applications using machine learning, in: Proc. International Conference on Computational Intelligence and Security, 2010.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, Acm Sigkdd Explorations Newsletter 11(1)(2009) 10-18.

[14] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, C. Siemens, DREBIN: Effective and Explainable Detection of Android malware in Your Pocket, Internet Society, San Diego, CA, 2014.

[15] J. Li, Y. Zhang, X. Chen, Y. Xiang, Secure attribute-based data sharing for resource-limited users in cloud computing, Computers & Security 72(2018) 1-12.

[16] G. DATA, 8400 new android malware samples every day. <https://www.gdatasoftware.com/blog/2017/04/29712-8-400-

new-android-malw are-samples-every-day>, 2017.

[17] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Computing Surveys (CSUR) 41(3)(2009) 15.

[18] Z. Li, L. Sun, Q. Yan, W. Srisa-an, Z. Chen, DroidClassifier: efficient adaptive mining of application-layer header for classifying android malware, in: Proc. 2016 International Conference on Security and Privacy in Communication Systems, 2016.

[19] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, X. Zhang, Exploring permission-induced risk in android applications for malicious application detection, IEEE Transactions on Information Forensics and Security 9(11)(2014) 1869-1882.

[20] H. He, E. A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. IEEE Transactions on Knowledge & Data Engineering 9(2008) 1263-1284.

[21] A. Jindal, A. Dua, K. Kaur, M. Singh, N. Kumar, S. Mishra, Decision tree and SVM-based data analytics for theft detection in smart grid, IEEE Transactions on Industrial Informatics 12(3)(2016) 1005-1016.

[22] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence 97(1-2)(1997) 273-324.

[23] M.S. Alam, S.T. Vuong, Random forest classification for detecting Android malware, in: Proc. Green Computing and Communications, 2013.

[24] L. Breiman, Random forests, Machine Learning 45(1)(2001) 5-32. doi: 10.1023/A:1010933404324