Intelligent Automation And Soft Computing, 2019

Vol. 25, no. 3, 637–647
https://doi.org/10.31209/2019.100000118

Novel Android Malware Detection Method Based on Multi-dimensional Hybrid Features Extraction and Analysis

Yue Li[1],Guangquan Xu[2,3], Hequn Xian[1,*], Longlong Rao[3] and Jiangang Shi[4,*]

[1]College of Computer Science and Technology, Qingdao University, Shandong Province, 266071,China
[2] Big Data School, Qingdao Huanghai University, Qingdao City, Shandong Province, 266427, China
[3] Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, Tianjin, 300350, China
[4] Shanghai Shang Da Hai Run Information System Co., Ltd, Shanghai, 200444, China
*Corresponding author: Hequn Xian and Jiangang Shi

ABSTRACT
In order to prevent the spread of Android malware and protect privacy information from being compromised, this study proposes a novel multi-dimensional hybrid features extraction and analysis method for Android malware detection. This method is based primarily on a multidimensional hybrid features vector by extracting the information of permission requests, API calls, and runtime behaviors. The innovation of this study is to extract greater amounts of static and dynamic features information and combine them, that renders the features vector for training completer and more comprehensive. In addition, the feature selection algorithm is used to further optimize the extracted information to remove a number of extraneous features, and a new multi-dimensional hybrid features vector is obtained. The multi-dimensional hybrid features vector is then used to train the classification model. Finally, the unknown samples are detected and identified by using the obtained classification model. Our experiment is conducted based on 359 malicious and 500 benign applications as experimental samples, and the results indicate that our proposed method performs better in the accuracy rate of Android malware detection compared with those methods using static methods alone.

KEY WORDS: Android Malware Detection, Dynamic Analysis, Feature Selection, Machine Learning, Multi-dimensional Hybrid Features.

1 INTRODUCTION

WITH Internet-centric mobile applications and mobile intelligent devices becoming increasingly popular, mobile applications are playing increasingly critical roles in helping users fulfil a number of types of functions, including shopping, map navigation, and online work. According to statistics from Gartner (2018), in 2017, smartphone sales to end-users totaled over 1.5 billion units, an increase of 2.7 percent over 2016, among which Android systems accounted for 85.9 %, iOS 14.0 %, and other operating systems 0.1 %. Data from AppBrain (AppBrain, 2018) indicated that, as of April 25, 2018, the number of applications available from the Google Play Store, the official Android application (app) market, is approximately 3.8 million. Nowadays, hackers and malware developers are more inclined to select Android operating systems as the preferred attack target (Felt, Finifter, Chin, Hanna, & Wagner, 2011; Park, Seo, & Yi, 2016). The reasons for this are attributed to the open-source nature and openness of the Android system, and the fact that numerous Android third-party application markets do not have a rigorous app review mechanism. The increase in the number of malicious Android apps is significant. A report from Symantec (2018) indicates that threats in the mobile space continue to increase annually. The number of new mobile malware variants increased by 54 % in 2017, as compared to 2016. And in the year of 2017,

CONTACT Jiangang Shi lukepro@163.com

there were on average 24,000 malicious mobile applications blocked each day. As is well known, malicious software is a significant threat to the Android system. And the speed of malware spreading is also amazing (Xiao, Gong, & Yu, 2011). The common malicious behaviors of malicious apps include malicious fee deductions, information theft, and SMS hijacking, that could lead to serious disclosure of user privacy information, and possibly result in loss of life or property. As the usage of Android smartphones is increasing significantly, malicious software is also becoming more prevalent. Privacy data of mobile phone users has become a primary target of malicious software. In order to effectively prevent the spreading of Android malware, it is necessary to develop a safe and efficient detection method to identify and prevent the disclosure of privacy information.

Besides the threat to user privacy, malware could significantly threaten the underlying infrastructure as it could 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 radio access network (RAN) in close proximity to mobile subscribers, that are primarily mobile phones for users. Mobile-edge computing allows content, services, and apps to be accelerated, increasing responsiveness from the edge. The experience of mobile subscribers can be enriched through efficient network and service operations, based on insight into the radio and network conditions. Although computing platforms can provide secure facilities, the potential threat from malware in mobile phones still exists.

In recent years, numerous Android malware detection methods have been proposed. The traditional Android malware detection methods can be divided into signature-based methods (Grace, Zhou, Zhang, Zou, & Jiang, 2012) and behavior-based methods (Zolkipli & Jantan, 2010). Signature-based detection methods (Grace et al., 2012) are employed to extract the signatures of malicious software samples and store them in a database. When scanning the app to be tested, the stored signatures can be compared to the scanned one to determine whether it is a known malware. This method has the advantages of fast detection speed and high accuracy. However, the disadvantage lies in that it cannot detect unknown malware, and the virus signature database has to be continually updated. Behavior-based detection methods (Zolkipli & Jantan, 2010) are employed to monitor the behavior features of an app when it is running, and then match it to the known malicious behavior features to determine whether the target file contains malicious feature code. Although this method has a significant false-positive rate, it can detect unknown malicious codes or viruses.

As malicious software constantly uses new technologies and methods, the typical analysis methods based on signature and behavior analysis have lost their timeliness. As a result, people are continually attempting to apply new algorithms and methods to Android malicious software detection (Liang, Wu, Xu, & Ma, 2015; Sufatrio, Tan, Chua, & Thing, 2015). With the maturing of large-data-related technologies in recent years, a number of researchers have begun to introduce data mining and machine learning methods into Android malware detection (Arslan, Gunduz, & Sagiroglu, 2016).

Machine learning-based detection methods train the machine learning classification algorithm (Gu, Sun, & Sheng, 2017) by using the feature vector set extracted from the sample set, that can automatically predict the classification of unknown malware. The method of extracting features for each Android app is divided into static analysis and dynamic analysis. The static analysis method (Shabtai, Moskovitch, & Elovici, 2009) primarily utilizes the decompile technology to extract the application programming interface (API) call sequence, program instruction sequence, and other static features from an app, while it is not necessary to run the app. This method has the advantages of rapid detection and high efficiency. Static analysis methods are typically based on a number of rules to retrieve, match, and compare the known features, and then obtain the relevant results. However, in a number of cases, the static analysis method could yield false positives, that reduces the overall accuracy of static detection. In addition, utilizing code obfuscation techniques (You & Yim, 2010; Yin et al., 2016, 2017) can bypass detection by the static analysis method. Therefore, it can be combined with dynamic analysis to improve the accuracy rate. The dynamic analysis method (Tam, Fattori, Khan, & Cavallaro, 2015) uses "sandbox or virtual machine" to simulate the running of apps and analyzes the run-time behavior characteristics of the application through interception and monitoring technologies. To some extent, this method can bypass code obfuscation and other code protection mechanisms, however, the detection speed is relatively slow.

In order to overcome the shortcomings of the existing research methods, we propose a hybrid features analysis method to detect Android malware, that combines the advantages of static and dynamic analysis methods. In this study, the hybrid features vectors are extracted using a hybrid feature analysis method. In order to determine the most efficient detection method to deal with Android malware threats, we train the five different machine learning classifiers with the vectors, including the support vector machine (SVM), decision tree (J48), random forest, naive Bayes, and k-nearest neighbor. Experimental results indicate that, compared to using only the static analysis method, the feature set

extracted by the hybrid analysis method is more efficient in training classifiers.

The rest of this study is organized as follows. Related work is discussed in Section 2. Section 3 briefly describes the Android malware detection model, analyzes the various feature extraction methods, and optimizes the extracted feature sets using the feature selection algorithm. Section 4 presents the experimental results. Finally, we conclude in Section 5 and discuss future studies.

2 RELATED WORK

IN recent years, there have been numerous related studies applying machine learning methods in the Android malware-detection field.

In the static analysis method, Cho, Kim, Shim, Ryu, and Im (2016) proposed a malware family classification framework using a sequence alignment method, which was widely used in the bioinformatics field. Yerima, Sezer, McWilliams, and Muttik (2013) used static analysis methods to detect unknown Android malware by means of training Bayesian classifiers. The study was based on a great number of actual malware samples, extracting applications permissions as features. It could filter out the benign applications to decrease the workload of virus analysis experts and overcome the limitations of the traditional signature-based detection methods. Chan and Song (2014) proposed a static Android malware detection method that extracted the permissions and API calls characteristics of each app as the feature-vector set for classifier training. Peng et al. (2012) used probabilistic generative models to calculate the risk level and risk scores according to app request permissions. Drebin (Arp, Spreitzenbarth, Hubner, Gascon, & Rieck, 2014) conducted a static analysis of the Android package (APK) file. They not only extracted the permissions of the application request from the manifest file, but also analyzed the application of the sensitive API calls and a number of network addresses from the Dalvik executable (Dex) file. Drebin transformed the feature information obtained from static analysis into feature vectors, and then detected malware by using support vector machine algorithms. Wu, D., Mao, Wei, Lee, and Wu, K. (2012) proposed a method of malicious behavior analysis based on static behavior characteristics. This method could characterize the malicious behavior by extracting static information from the APK, including requests permissions, components, intents, and API calls. In order to detect different types of malicious behaviors, this study compared the results of different algorithms in the detection of different feature sets and recognition accuracy. According to the experimental results, the greatest recognition rate of this method was when the application is first clustered using the k-means algorithm and then classified using the k-nearest neighbor algorithm.

In the dynamic analysis method, Amos, Turner, and White (2013) proposed STREAM, a feature vector collection framework, that accelerated the large-scale verification of machine learning classification of Android malware. It was a distributed mobile malware detection framework that could automatically train and evaluate malware classifiers. In addition, STREAM was designed to be configurable, allowing future researchers to modify and configure the framework according to their own needs. Dash et al. (2016) proposed DroidScribe, a method of automatic classification of Android malware based on dynamic runtime behavior analysis, that used dynamic analysis to observe the runtime behavior of system calls, and provided an Android malware detection method different from the static method. Burguera, Zurutuza, and Nadjm-Tehrani (2011) proposed Crowdroid, a method of using dynamic methods to analyze application behavior to detect Android malware by collecting the system call traces of apps that were running on different Android platforms as feature sets and using clustering algorithms to detect malware. Sahs and Khan (2012) proposed a detection system based on machine learning, extracting great numbers of features and using off-line methods to train a one-class support vector machine. Rieck, Trinius, and Willems (2011) proposed a framework for automatically analyzing the malware behavior using machine learning methods that performed a behavioral analysis in an incremental manner, avoiding the run-time and memory overhead of previous methods.

3 PROPOSED DETECTION METHOD

IN using machine learning to detect Android malware, the bulk of current studies are based on either static or dynamic analysis to extract the features of Android apps. We proposed an approach that combines the advantages of static and dynamic analysis. Using the hybrid-analysis method, we select three characteristic attributes that can reflect the behavior of Android apps in nature as feature vectors: the request permissions, API calls information, and dynamic runtime behaviors of Android apps. The three extracted features are then formed into a hybrid feature vector, that is used to train the machine learning classification algorithm. Finally, the unknown samples are detected and identified according to the classification model that has been trained.

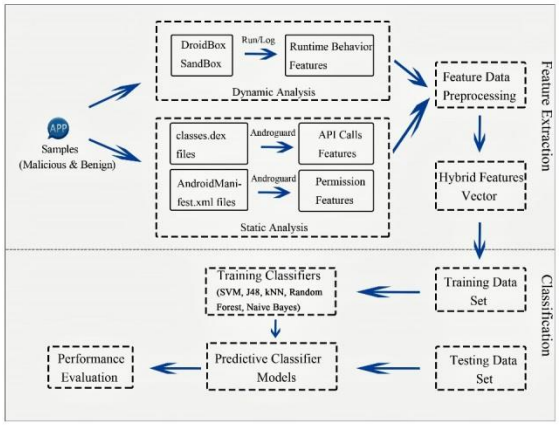

Figure 1. Architecture of proposed detection method.

3.1 Architecture of The Proposed Approach

An overview of the proposed method is shown in Figure 1. The method of using machine learning to detect malware is divided primarily into the following parts: data collection, feature extraction, feature data preprocessing, classifier model training, and classification results. The complete malware detection process can be divided into two phases: the training phase, and the testing phase. In the training phase we first extract feature vectors from benign software and malicious software. The feature vectors are then selected to remove the features that have no effect on the classification results, and the optimized feature vectors are obtained. Finally, a hybrid features vector is formed as an input for the classifier model. Different classifier models are then selected to train, and the classifier models are obtained through continuous training. In the testing phase, the unknown samples are detected by the obtained classifier model. As 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.

3.2 Feature Extraction

This section introduces the extraction of hybrid feature vector sets. The acquisition of hybrid feature vectors is a highly critical step. First, the static analysis method is used to analyze the Android APK files, that can extract the permissions features and sensitive API calls features of each app. Second, the dynamic analysis method can be used to extract the runtime behavior features of each app in the running process. Finally, after obtaining the hybrid feature vector sets, the feature selection algorithm is used to optimize the acquired feature information and eliminate the features that have no influence on the classification.

3.2.1 Static analysis and static features

Currently, the bulk of the static analysis methods are extracted and analyzed from, amongst others, AndroidManifest.xml file, Lib library files (.so files), and Java source files. In this study, we analyze primarily the classes.dex and AndroidManifest.xml files in the APK file, and then invoke the reverse analysis tool to parse the above two files and extract the feature vectors. The AndroidManifest.xml file is one of the most important files in an Android app. It is an important permission request and definition profile for the Android system, and programmers must predefine and apply for permissions required by the app in the AndroidManifest.xml file when developing the app. Therefore, we parse this file to extract the permissions characteristics of the app. The Dex file is an executable bytecode file on Android that is compiled by the Java Virtual Machine (JVM) and then compiled by the Android virtual machine Dalvik. The classes.dex is an essential file for each app, and contains the primary execution code for Android apps. To facilitate the reverse analysis, we use related tools to obtain readable Java sources through decompiled DEX files.

***Permission Extraction*:** The primary aim of the Android system setup permission mechanism is to restrict apps accessing sensitive resources. However, the "all-or-nothing" feature of the permission mechanism is a security weakness. When developing an app, the developer must first apply for all the required permission information in the AndroidManifest.xml file. Zhou and Jiang (2012) reported that, from statistics of the frequency of use of the number of permissions in benign and malicious apps, a number of the SMS-related permissions, as well as boot-strap self-starting permissions were typically extensively used in malicious apps. A number of malicious apps must request appropriate sensitive permissions. The differences in information in these permissions provide the theoretical feasibility of the permissions as a feature of Android malware detection. The process of acquiring Android app permissions is to decompile the APK file using the decompile tool, obtain the AndroidManifest.xml file, and then read the file and obtain the permissions information declared in the file.

In this study, we use the open source tool Androguard (2018) to extract the permission features from the APK package. The Androguard tool (Androguard, 2018) is a powerful Android malware analytics tool that provides a set of toolkits to assist analysts in quickly identifying and analyzing the APK files, making it simpler to get the information required for the static analysis. We use the androlyze.py tools in the Androguard open source project to extract sensitive permission features from normal samples and malicious samples. By analyzing the results extracted from a great number of app samples, the apps that exhibit malicious behavior frequently require

numerous sensitive permissions, such as malicious fee-absorbing applications that typically apply for SMS-related permissions, where SEND_SMS permissions allow applications to send text messages, READ_SMS permissions allow applications to read SMS content. Of these, the ACCESS_NETWORK_STATE, READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, and INTERNET appear most frequently in normal and malicious samples. We counted the number of times a permission appears in samples and sorted the results in descending order. Due to the length of the paper, we only list the top 10 permissions here. Permissions and their functional descriptions are presented in Table 1. After obtaining the permission list, we selected the top 45 relevant permissions to form the feature vector through optimization and analysis. Each app can be represented by a 45-dimensional vector $[Per]_{1\times 45}$, and each dimension corresponds to a permission. If the AndroidManifest.xml file of an app 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 user contact information

***API Calls Extraction*:** The APIs investigated in this study refer to the function provided by the Android system itself. By invoking these functions, the app can access and obtain an amount of sensitive data in the mobile phone, including contacts, geographic location, photos, and accounts. It could also trigger high-risk behaviors such as secretly connecting the network and sending malicious SMS messages for deducting expenses. These APIs, that are related to sensitive data and high-risk behaviors, are referred to as sensitive APIs in this study. As with the permissions information, there are significant differences in the use of these sensitive APIs because of the difference between benign software and malicious software. The malicious application of the number of calls to sensitive APIs is significantly greater than the benign application, that can reflect the actual behavior characteristics of an app to some extent, and therefore can be used as a feature of the app to identify malicious behavior. We use the open source tools Baksmali (2018) 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 great number of sample sets, and then used the filter feature selection algorithm Relief (Kira and Rendell 1992) to optimize them. We then totaled the number of times each API is called as the initial value of the relevant statistic vector component. After the feature selection process, we obtained 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 presents the 22 selected API calls.

Table 2. Sensitive API calls.

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

3.2.2 Dynamic analysis and dynamic features

In the dynamic analysis phase, the primary work is the collection of the runtime behavior features of each app. Android apps comprise a variety of components that can trigger a series of interface calls. In order to optimally collect the runtime behavior features of the unknown sample in the behavioral detection of the app, when the app installed in the simulator is running, we use the automated test tool Monkey (2018) to simulate the event flow to run all components of the app. The Monkey can generate pseudo-random event streams, that can send a series of event streams to the app and can obtain the behavior characteristics when the app receives various events. Automatic test technology uses programs instead of humans to simulate the daily operation of a user. 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 (2018) to monitor the runtime behavior of apps. This is an Android dynamic analysis tool that allows rapid collection and visually displays the behavior of the app. Its primary functions are monitoring the information that includes the network communication

data, file read and write operations, information leakage in SMS, and broadcast receiver component information. By using the log mechanism of the Android systems and DroidBox, the app behavior information in the framework layer and native layer, that can accurately reflect the behavioral characteristics of the app, can be obtained. We installed and ran each app on DroidBox, and then used automated test techniques to monitor whether each app exhibited malicious behavior, including automatic connection to the network, sending malicious SMS messages, and obtaining privacy information. 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 significant number of aspects such as the battery, binder, network, and user activity. Of these aspects, behavior_sentSMS represents the behavior of sending SMS messages, behavior_out-goingCalls represents the behavior of making a call, behavior_openingKeyboard is the behavior that opens keyboard input, behavior_packetsWiFi represents the behavior of sending packets over a WiFi, and behavior-r_openingCamera represents the behavior of opening the camera. As a result, we obtained 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 two sections, three feature vectors of three types of features are formed Each app can obtain a set of permission feature vectors $[Per]_{1\times 45}$, a set of API calls feature vectors $[API]_{1\times 22}$, and a set of runtime behaviors feature vectors $[Runbehavior]_{1\times 20}$. Combining these three feature vectors sets, each app can be represented by an 87-dimensional hybrid feature vector $[Per, API, Runbehavior]_{1\times 87}$. Each feature in the hybrid feature vector 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 represent 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. We extracted a significant number of features. However, in order to improve the efficiency and accuracy of the classifier, it is necessary to remove the features that have no effect on the classification. At the same time, there are excessive irrelevant features that have an influence on the effect of the classification. The greater the number of features used in training the classification model, the longer it takes to train the classification model. Therefore, the feature selection is critical to the training of the model. This study assumes that the initial feature set contains all the important information. The process of feature selection is to select a subset of the features that contain all the important information from the initial set of features.

In this study, we use the filter feature selection algorithm to first select the data sets, and then train the classifier. The feature selection process is independent of the subsequent classifier. Kira and Rendell (1992) proposed Relief, that is a highly efficient filter feature selection algorithm. It uses "relevant statistic vectors" to measure the importance of features. The algorithm is primarily aimed at solving two classification problems. The basis of Relief is how to determine the value of the "relevant statistic vector". Assume that the training set D is $\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$, for each sample x_i, its feature j corresponds to the relevant statistic vector is as follows:

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

where x_a^j represents the value of sample x_a on the feature j, $x_{i,nh}$ is labeled a "near-hit", that represents the closest neighbor of sample x_i in its same category, and $x_{i,nm}$ is labeled "near-miss", that represents the nearest neighbor of sample x_i in a different category. As the features of the malicious samples and benign samples extracted in this study are discrete, if $x_a^j = x_b^j$, then $diff(x_a^j, x_b^j) = 0$; otherwise $diff(x_a^j, x_b^j) = 1$. As can be seen from the above equation, if the distance between x_i and its near-hit $x_{i,nh}$ on feature j is less than x_i and its near-miss $x_{i,nm}$, then it is indicated that feature j is advantageous to distinguish between the malicious samples and benign samples; otherwise, it is disadvantageous. Therefore, the greater the value of equation (1), the stronger the classification ability of the feature is. From equation (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, and the greater the vector component value, the stronger the classification ability.

3.4 Machine Learning Classifier

The malware detection based on machine learning is a new application field of machine learning, and its essence is to classify benign software and malicious software using classification algorithms. Android malware detection belongs to two-classification problems, and we choose different classifier algorithms to detect malicious software. The classifier algorithms we used include SVM (Cristianini & Shawe-Taylor, 2000; Gu & Sheng, 2017; Gu, Sheng, & Wang, 2015), k-nearest neighbor (Liao & Vemuri, 2002), naive Bayes (Domingos & Pazzani, 1997), decision tree (J48) (Quinlan, 1986), and random forest (Ho, 1998). Of these, the J48 decision tree algorithm we used in our experiment is the implementation of the C4.5 algorithm (Quinlan, 1993) in WEKA (Hall, Frank, & Holmes, 2009). Different classifier algorithms have different detection capabilities, therefore, it is essential to select the appropriate classifier algorithm. Typically, the performance of a classifier algorithm is evaluated by using three performance measures: true positive rate (TPR), false positive rate (FPR), and accuracy. The comparison and analysis of different classifier algorithms is a critical point in this study. In the experimental verification phase, we will make a comparative analysis of these algorithms to determine which has the best capabilities for Android malware detection.

4 EXPERIMENTS AND RESULT

IN this section, we use the machine learning tool WEKA (Hall et al., 2009) to train the classification model for the features obtained from the experimental samples. Based on the Java environment, WEKA is free open source machine learning software, that integrates a great number of machine learning algorithms and has the characteristics of significant efficiency and accuracy. All experiments were performed on a computer with a 3.20GHz Intel (R) Core (TM) i5 CPU, with 8GB of memory.

4.1 Data Collection

In order to ensure the reliability and wide coverage of the experiment, we collected a total of 359 malicious apps and 500 benign apps as experimental samples. Of these, 228 malicious samples were derived from the third-party sample collection platform VirusShare (https://virusshare.com), and 131 malicious samples from the set of malware samples provided by Contagiomobie (http://contagiominidump.blogspot.co-m). These two platforms systematically collected a wide range of families of Android malicious samples, including their various derivative versions, and provided powerful data support for our malware detection study. The benign samples used in this study were primarily downloaded from the Google Play Store. We assumed that all of the apps from the Google Play Store were benign applications. In this experiment, we randomly selected 150 malicious apps and 150 benign apps from the experimental samples, and then combined them as a training set. We similarly obtained a test set. The features were then extracted from each app according to the method described in Section 3. By integrating the static feature vectors and the dynamic feature vectors, hybrid feature vectors for classification were formed. The following experiments were performed on these two data sets.

4.2 Performance Evaluation

The following introduces a number of appropriate performance metrics that evaluate the performance of a classification algorithm. The four basic metrics are true positive (TP), false positive (FP), true negative (TN), and false negative (FN). where TP is the number of malicious applications classified correctly, FP is the number of benign applications incorrectly classified, TN is the number of benign applications correctly classified, and FN is the number of malicious applications incorrectly classified. As can be seen in Table 3, these four metrics can form a confusion matrix.

Table 3. Confusion matrix.

Prediction	Malicious	Benign
Malicious	TP	FN
Benign	FP	TN

The following five performance measures are derived from the confusion matrix for calculating and evaluating the performance of the classifier:

$$True\ positive\ rate(TPR) = Recall = \frac{TP}{TP + FN} \quad (2)$$

$$False\ positive\ rate(FPR) = \frac{FP}{TN + FP} \quad (3)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

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

Where TPR, or the recall rate, is the proportion of malware being correctly predicted by the classifier, and FPR, or false alarm rate, 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 the total number of samples, and is used to measure system errors, and the greater the value, the smaller the system error. Precision is the number of samples correctly classified as malicious among those classified as malicious. The

Matthews correlation coefficient (MCC) is used to evaluate the quality of binary classifications in machine learning. The value range is [-1,1], where 1 represents the perfect prediction, 0 represents the same effect as the random classifier, and -1 represents the predicted result being completely different from the actual result. The greater the values of accuracy and TPR, the smaller the value of FPR, and the better the classification effect.

4.3 Experiment Results

There are two primary aims of our study: to verify whether the combination of static analysis and dynamic analysis can improve the accuracy of malware detection, and to determine which classification algorithm is the best for Android malicious software detection.

In Table 4, we present the classification results for five different classifiers when using only static methods to extract features (i.e., permission, API calls). The results indicate that the accuracy rate of the five classifier algorithms is greater than 84 %. The table also indicates that the random forest algorithm has the greatest accuracy rate of 92.07 %, and its classification effect is the best. By contrast, Table 5 presents the classification results for the five different classifier algorithms, SVM, k-nearest neighbor, naive Bayes, decision tree (J48), and random forest, when the feature extraction uses the hybrid analysis method (i.e., permission, API calls, and runtime behavior). As can be seen from Table 5, the performance of the random forest algorithm remains the best with an accuracy rate of 94.89 %, that is 2.82 % greater than when only using static methods. The classification accuracy of the SVM algorithm improved by approximately 2.4 % to 93.66 %. Compared to Table 4, we find that the performance metrics of the experimental results are improved after using the hybrid analysis method, and it is clear that the hybrid analysis method improves the detection accuracy.

Table 4. Classification results from static methods.

Classifier Model	Measure Metrics (%)				
	TPR	FPR	Accuracy	Precision	MCC
SVM	92.47	9.74	**91.27**	**90.47**	**0.83**
J48	91.97	14.72	88.19	86.20	0.77
Naive Bayes	93.49	19.77	85.74	82.54	0.74
KNN	90.01	19.41	84.56	82.26	0.71
Random Forest	92.57	8.55	**92.07**	**91.54**	**0.84**

Table 5. Classification results from hybrid methods.

Classifier Model	Measure Metrics (%)				
	TPR	FPR	Accuracy	Precision	MCC
SVM	95.17	7.74	**93.66**	**92.48**	**0.87**
J48	93.38	14.02	89.34	86.95	0.80
Naive Bayes	89.39	19.05	84.52	82.43	0.71
KNN	92.31	17.65	86.71	83.95	0.75
Random Forest	95.3	5.3	**94.89**	**94.73**	**0.90**

Figure 2 shows more visual and intuitive classification effects of the different classifier algorithms in Android malware detection. It can be clearly seen that the random forest algorithm has the best classification effect, followed by the SVM algorithm, while the naive Bayes algorithm has the smallest detection accuracy. To summarize, the experimental results indicate that the feature extraction method of hybrid analysis can improve the accuracy of classification results in Android malware detection. In addition, by comparison of the performance of five different classifiers, we determined that the random forest algorithm has the best detection effect.

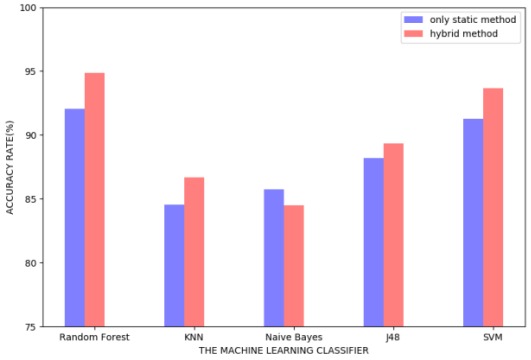

Figure 2. Accuracy analysis: Using different classifiers.

5 CONCLUSIONS AND FUTURE WORKS

IN this study, we proposed a multi-dimensional hybrid-features extraction and analysis method for the detection of Android malwares by extracting permissions, API calls, and runtime behaviors as feature sets. Combining the three into a set of hybrid-feature vectors, we used the different machine learning classification algorithms to train the classification detection model after the feature selection algorithm was used to optimize it. We validated the proposed method simulation experiments. The experimental results indicated that this method could effectively detect and classify Android malware and obtain greater detection rates.

Typically, because of the diversity of malicious behavior in malicious apps, the features extracted by the hybrid analysis method can more comprehensively and effectively show the characteristics of Android apps. We demonstrated that the multi-dimensional hybrid analysis method can improve the accuracy of Android malware detection compared to single static feature extraction methods. In addition, after reducing the dimension of the extracted hybrid feature vectors, a number of extraneous features were removed, that increased the classification accuracy and achieved better detection effects. Finally, we chose different classification algorithms to compare the classification effects, and through analysis it was found that the random forest and SVM algorithms exhibited the greatest accuracy rates.

For future studies, we will 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 and select better feature selection algorithms to reduce the redundancy of features, and further improve classification efficiency.

6 ACKNOWLEDGEMENT

THIS work is partially sponsored by the State Key Development Program of China (No. 2017YFE-0111900, No. 2017YFB1401201), National Science Foundation of China (No. 61572355, U1736115).

7 DISCLOSURE STATEMENT

NO potential conflict of interest was reported by the authors.

8 REFERENCES

Amos, B., Turner, H., & White, J. (2013) Applying machine learning classifiers to dynamic Android malware detection at scale. *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)* (pp. 1666-1671). Sardinia, Italy: IEEE.

Androguard (2018). Retrieved from https://code.google.com/archive/p/androguard.

AppBrain (2018). Android Operating System Statistics, Google Play Stats. Retrieved from https://www.a-ppbrain.com/stats/number-of-android-apps.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS Symposium 2014. DOI: 10.14722/ndss.2014.23247.

Arslan, B., Gunduz, S., & Sagiroglu, S. (2016). A review on mobile threats and machine learning based detection approaches. *2016 4th International Symposium on Digital Forensic and Security (ISDFS)* (pp. 7-13). Little Rock: IEEE.

Baksmali (2018). Retrieved from https://github.com/JesusFreke/smali.

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* (SPSM '11) (pp. 15-26). New York, USA: ACM.

Chan, P. P., & Song, W. K. (2014). Static detection of Android malware by using permissions and API calls. *2014 International Conference on Machine Learning and Cybernetics* (pp. 82-87). Lanzhou, China: IEEE.

Cho, I. K., Kim, T. G., Shim, Y. J., Ryu, M., & Im, E. G. (2016). Malware Analysis and Classification Using Sequence Alignments. Intelligent Automation & Soft Computing, 22(3), 371-377.

Cristianini, N., & Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge: Cambridge University Press.

Dash, S. K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., & Cavallaro, L. (2016). DroidScribe: Classifying Android Malware Based on Runtime Behavior. *2016 IEEE Security and Privacy Workshops (SPW)* (pp. 252-261). San Jose, CA, USA: IEEE.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. Machine learning, 29(2), 103-130.

DroidBox (2018). An Android Application Sandbox for Dynamic Analysis. Retrieved from http://code.google.com/p/droidbox.

Felt, A.P., Finifter, M., Chin, E., Hanna S., & Wagner, D. (2011). A survey of mobile malware in the wild. Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 3-14). New York, USA: ACM.

Gartner (2018). Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017. Retrieved from https://www.gartner.com/newsroom/id/3725117.

Grace, M., Zhou, Y., Zhang, Q., Zou, S. H., & Jiang, X. X. (2012). Riskranker: scalable and accurate zero-day android malware detection. Proceedings of the 10th international conference on Mobile systems, applications, and services (pp. 281-294). New York, USA: ACM.

Gu, B., & Sheng, V.S. (2017). A robust regularization path algorithm for v-support vector classification. In *IEEE Transactions on Neural Networks and Learning Systems*, 28(5), 1241-1248.

Gu, B., Sheng, V.S., & Wang, Z. (2015). Incremental learning for v-support vector regression. Neural Networks, 67, 140-150.

Gu, B., Sun, X., & Sheng, V. S. (2017). Structural Minimax Probability Machine. I*EEE Transactions on Neural Networks and Learning Systems*, 28(7), 1646-1656.

Hall, M., Frank, E., & Holmes, G. (2009). The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 10-18.

Ho, T.K. (1998). The random subspace method for constructing decision forests. IEEE transactions on pattern analysis and machine intelligence, 20(8), 832-844.

Kira, K., & Rendell, L.A. (1992). The feature selection problem: Traditional methods and a new algorithm. Proceedings Tenth National Conference on Artificial Intelligence (pp. 129-134). San Jose, CA: AAAI.

Liang, H., Wu, D., Xu, J., & Ma, H. (2015). Survey on privacy protection of android devices. *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 241-246). New York, USA: IEEE.

Liao, Y., & Vemuri, V.R. (2002). Use of k-nearest neighbor classifier for intrusion detection. Computers & security, 21(5), 439-448.

Monkey (2018). Retrieved from https://developer.android.com/studio/test/monkey.html.

Park, S., Seo, C., & Yi, J. H. (2016). Cyber threats to mobile messenger apps from identity cloning. Intelligent Automation & Soft Computing, 22(3), 379-387.

Peng, H., Gates, C., Sarma, B., Li, N. H., Qi, Y., Potharaju, R., … Molloy, I. (2012). Using probabilistic generative models for ranking risks of android apps. Proceedings of the 2012 ACM conference on Computer and communications security (pp. 241-252). New York, USA: ACM.

Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

Quinlan, J. R. (1993). C4.5: Programs for machine learning. San Francisco: Morgan Kaufmann.

Rieck, K., Trinius, P., & Willems, C. (2011). Automatic analysis of malware behavior using machine learning. Journal of Computer Security, 19(4), 639-668.

Sahs, J., & Khan, L. (2012). A Machine Learning Approach to Android Malware Detection. *2012 European Intelligence and Security Informatics Conference* (pp. 141-147). Odense, Denmark: IEEE.

Shabtai, A., Moskovitch, R., & Elovici, Y. (2009). Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A state-of-the-art Survey. Information security technical report, 14(1), 16-29.

Sufatrio, Tan, D. J., Chua, T. W., & Thing, V. L. (2015) Securing android: a survey, taxonomy, and challenges. ACM Computing Surveys (CSUR), 47(4), 58.

Symantec (2018). Internet Security Threat Report 2018. Retrieved from https://www.symantec.com/security-center/threat-report.

Tam, K., Fattori, A., Khan, S., & Cavallaro, L. (2015). CopperDroid: Automatic Reconstruction of Android Malware Behaviors. In NDSS *Symposium 2015*. DOI: 10.14722/ndss.2015.23145.

Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012). DroidMat: Android Malware Detection through Manifest and API Calls Tracing. *2012 Seventh Asia Joint Conference on Information Security* (pp. 62-69). Tokyo, Japan: IEEE.

Xiao, R., Gong, X., & Yu, T. (2011). A Simulation Approach To The Control Mechanism Of Individual And Web Site In Malware Spread. Intelligent Automation & Soft Computing, 17(6), 781-792.

Yin, Y., Aihua, S., Min, G., Yueshen, X., & Shuoping, W. (2016). QoS prediction for web service recommendation with network location-aware neighbor selection. International Journal of Software Engineering and Knowledge Engineering, 26(04), 611-632.

Yerima, S. Y., Sezer, S., McWilliams, G. & Muttik, I. (2013). A New Android Malware Detection Approach Using Bayesian Classification. *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)* (pp. 121-128). Barcelona, Spain: IEEE.

Yin, Y., Xu, Y., Xu, W., Gao, M., Yu, L., & Pei, Y. (2017). Collaborative service selection via ensemble learning in mixed mobile network environments. Entropy, 19(7), 358.

You, I., & Yim, K. (2010). Malware Obfuscation Techniques: A Brief Survey. *2010 International Conference on Broadband, Wireless Computing*, Communication *and Applications* (pp. 297-300). Fukuoka, Japan: IEEE.

Yin, Y., Yu, F., Xu, Y., Yu, L., & Mu, J. (2017). Network location-aware service recommendation with random walk in cyber-physical systems. Sensors, 17(9), 2059.

Zhou, Y., & Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. *2012 IEEE Symposium on Security and Privacy* (pp. 95-109). San Francisco, CA, USA: IEEE.

Zolkipli, M. F., & Jantan, A. (2010). Malware behavior analysis: Learning and understanding current malware threats. *2010 Second International Conference on Network Applications, Protocols and Services* (pp. 218-221). Kedah, MYS: IEEE.

9 NOTES ON CONTRIBUTORS

Yue Li is currently an undergraduate student at College of Computer Science and Technology, Qingdao University, China. Her research interests include information security, IoT security.

Guangquan Xu is a Ph.D. and full professor at the Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, China. He received his Ph.D. degree from Tianjin University in March 2008. He is a member of the CCF and IEEE. His research interests include cyber security and trust management. He is the director of Network Security Joint Lab and the Network Attack & Defense Joint Lab. He has published 70+ papers in reputable international journals and conferences, including IEEE IoT J, FGCS, IEEE access, PUC, JPDC, IEEE multimedia, and so on. He served as a TPC member for IEEE UIC 2018, SPNCE2019, IEEE UIC2015, IEEE ICECCS 2014 and reviewers for journals such as IEEE access, ACM TIST, JPDC, IEEE TITS, soft computing, FGCS, and Computational Intelligence, and so on.

Hequn Xian received Ph.D degree in the Institute of Software, Chinese Academy of Sciences in 2009. He was a visiting scholar with college of information science and technology, the Pennsylvania State University. His research interests include cryptography, cloud computing security, and network security.

Longlong Rao is currently a master student at the Tianjin Key Laboratory of Advanced Networking (TANK), School of Computer Science and Technology, Tianjin University, China. He received a bachelor degree from the Xizang Minzu University in July 2013. His interests include network attack and PT.

Jiangang Shi is CEO of Shanghai Shang Da Hai Run Information System Co., Ltd, leading multiple information systems projects development in the field of Campus Information Service. His research interests include software engineering, deep learning, data mining.