Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2022.023874
Article

Tech Science Press

Android Malware Detection Based on Feature Selection and Weight Measurement

Huizhong Sun[1], Guosheng Xu[1,*], Zhimin Wu[2] and Ruijie Quan[3]

[1]School of Cyberspace Security, Beijing University of Posts and Telecommunication, Beijing, 100786, China
[2]National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT), Beijing, 100029, China
[3]University of Technology Sydney, Sydney, Australia
*Corresponding Author: Guosheng Xu. Email: guoshengxu@bupt.edu.cn
Received: 24 September 2021; Accepted: 28 October 2021

Abstract: With the rapid development of Android devices, Android is currently one of the most popular mobile operating systems. However, it is also believed to be an entry point of many attack vectors. The existing Android malware detection method does not fare well when dealing with complex and intelligent malware applications, especially those based on feature detection systems which have become increasingly elusive. Therefore, we propose a novel feature selection algorithm called frequency differential selection (FDS) and weight measurement for Android malware detection. The purpose is to solve the shortcomings of the existing feature selection algorithms in detection and to filter out other effective features. Weight measurement is used to optimize the detection accuracy of the classifier and improve the accuracy of detection. We combine the optimized features and the detection model for verification and evaluation. Experiments were conducted on the OmniDroid dataset, which is a large and comprehensive dataset of features extracted from 22,000 real malware and benign samples. Theoretical analysis and experimental results showed that the FDS algorithm and weight measurement are effective, feasible, and exhibit advantages over other existing malware detection models. In detecting Android malware samples, the proposed method can achieve an accuracy of 99% and an F1-score of 98%.

Keywords: Malware detection; feature detection; weight measurement; optimized features

1 Introduction

Android is an open-source operating system for smart mobile devices that was officially released in November 2007. As a result of its open and free functions, the Android operating system has rapidly become popular. According to the latest statistics in October 2019, Android exhibited the largest market share of smart mobile device operating systems, accounting for 87% [1]. At the same time, a large number of Android malware has been released to the Android application (app) market. According to a report from Tencent Mobile Security Lab, 1,898,731 new Android malware applications were found in the first half of 2019, and they infected approximately 38.13 million mobile phone users [2]. These malware applications are used by attackers to damage user systems, invade user privacy, maliciously deduct fees, etc., thereby posing a serious threat to user information and property security. Therefore, many researchers conducted extensive research on the detection of malware applications on the Android platform.

Several approaches to Android malware detection have been proposed, and they are categorized into two types, namely, dynamic detection and static detection [3]. Dynamic detection is to execute Android applications in real devices or run Android applications in a controlled environment, such as Android virtual devices or sandboxes, to monitor the behavior of applications. Several dynamic detection approaches [4–9,10] have been proposed. However, these approaches rely on the ability to detect malicious behavior during runtime while providing the perfect environment to kick-start malicious codes [11]. By contrast, static detection involves the analysis of application features to detect malware. These approaches can identify applications before the software is installed and prevent malware in advance. By reverse engineering the Android application package (APK), we can obtain related files to extract static features. Unlike the process of obtaining dynamic features, extracting static features is relatively convenient and does not waste the user's system resources. The extraction of static features usually involves a large amount of application information, such as API calls, permissions, opcodes, and intents. These data can further extract the key feature information required by the application functions. However, the measurement of the importance of features and the assignment of appropriate weights to each feature has become necessary. Such requirements cannot be fulfilled with previous static detection methods based on machine learning. For example, a dataset collected in a real environment contains many of the features of the application; therefore, many features exist related to each application extracted by the researchers. These features are key factors that reflect maliciousness and benignity. Hence, if all the features of each dimension in the extracted data set are used as training, then the detection model becomes affected by "dimension disaster" and consumes a considerable amount of computational overhead. Furthermore, many features do not contribute to the improvement of detection accuracy, or they have little to do with malware detection. Therefore, based on feature selection and feature weight measurement, we propose a novel Android malware detection model to reduce the computing resources of irrelevant features and adapt to the optimized detection model, thereby improving the detection accuracy for Android malware. In summary, the main contributions of this work are as follows:

(1) Analysis of the statistical information of the difference between malicious and benign samples and application of the frequency differential selection (FDS) algorithm to extract the importance of related features and thereby reduce the impact of irrelevant features;

(2) Measurement of the importance of the features, calculation of the weight of each feature, and establishment of the optimal weight for the classifier to improve the accuracy of model detection;

(3) Combination of feature selection and optimized feature weights for evaluation in the proposed method to achieve an accuracy of 99% in detecting Android malware samples and an F1-score of 98%.

The remaining summary structure of this paper is arranged as follows. First, our research on the Android malware detection is described in Section 2. Second, based on the current problems encountered in Android malicious detection, we introduce the solutions proposed in this article, such as the Android malware detection method shown in Section 3. Finally, as shown in Section 4, we evaluate and analyze our proposed scheme. In Section 5, we summarize the work in this field and prospects for future work.

2 Related Work

The two types of Android malware detection methods are dynamic detection methods and static detection methods. However, static detection methods have been more widely studied and applied than dynamic detection methods. In static detection, the APK must be reversed, and the information from various files, such as permissions and API information, must be extracted. Felt et al. [12] evaluated the availability of permission mechanisms and analyzed sensitive permission characteristics to identify malware applications. Zhang et al. [13] proposed a clustering algorithm that retains heterogeneous information from multiple sources, including the results of static analysis, meta-information of Android applications, and raw tags from anti-virus engines. This study does not take fuzzy malware into account and only sets the whitelist approach to exclude common third-party libraries for Android applications. Arora et al. [14] proposed a permission-based PermPair detection model, which focuses on exploiting potentially dangerous permissions in Android malware when granting permissions to known or unknown users. Additionally, they performed different important combinations of potentially dangerous permissions to identify causes of important permissions for malware detection. However, although this permission-based detection model is very fast, most Android applications are often complex in function and require more permission. Thus, obtaining guaranteed detection accuracy in practical applications is difficult. Kumar et al. [15] proposed an approach to distinguish malicious and benign applications using association rules based on limited permissions, which uses the convolutional neural network convolutional neural networks (CNN) model for malicious code detection based on pattern recognition and permissions triggered risk of Android IOT devices. However, due to the lack of behavioral semantic analysis, this approach cannot cope with the evolution of malware. Furthermore, Nix et al. [16] focused on the classification of the Android applications by using system API call sequences and investigated the effectiveness of deep neural networks in malware detection; the results indicated a detection accuracy rate of 95.7%. Mclaughlin et al. [17] used the CNNs to learn the features of malware applications from the original Opcode sequence; the study reported a recall rate of 96.29%. Mcwilliams et al. [18] utilized the permission information of an application as the data feature, developed and analyzed an active machine learning method that is based on Bayesian classification, and demonstrated the method’s high-precision detection capabilities.

In dynamic detection, the application must be installed, and information such as system calls and network traffic must be obtained. Early researchers lacked knowledge of malware application behavior patterns and thus processed the resource usage of mobile phones as the basis for detection. With the study of dynamic information, researchers obtained the highly representative dynamic features of multiple applications. Martinelli et al. [5] established a system call-based detection application on a CNN and achieved an accuracy rate of 85% to 95%. Vinod et al. [6] studied several dynamic feature selection algorithms for system calls. Liang et al. [7] regarded system calls as feature sequences and designed an end-to-end malware recognition model with an accuracy rate of 93.1%. Ke et al. [19] used a random forest algorithm to identify and classify malware behaviors in behavior logs and reported an average accuracy rate that reached 96.8%. Dynamic detection exhibits high detection accuracy in dealing with the evolution of Android malware based on semantic analysis of behavioral features; however, this detection method consumes a lot of resources, such as computing resources and storage resources, resulting in low detection efficiency. Kumar et al. [20] proposed a blockchain and machine learning-based malware detection mechanism for IOT devices. The machine learning mechanism uses clustering and classification algorithms to automatically extract malware information and store it in the blockchain. The framework uses the blockchain to store the real information of the extracted features in a distributed malware database to improve the speed and accuracy of runtime malware detection. While the application of blockchain technology in such areas can solve the storage resource problem, the computational complexity is high. Therefore, to perform efficient and accurate detection of Android malware, dynamic analysis of API call views and static features of application software is required.

Android malware detection methods based on static detection use many types of features. DREBIN [21] applies eight static features, namely the following: requesting permissions, hardware access resources, components and services used by the application, filtering intent, app-defined API calls, defined service permissions, highly sensitive API calls, and network resources. Zhang et al. [22] used four features, namely the following: permissions, API lists, components, and strings. Ke et al. [19] indicated that permissions and APIs are the most commonly used features in many application detection methods that are based on static features. Researchers use different classification algorithms for static feature detection. Wang et al. [23] used three traditional machine learning algorithms: support vector machines, random forests, and K nearest neighbors (KNN). Arshad et al. [24] used the random forest algorithm to establish a three-level hybrid malware detection model. Kumar et al. [25] proposed two machine learning supported methods for static analysis of Android malware. The first approach is based on static analysis to discover malicious features through probability statistics, which reduces the uncertainty of information. The second approach is to analyze the existing data set and propose a feature extraction algorithm. Both methods transform high-dimensional data into low-dimensional data to reduce the uncertainty of dimensionality and feature extraction. Classification algorithms come in many types with different processing effects on different problems. Additionally, a single feature or a combination of multiple features affects the detection accuracy of a model.

The above research shows that using a good feature selection method to find the best feature subset can reduce the processing complexity and improve the performance of the learning model by eliminating unimportant features. For example, Aonzo et al. [26] selected a small feature subset from the list of the most important features and proved that these small subsets of features are sufficient for good classification. Therefore, when detecting complex and intelligent malware applications, we use multiple dimension features information to reveal the different functions of the applications in various aspects. Also, we analyze the representative information required by the application functions and design a feature selection algorithm, called the FDS algorithm. The algorithm excludes atypical features and follows the original rules of features. Then, we use a combination of machine algorithms and weight measurement to detect emerging complex and intelligent Android malware, thereby improving the efficiency and accuracy.

3 Proposed Method

To achieve the detection of Android malware, we follow a three-step process for each application file (Fig. 1). First, we use two tools to collect the static features of the Android application (Section 3.1). Second, we analyze the functional characteristics of the applications in the dataset and measure the related features, such as API calls (Android system API calls, third-party API calls, custom API calls), usage permissions, API calls to measure static functions of the application, opcodes, and intents (service, receivers), and subsequently analyze the related features (Section 3.2). Finally, we use the feature selection algorithms (FDS), weight measurement, and the related machine learning algorithms to detect Android malware.

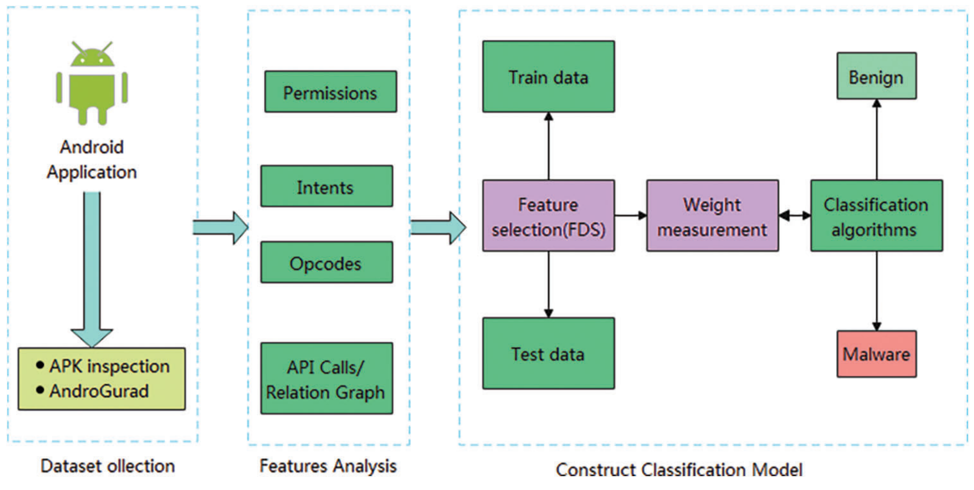

Figure 1: Architectural overview of Android malware detection

3.1 Dataset Collection

Malware detection is inseparable from the support of effective malware sample sets. The commonly used malware sample sets are mainly divided into two categories according to their sources. One category comprises malware sample sharing websites, such as Contagio [27] and VirusShare [28]. Classes are commonly used datasets that contain family information; examples include the Genome dataset [29], Drebin dataset [21] and the RmvDroid dataset [30]. Malware sample sharing websites provide many malware samples for analysis that are used by researchers. These platforms greatly promoted the development of malware detection technology.

In the current work, the DroidBox tool provided in [31] is used, and some monitoring and analysis functions of the tool are adopted to obtain the features of 22,000 real malware and benign samples. For example, Androguard [32] is python-based and maps the classes.dex files, classes, methods, etc. in the APK file to python objects. It not only provides reverse engineering of APK files, malware detection, and threat assessment but also visualizes program behavior. This function is widely utilized and is an important auxiliary tool for analyzing APK.

3.2 Feature Analysis

Extensive research that explored Android malware detection exists; however, most of the existing studies only used limited types of features to detect Android malware. Each feature type represents only few application attributes. In the current study, we use multiple feature information to reflect the different features of malware in various aspects. Preprocessing methods are employed to extract and process multiple types of features. The use of different features to evaluate the differences between two samples is also analyzed. Users can download related APK for the Android system according to their own needs; therefore, for the original file APK of the application, we used the APK disassembly tool Androguard to obtain the manifest.xml file and classes.dex. According to these two files, we extracted the following features: permission, intent filter, opcode, and API call.

• Permission: Android security architecture stipulates, in default, that an application exhibits no permission to perform any operation that results in an adverse effect on other applications, operating systems, or users, which include the following: reading and writing the user's private data (such as contacts or email), reading and writing other App files, performing network access, keeping the device awake, etc. If users want to use these protected device functions, developers should add one or more <uses-permission> tags in AndroidManifest.xml. The permission statement reflects the user's use of the application's resources and information.

• Intent filter: Intent is an abstract description of the operation to be performed. It is used to request operations in other components (i.e., activities, services, broadcast receivers) and complete the interaction between components. The intent filter is utilized to assist with the communication between the various Android components. For example, call the startActivity() of the Activity instantiation object to start an activity, or pass it to all interested BroadcaseReceivers by broadcaseIntent(), or start a background service by startService()/bindservice(). Primarily, the intent filter is used to start the activity or service (and carry parameter information that must be passed). In short, the intent filter exhibits the function of activating components and carrying the data.

• Opcode: Opcodes are mainly concentrated in the smali code of malware and benign software.

• API call: API calls reflect the sensitive data and resources involved in the process of interaction between the user's application and the system. We can gain insight into the user's intentions through the view analysis of API calls/Relation Graph.

Android malware reflects that the attackers realize their malicious operations through specific API calls. Each permission can correspond to multiple API operations, so the specific operation of the application can be expressed to a certain extent through the detection of permissions. The permission feature is easy to obtain, which is another reason why this feature attracted attention. It is the master of the static detection collection, basically referring to all the features in the APK, using the basic idea of "I want it all" and training all permissions as features. However, considering that too many permission features exist may not make the detection effect better. In contrast, it may also reduce the efficiency of detection. Therefore, the current research based on permissions is generally used in combination with the number of features and other factors. For instance, filtering intentions, network measurement or CPU Usage, construct combined features in Drebin, and establish SVM detection models for the classification of malware families.

Intent itself is a passive data structure that holds the abstract description of the operation to be carried out by the user, which may be regarded as the key information for classifying sample behavior. MAST [33] is a machine learning approach based on the multiple correspondence analysis for automatically identifying malicious applications from various Android markets. The tool is aimed toward ranking apps for inspection by a human security analyst, thereby giving priority to suspicious applications. In [34], Andro-Dumpsys was proposed as a novel and feature-rich hybrid anti-malware system, which combines malware-centric attributes with intent-based features for malware detection and classification.

Opcode is utilized to decompile the application through the APK tool to obtain the decompiled APK file combination of the application, then traverse the files corresponding to each application (smali files, decompiled files), and extract the operations in the smali files Code, integrating the opcodes in multiple smali files into the same file to get the opcode application sequence. Deep learning is applied to opcode to analyze the execution trajectory of user behavior at the kernel layer. Essentially, it is used to build a detection model to identify confusing Android malware.

Whether based on static analysis-based malware detection or dynamic analysis-based malware detection, application API calls is a key factor. For example, Wu et al. [35] track and analyze the API call trajectory of different applications, and they count the frequency of API calls to detect the establishment of Android malware. DroidAPIMiner [36] focuses on the dynamic analysis of high-risk API calls generated during the execution of the application, including the package, class, method, and call parameters of the API. Gascon et al. [37] proposed to establish a Markov chain based on API calls, trying to analyze the logical relationship between API calls, to identify the intention of the attacker.

However, while machine learning (ML) classifiers have been widely used for Android malware detection, the application of machine learning classifiers also faces a problem. With malware evolves, the performance of such classifiers degrades significantly over time.

Therefore, we refer to the literature [38] to construct an API relationship view based on the information provided by the application. Each node in the graph represents an entity, such as: method, class, package, and permission; these four entities together provide enough capability to capture the internal relationships between API. Each edge represents the relationship between two entities, such as the reference relationship between packages and classes involved in the API. Then it extracts API semantics from the relation graph by converting each API entity into an embedding and grouping similar APIs into clusters. The extracted API semantics in the format of API clusters can be further used in existing Android malware classifiers to detect evolved malware, thus slowing down aging.

3.3 Feature Selection

Machine learning is usually applied to utilize the statistical features of multiple dimensions. These features may be unrelated and may show interdependence, thereby leading to the following consequences: i) the larger the number of features, the longer the feature analysis and model training will take; ii) the larger the number of features, the easier the “dimension disaster” will occur, the more complex the model will become, and the greater the decrease in the model’s promotion ability will be. Feature selection can eliminate irrelevant or redundant features, thereby reducing the number of features, improving model accuracy, and reducing the runtime.

Therefore, this study designs a novel FDS algorithm to eliminate atypical features and follow the original rules of features for feature selection. The algorithm is analyzed and designed from the most essential features, which is the frequency of each feature in benign applications and malware applications. The atypical and typical features of the dataset are as follows: typical benign features (definition 2), typical malware features (definition 3), and atypical features (definition 4). Therefore, using the ratio of the frequency difference between the features in the benign and malware applications to the total number of samples as the criterion for evaluating a feature, this study derives the following design feature evaluation formula:

$$\mathrm{S}_{\mathrm{evaluation}} = \frac{|N_b - N_m|}{(T_b + T_m)} \tag{1}$$

N_{m} represents the number of malware applications containing feature f_i, N_b represents the number of benign applications containing feature f_i, T_m represents the total number of malware applications, and T_b represents the total number of benign applications. The feature evaluation formula calculates the modulus of the difference between the number of occurrences of the feature, which is a benign application or a malware application. Then, it divides the absolute value of the difference by the total number of samples. The formula features are as follows: (1) the score of each feature can be calculated and is used as the basis for feature selection, and (2) the formula can effectively screen atypical features. The following observations are made according to the analysis of the atypical and typical features and the calculation of formula: (1) the $\mathrm{S}_{\mathrm{evaluation}}$ value of atypical features is relatively small and the $\mathrm{S}_{\mathrm{evaluation}}$ value of typical features is relatively large. Therefore, atypical features can be effectively removed, and other effective features can be selected. At the same time, the formula is designed from the perspective of the most primitive law of features, and it will not interfere with the proportion of typical malware features.

To effectively filter features, this work provides the following relevant definitions:

Definition 1 (Benign features) for a feature f_{i}, the number of times it appears in benign applications and malware applications is recorded as the two-tuple $cont_i = (N_b,\ N_m)$. If $\mathrm{N}_b > N_m$, then the feature is called a benign feature.

Definition 2 (Malware features) for a feature f_{i}, the number of times it appears in benign applications and malware applications is recorded as a two-tuple $cont_i = (N_b,\ N_m)$. If $\mathrm{N}_m > N_b$, then the feature is called a malware feature.

Definition 3 (Typical benign features) If the feature f_i is a benign feature, that is, $cont_i = (N_b,\ N_m)$, where $\mathrm{N}_b > N_m$, and meets the following:

$$\frac{(N_b - N_m)}{(T_b + T_m)} \geq \alpha \tag{2}$$

Then the typical feature is called benign.

Definition 4 (Typical malware features) If the feature f_i is a malware feature, that is, $cont_i = (N_b,\ N_m)$, where $\mathrm{N}_m > N_b$, and meets the following:

$$\frac{(T_b + T_m)}{(T_b + T_m)} \geq \alpha \tag{3}$$

Then the typical feature is called malware.

Definition 5 (Atypical features) If the feature f$_i$ satisfies one of the following two conditions, it is called an atypical feature:

(1) $cont_i = (N_b,\ N_m)$, and

$$\begin{cases} \dfrac{N_b}{(T_b + T_m)} \leq \alpha, N_m \approx 0 \\ \dfrac{N_m}{(T_b + T_m)} \leq \alpha, N_b \approx 0 \end{cases} \tag{4}$$

That is, the feature appears less frequently in benign applications and almost never appears in malware application, or it appears less frequently in malware applications and almost never appears in benign applications.

$$\frac{(N_b - N_m)}{(T_b + T_m)} < \alpha \tag{5}$$

Hence, this feature appears in large numbers in benign and malware applications, but the absolute value of the difference in the numbers of occurrences is small.

According to the above definition, the idea of the FDS algorithm is as follows: first, it counts the total number of samples and the number of times each feature appears in benign and malware applications; then, it calculates the $S_{evaluation}$ value of each feature according to Eq. (1) and follows definition 3 to definition 5 for feature selection. The pseudo code of the algorithm is as follows:

Algorithm 1: Feature selection algorithm-FDS

```
Input: feature set F, threshold α.
Output: new feature set F'
1: Count Malware $T_m$;
2: Count Benign $T_b$;
3: I = 1;
4: For i in length (F.size) do
5: Count in Malware $f_i$ is $N_m$;
6: Count in Benign $f_i$ is $N_b$;
7: $N_{bm} = |N_b - N_m|$;
8: $T_{bm} = T_b + T_m$;
9: $S_{evaluation} = N_{bm}/T_{bm}$;
10: If $S_{evaluation} \geq \alpha$ then
11: F' = $f_i$;
12: end if
13: end for
```

Lines 1 to 2 count the number of malware and benign samples; lines 3 to 10 execute the loop, count the number of times each feature appears in benign and malware applications, and then calculate the $S_{evaluation}$ value of each feature. According to definitions 3 to 5, the features that satisfy the conditions are selected. Different threshold ɑ values produce different numbers of features (refer to Section 4.3 for the summary of threshold judgment). The algorithm description indicates that the FDS algorithm must count only the number of each feature appearing in benign and malware applications, calculate the $S_{evaluation}$ value of each feature, and perform feature selection according to related definitions. Relative to the chi-square test, information gain, and other algorithms, the FDS algorithm exhibits a simple feature evaluation; its feature selection basis is reasonable, its calculation consumption is low, and its time cost is linear. Moreover, the increase in the runtime of the FDS algorithm is fixed to be proportional to the increase in the sample input size. The algorithm steps indicate that the number of executions of lines 1 to 2 is not associated with the size of the sample and that the executions are performed only twice. The number of executions in lines 3 to 10 varies with the sample size n; it is $5 * n$ at least and $6 * n$ at most. Therefore, the computational complexity of the algorithm is only related to the number of samples, and the time complexities of the best and worst cases of the algorithm are $O(n)$.

3.4 Weight Measurement

Assume that x features exist in the expected feature set: $f_i = \{1, 2, \ldots, x\}$. As the importance values of features differ, each feature is assigned a weight (w_i) on this basis of the nature of feature f_i. Accordingly, the feature vector transforms into the weighted feature vector, which is from the original of the application. We multiply each feature by its corresponding weight and then use the classifier for fine-tuning to find the optimal weight.

3.5 Classification Detection Algorithm

Currently, most malware detection methods mainly focus on machine learning algorithms. Their data inputs include malware sample datasets and benign sample datasets. These methods often divide a dataset into two parts: training set and test set. The data in the training set are used to learn and build a classifier, and the data in the test set are used to verify the performance of the method. Through feature extraction, each sample in the training set is expressed as a feature vector of fixed dimensions $x = < x_1, x_2, \ldots, x_k >$, where k represents the vector dimension, which denotes the number of features. As the sample data in the training set are known to be malware, the corresponding label information is added after the vector. For example, label 0 indicates that the sample is a benign sample, and label 1 indicates that the sample is malware. Existing machine learning algorithms, such as support vector machines and random forests, are then employed. The feature space of the training samples is learned, and the corresponding classifier is constructed. For each sample in the test data, the feature vector is used as an input in the classifier, and the returned label result indicates whether the sample has malware behavior [39]. For malware detection, usually metric applications exhibit the following: false positive rate (FPR), true positive rate (TPR), recall rate, accuracy rate, F1-score value [40]. The main purpose of each malware detection method is to obtain a high TPR and a low FPR.

Therefore, in this paper, we also adopt several popular machine learning algorithms for training datasets. These methods use feature engineering to convert a program into a feature vector for representation. Then, it builds a classifier to detect and classify malware quickly and accurately. The method can deal with different malware code variants to a certain extent and entails minimal time cost.

4 Experiment and Result Analysis

4.1 Experiment Setup

To verify the applicability of our model to Android malware detection, we use a GPU server with a core CPU capacity of 64 GB, RAM memory of 512 GB, and an operating system running Ubuntu 18.04. The experimental environment is based on Python 3.5, and the classifier is applied in [41]. To validate our proposed approach, we use the OmniDroid dataset [42], which comprehensively covers the diversity of the Android application. For the data, 70% are used as the training data, and 30% are used as the test data. Cross validation is performed ten times. Based on the average precision, the average recall rate, and the average AUC value determine the optimal parameters for the malware detection model.

4.2 Threshold Selection

To verify whether the FDS algorithm achieves the design purpose, we select five features from 22,000 features for calculation and analysis. Tab. 1 shows the feature information of the five features (some typical features are among the 11,000 features) and the calculation results of the FDS algorithm. The detailed calculation process is described as follows. i) The number of malware samples and benign samples are counted (T_m = 11,000 and T_b = 11,000. ii). The number of times N_m and N_b of various types of features appear in the malware and benign samples are counted. iii) The $S_{evaluation}$ value of each feature is calculated according to Eq. (1). iv) According to the feature selection criteria, features with $S_{evaluation}$ value greater than or equal to α are selected.

Table 1: Example data and FDS calculation analysis

Feature Information	N_m	N_b	$S_{evaluation}$
READ_PHONE_STATE	8823	4084	0.5687
Android.content	10763	10938	0.2145
Android.intent.action.BOOT_COMPLETED	4185	2380	0.08
New-instance	10994	10995	0.00005
Service	3587	5758	0.0987

The $S_{evaluation}$ value is obtained by calculating the features of each dimension of the sample through the FDS algorithm. Tab. 1 shows the difference between the features of the benign application and those of the malicious application. The Android malware detects the model whose $S_{evaluation}$ value is less than the threshold α and removes it. Considering space limitations, we only list the $S_{evaluation}$ values of some features.

Given the choice of threshold a, we attempt the following approach. i) We start with a low threshold, such as 1e-4. ii) We reduce our features by using the selection from the model fit and transform. iii) We calculate the accuracy of the metrics for our estimator (Random forest classifier in our case) for selected features. iv) We increase the threshold and repeat all steps starting from point 1. Using this approach, we can estimate the best threshold for our data and estimator. Additionally, Fig. 2 shows that the number of features is about 7,000 and that the accuracy is optimal. At this point, the best threshold is α = 0.09.

4.3 Weight Analysis

In this section, we analyze the weights of features. In the sample training stage, we use feature selection algorithms to find the important features of the difference between malware samples and benign samples and to reduce the feature dimension and computational complexity. Then, we input the random forest classifier, debug the hyperparameters of the classifier, and obtain the optimal weight of each feature according to the detection results. In Tab. 2, we listed only the top ten features with a weight value greater than 0.029.

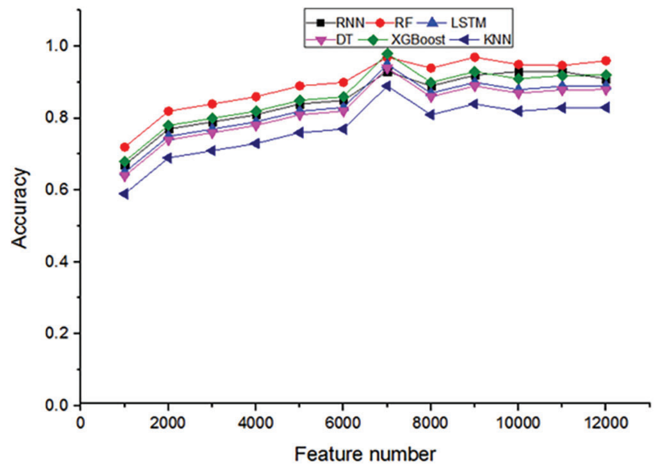

Figure 2: Accuracy comparison of the feature number of six classifiers

Table 2: Ten feature weights of the data set

Feature	Weighted value
Android_telephony_TelephonyManager_getDeviceId	0.0495
Android.permission.SEND_SMS	0.0456
GetSubscriberId	0.0438
GetDeviceId	0.0406
Android_util_Bsed64_decode	0.0399
Android.intent.action.BATTERY_CHANGED	0.0361
Android_util_Base64_encode	0.0348
Android.permission.GET_TASKS	0.0334
Android.permission.READ_PHONE_STATE	0.0298
Android.permission.WAKE_LOCK	0.0296

4.4 Result Analysis

In the first experiment, we provide all the features (22,000 features) to six classifiers, namely, (K-Nearest Neighbor) KNN, decision tree, (eXtreme Gradient Boosting) XGBoost, (Recurrent Neural Network) RNN, (Long Short-Term Memory) LSTM, and random forest. The accuracy of malware detection on the test data reaches 89% (Tab. 3).

Table 3: Comparison of results of six classifiers

Classifier	Accuracy	Precision	Recall	F1-Score
KNN	0.84	0.86	0.83	0.84
Decision Tree (DT)	0.85	0.88	0.86	0.87
Random Forest (RF)	0.87	0.89	0.87	0.88
XGBoost	0.89	0.92	0.94	0.93
RNN	0.88	0.93	0.92	0.92
LSTM	0.86	0.91	0.95	0.93

In the second experiment, we apply the features with a length of 7,000 and use weights to measure them. The accuracy rate increases by 99%. The results are shown in Tab. 4.

Table 4: Comparison of results of six classifiers for feature selection and weight measurement

Classifier	Accuracy	Precision	Recall	F1-Score
KNN	0.93	0.95	0.93	0.94
Decision Tree	0.93	0.94	0.92	0.93
Random Forest	0.99	0.99	0.98	0.98
XGBoost	0.94	0.94	0.95	0.95
RNN	0.92	0.93	0.89	0.91
LSTM	0.93	0.95	0.93	0.94

Our method compare with other methods in [13,14,25,26,42]. We realize the joint optimization of feature selection and weight measurement. The experimental results are shown in Tab. 5. The accuracy of the proposed method appears to be superior.

Table 5: Comparison of results of classifiers for feature selection and weight measurement

Classifier	Accuracy	Precision	Recall	F1-Score
Zhang [13]	0.95	0.95	0.93	0.94
Arora [14]	0.96	0.96	0.95	0.95
Kumar [25]	0.98	0.98	0.98	0.98
Aonzo [26]	0.98	0.98	0.98	0.98
Martín [42]	0.90	0.90	0.89	0.89
our method	0.99	0.99	0.98	0.98

Fig. 3 illustrates the comparison of the ROC curves by of different methods. The AUC value of the proposed method is 0.98. The AUC value of the former method thus exceeds that of the latter by 0.03.

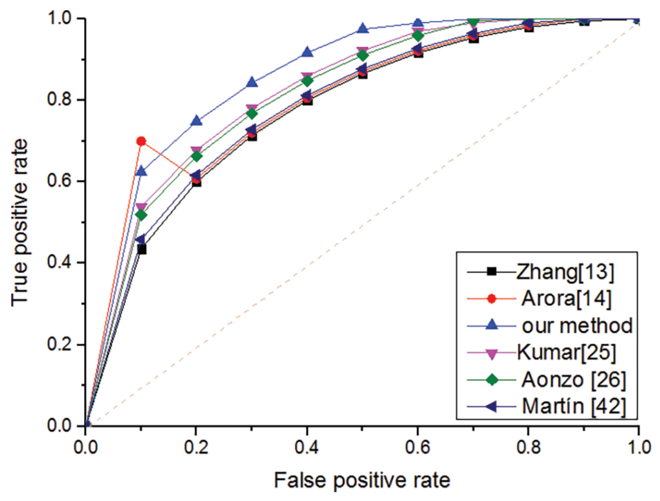

Figure 3: ROC curve for our method of different methods

4.4.1 Sustainable of Android Malware Detection

Typically, ML-based classification methods explore a variety of features, computed through static, dynamic, or hybrid application analysis. In most cases, these features are based on the application's use of permissions or APIs. However, due to the evolution of Android malware attack strategies, classifiers built based on these features may not be sustainable - they need to be constantly retrained for future use, or their performance may degrade over time.

In this paper, we extensively evaluate the sustainability of ML-based malware classifiers to understand how to design more sustainable malware classification methods. Therefore, to validate the sustainability of our proposed Android malicious detection model, we collect application samples from VirusTotal from early 2016 to 2018 (21694 samples). At the same time, we conducted two studies. In the same-period study, we assess the performance of each technique with training and testing apps developed in a same period of time (in the minimal unit of year). For each given mixed dataset, we randomly selected a third of samples from each class (malware or benign) and reserved them as unseen/novel samples for training, and the remaining for testing.

As shown Tab. 6, the test results show that it has the highest malware detection precision rate 0.99 on the latest (2016) dataset, the accuracy rate of the 2017 test set is 0.98, the precision rate of the 2018 test set is 0.96, and the average detection precision rate is 0.98. Although the classification performance seems to gradually decrease with the growth of the years, the decline is not obvious. What this implies is that the features seem to be better robust against the evolution of malware than for differentiating benign apps from all kinds of malware as a whole.

Table 6: Performance for Android malware detection

Period	Benign	Malware	Precision	Recall	F1-Score
2016	5210	4270	0.9926	0.9913	0.9919
2017	4815	2589	0.9754	0.9712	0.9733
2018	5420	3890	0.9645	0.9625	0.9635

4.4.2 Efficiency of Android Malware Detection

In order to understand how effective FDS is for feature selection, we investigated 6 alternative approaches to feature selection: 1 full dataset and 5 different subsets. Our investigation shows that FDS works best on 128 metrics that differ significantly between benign and malicious applications. To avoid any bias caused by the choice of machine learning algorithms, we also experimented with six other machine learning algorithms: Random Forests, Decision Trees, KNN, RNN, LSTM, and XGBoost. We found that FDS performed best when using Random Forests.

We compared the five new feature sets with FDS by separately feeding them to the Random Forest algorithm for unified malware detection. The evaluation results are shown in Fig. 4.

Interestingly, FDS worked best, even better than full for all metrics. The reason is that the additional other nondiscriminatory metrics may provide noisy information and mislead the classifier to perform poorly. Therefore, FDS is more desirable, because it achieves better effectiveness with fewer features. This experiment also demonstrates that more features do not necessarily improve classification capability. Instead, they can worsen the capability especially if they seem to be nondiscriminatory. Therefore, the experimental results show that the default feature set of FDS, exhibits the best effectiveness compared with other five alternative feature sets, because it balances well the diversity and relevance of features.

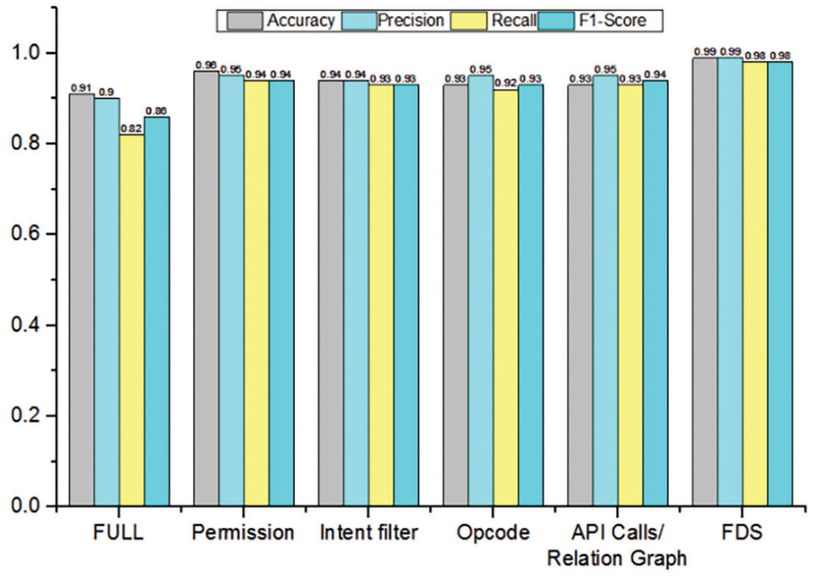

Figure 4: FDS's effectiveness with five alternative feature sets

5 Conclusion and Future Work

In this work, we propose the FDS algorithm and weight measurement for Android malware detection based on diverse application features. The objectives are to solve the shortcomings of the existing feature selection algorithms in detection and to filter other effective features. We use weight measurement to optimize the detection accuracy of classifiers and improve the accuracy of detection. The theoretical analysis and experimental results show that the FDS algorithm and weight measurement are effective and feasible. Additionally, these approaches offer their own advantages over other existing malware detection models. However, their limitation regarding atypical features is slightly rough and may cause some typical features to be limited to atypical features. Their accuracy in detecting unknown malware is also slightly insufficient. Future directions worthy of further discussion include the following: i) research on deep learning detection methods for protection against attacks and ii) other malware behavior detection based on Android, such as the detection of phishing websites under hidden links.

Acknowledgement: The authors would like to thank the editor and the anonymous referees for their constructive comments.

Funding Statement: This work was financially supported by "the National Key Research and Development Program of China" (No.:2017YFB0801900).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] IDC, 2019. [Online]. Available: https://www.idc.com/promo/smartphone-market-share.
[2] m.qq.com, 2019. [Online]. Available: https://m.qq.com/security_lab/news_detail_517.html.
[3] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian *et al.,* "Constructing features for detecting Android malware applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[4] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao *et al.,* "Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware," in *Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security*, pp. 757–770, 2020.

[5] F. Martinelli, F. Marulli and F. Mercaldo, "Evaluating convolutional neural network for effective mobile malware detection," *Procedia Computer Science*, vol. 112, no. 112, pp. 2372–2381, 2017.

[6] P. Vinod, A. Zemmari and M. Conti, "A machine learning based approach to detect malware Android apps using discriminant system calls," *Future Generation Computer Systems*, vol. 94, no. 1, pp. 333–350, 2019.

[7] H. Liang, Y. Song and D. Xiao, "An end-to-end model for Android malware detection," in *Proc. of the IEEE Int. Conf. on Intelligence and Security Informatics (ISI)*, Beijing, China, pp. 140–142, 2017.

[8] K. Xu, Y. Li, R. Deng, K. Chen and J. Xu, "DroidEvolver: Self-evolving Android malware detection system," in *2019 IEEE European Sym. on Security and Privacy (EuroS&P)*, Stockholm, Sweden, pp. 47–62, 2019.

[9] A. Rahim Khan and M. K. Jain, "Feature point detection for repacked Android apps," *Intelligent Automation & Soft Computing*, vol. 26, no. 6, pp. 1359–1373, 2020.

[10] Y. Li, G. Xu, H. Xian, L. Rao and J. Shi, "Novel Android malware detection method based on multi-dimensional hybrid features extraction and analysis," *Intelligent Automation & Soft Computing*, vol. 25, no. 3, pp. 637–647, 2019.

[11] M. K. Alzaylaee, S. Y. Yerima and S. Sezer, "DL-Droid: Deep learning based Android malware detection using real devices," *Computers & Security*, vol. 89, pp. 101663, 2020.

[12] A. P. Felt, K. Greenwood and D. Wagner, "The effectiveness of application permissions," in *Usenix Conf. on Web Application Development*, Portland, USA, pp. 7, 2011.

[13] Y. Zhang, Y. Sui, S. Pan, Z. Zheng, B. Ning *et al.,* "Familial clustering for weakly-labeled Android malware using hybrid representation learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3401–3414, 2020.

[14] A. Arora, S. K. Peddoju and M. Conti, "PermPair: Android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2020.

[15] R. Kumar, X. Zhang and R. U. Khan, "Research on data mining of permission-induced risk for android IoT devices." *Applied Sciences*, vol. 9, no. 2, pp. 277, 2019.

[16] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *Int. Joint Conf. on Neural Networks (IJCNN)*, Alaska, USA, pp. 1871–1878, 2017.

[17] N. McLaughlin, J. M. delRincon, B. J. Kang, S. Yerima, P. Millerx *et al.,* "Deep Android malware detection," in *Proc. of the Seventh ACM Conf. on Data and Application Security and Privacy*, Scottsdale Arizona, USA, pp. 301–308, 2017.

[18] G. Mcwilliams, S. Sezerand and S. Y. Yerima, "Analysis of Bayesian classification based approaches for Android malware detection," *IET Information Security*, vol. 8, no. 1, pp. 25–36, 2014.

[19] D. Ke, L. Pan, S. Luo and H. Q. Zhang, "Android malware behavior recognition and classification method based on random forest algorithm," *Journal of ZheJiang University (Engineering Science)*, vol. 53, no. 10, pp. 2013–2023, 2019.

[20] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar *et al.,* "A multimodal malware detection technique for Android IoT devices using various features," *IEEE Access*, vol. 7, pp. 64411–64430, 2019.

[21] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck *et al.,* "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. of the 21st Annual Network and Distributed System Security Sym. (NDSS)*, San Diego, California, 2014.

[22] Y. Zhang, Y. Yang and X. Wang, "A novel Android malware detection approach based on convolutional neural network," in *Proc. of the 2nd Int. Conf. on Cryptography, Security and Privacy*, Guiyang, China, pp. 144–149, 2018.

[23] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu *et al.,* "DroidEnsemble: Detecting Android malware applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.

[24] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song *et al.,* "SAMADroid: A novel 3-level hybrid malware detection model for Android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[25] R. Kumar, Z. Xiaosong, R. Khan, J. Kumar and I. Ahad, “Effective and explainable detection of Android malware based on machine learning algorithms,” in *Proc. of the 2018 Int. Conf. on Computing and Artificial Intelligence*, Sanya, China, pp. 35–40, 2018.

[26] S. Aonzo, A. Merlo, M. Migliardi, L. Oneto and F. Palmieri, “Low-resource footprint, data-driven malware detection on Android,” *IEEE Transactions on Sustainable Computing*, vol. 5, no. 2, pp. 213–222, 2017.

[27] Contagio Mobile, “Mobile malware mini dump,” 2013. [Online]. Available: http://contagiominidump.blogspot.com.

[28] VirusShare, [Online]. Available: https://virusshare.com/.

[29] Complete Genomics, [Online]. Available: https://www.completegenomics.com/public-data/.

[30] H. Wang, J. Si, H. Li and Y. Guo, “RmvDroid: Towards a reliable Android malware dataset with app metadata,” in *Proc. of the 16th Int. Conf. on Mining Software Repositories (MSR)*, Montreal, pp. 404–408, 2019.

[31] GitHub, [Online]. Available: https://github.com/pjlantz/droidbox.

[32] pypi.org, [Online]. Available: https://pypi.org/project/androguard/.

[33] S. Chakradeo, B. Reaves, P. Traynor and W. Enck, “Mast: Triage for market-scale mobile malware analysis,” in *Proc. of the Sixth ACM Conf. on Security and Privacy in Wireless and Mobile Networks*, Budapest, Hungary, pp. 13–24, 2013.

[34] J. Jang, H. Kang, J. Woo, A. Mohaisen and H. K. Kim, “Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information,” *Computer & Security*, vol. 58, no. 3, pp. 125–138, 2016.

[35] D. Wu, C. Mao, T. Wei, H. Lee and K. Wu, “DroidMat: Android malware detection through manifest and API calls tracing,” in *Proc. of the Seventh Asia Joint Conf. on Information Security (AsiaJCIS 2012)*, Tokyo, Japan, pp. 62–69, 2012.

[36] Y. Aafer, W. Du and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” in *Int. Conf. on Security and Privacy in Communication Systems*, pp. 86–103, 2013.

[37] H. Gascon, F. Yamaguchi, D. Arp and K. Rieck, “Structural detection of Android malware using embedded call graphs,” in *Proc. of the Workshop on Artificial Intelligence and Security*, Berlin, Germany, pp. 45–54, 2013.

[38] X. Zhang, Y. Zhang and M. Zhong, “Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware,” in *Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security*, pp. 757–770, 2020.

[39] S. Zou, H. Sun, G. Xu and R. Quan, “Ensemble strategy for insider threat detection from user activity logs,” *Computers Materials & Continua*, vol. 65, no. 2, pp. 1321–1334, 2020.

[40] Towards Data Science, [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

[41] R. Vinayakumar, K. P. Soman, P. Poornachandran and S. Kumar, “Detecting Android malware using long short-term memory (LSTM),” *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1277–1288, 2018.

[42] A. Martín, R. Lara-Cabrera and D. Camacho, “Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset,” *Information Fusion*, vol. 52, no. 7, pp. 128–142, 2019.