IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

FAMD: a fast multifeature Android malware detection framework, design and implementation

HONGPENG BAI[12], NANNAN XIE[12], XIAOQIANG DI[123],AND QING YE[1]
[1]School of Computer Science and Technology. Changchun University of Science and Technology, Changchun China
[2]Jilin Province Key Laboratory of Network and Information Security, China
[3]Information Center of Changchun University of Science and Technology, China

Corresponding author: Nannan Xie (e-mail:xienn@cust.edu.cn).

This work was supported in part by the 13th Five-Year Science and Technology Research Project of the Education Department of Jilin Province under Grant No. JJKH20200794KJ, the Innovation Fund of Changchun University of Science and Technology under Grant No. XJJLG-2018-09, the fund of Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education (Jilin University) under Grant No. 93K172018K05.

ABSTRACT With Android's dominant position within the current ssmartphone OS, increasing number of malware applications pose a great threat to user privacy and security. Classification algorithms that use a single feature usually have weak detection performance. Although the use of multiple features can improve the detection effect, increasing the number of features increases the requirements of the operating environment and consumes more time. We propose a fast Android malware detection framework based on the combination of multiple features: FAMD (Fast Android Malware Detector). First, we extracted permissions and Dalvik opcode sequences from samples to construct the original feature set. Second, the Dalvik opcodes are preprocessed with the N-Gram technique, and the FCBF (Fast Correlation-Based Filter) algorithm based on symmetrical uncertainty is employed to reduce feature dimensionality. Finally, the dimensionality-reduced features are input into the CatBoost classifier for malware detection and family classification. The dataset DS-1, which we collected, and the baseline dataset Drebin were used in the experiment. The results show that the combined features can effectively improve the detection accuracy of malware that can reach 97.40% on Drebin dataset, and the malware family classification accuracy can achieve 97.38%. Compared with other state-of-the-art works, our framework achieves higher accuracy and lower time consumption.

INDEX TERMS Android Malware, CatBoost, Dalvik Opcode, Malware Detection

I. INTRODUCTION

IN the past ten years, advancements in mobile internet technology have changed the lifestyles of countless users and have also brought tremendous changes to the proceedures used in various industries, such as governments and enterprises. However, a series of security risks have arisen in mobile internet technology. Malware applications are hidden in smart terminals, such as information leaks, Trojan horses, push advertising, and pose threats to user privacy. International Data Company (IDC) [1], estimates, estimates that Android's smartphone market share will hover around 86% in 2020. In 2019, Kaspersky's report [2] showed that 3,503,952 malicious installation packages were found in its mobile terminal products. The number of attacks on mobile devices increased by 50% in 2019, from 40,386 in 2018 to 67,500 in 2019. In addition to spyware and Trojans in traditional network security, the usage of stalkerware on mobile devices is growing. Due to the large number of Android malware, the fast update speed and the constant emergence of new types of malware, it is always challenging to study how to effectively detect malware, reduce the detection time and improve the detection efficiency.

Android malware detection research mainly includes two aspects. The one is the detection features, which include requested permissions, API calls, Dalvik opcodes, and intercomponent communication. Different features or combined features are employed to detect malicious applications. The other is the detection methods, which use different machine learning methods or combinations of methods as classifiers, such as SVM (Support Vector Machine), KNN (K-NearestNeighbor), RF (Random Forest), and deep learning methods, to identify the different behavior patterns, and establish detection systems. The purpose of these studies is to improve the accuracy of malware detection with the hope that the methods are effective in practice.

In order to achieve the above purpose, we propose a fast Android malware detection framework, FAMD, that combines multiple features and uses a classification technique to detect malware and classifiy malware families. It uses permissions and Dalvik opcodes as classification features and further uses the FCBF algorithm to process the features to construct low-dimensional feature vectors. Finally, the machine learning framework CatBoost based on the gradient boosting decision tree is used as the classifier to perform the classification of malware. The main contributions of this paper are as follows.

- We propose a fast Android malware detection framework, FAMD, which includes three parts: constructing a malware detection feature set, preprocessing the features for dimensionality reduction, and performing malware detection and family classification on the processed features. The purpose is to improve the accuracy of malware detection while reducing the feature dimensions.
- In terms of feature preprocessing, because the sequences of Dalvik opcode are segmented by the N-Gram method, the feature dimension is high. We use the FCBF algorithm to reduce the dimension of the features from 2467 to 500.
- CatBoost is adopted as the classifier for the first time in Android malware detection and family classification. Compare with other GBDT-based methods, CatBoost can solve the problems of gradient bias and prediction shift, thus reducing the occurrence of over-fitting and improving the classification accuracy and the generalization ability of the model.

The rest of the paper is organized as follows. Section II introduces related research on Android malware detection. Section III presents the framework FAMD, and gives the implementation of each part of the framework. Then, section IV provides more details of our framework implementation and discusses FAMD's evaluation results. Finally, section V concludes the paper.

II. RELATED WORK

A. STATIC ANALYSIS AND DYNAMIC ANALYSIS OF MALWARE

The current research on Android malware detection can be divided into static analysis and dynamic analysis from the perspective of feature extraction. Static analysis refers to the analysis of the source code or the analysis of the features extracted from the source code. This method can analyze a program's source code without the application being executed. The static analysis includes decompilation, reverse analysis, pattern matching, and static system call analysis. The advantages of static analysis are low resource consumption, fast detection, and low real-time requirements, and the disadvantage is that the detection accuracy is relatively low.

The static analysis method is the most commonly used method in current research. Enck et al. [3]designed a set of security rules that use a signature-based approach to detect the application being evaluated. Saracino et al. [4] proposed a host-based malware detection system for Android devices called MADAM, which simultaneously analyzes and correlates features at the kernel level, application level, user level and package level to detect and prevent malicious behavior. Kim et al. [5] proposed framework usage permissions, strings, API calls, and other features to reflect the various characteristics of applications from various aspects. Their feature vector generation method consists of an existence-based method and a similarity-based method, and these are very effective in distinguishing between malware and benign applications, even though malware has many properties that are similar to those of benign applications. In addition, Zhang et al. [6] keep the abstracted API calls of function methods to form a set of abstracted API calls transactions and calculate the confidence of association rules between the abstracted API calls. Combine machine learning to identify the different behavior patterns, and establish a detection system. The framework of MaMaDroid [7] constructs the sequence obtained in the API call graph as a Markov chain to detect malware from the perspective of behavior.

Dynamic analysis covers a family of methods based on analyzing the runtime behavior of an application. It is usually necessary to run the application in a specific environment to monitor the application's access to the network, system calls, files and memory, information access patterns, and processing behaviors. Dynamic analysis judges the maliciousness of an application by analyzing whether the abovementioned behaviors are normal. The advantage of dynamic analysis is that it is not affected by code obfuscation and encryption and can analyze an application based on its malware-like behavior. However, it consumes system more resources and requires analysts with high technical capabilities, which is not conducive to large-scale applications in testing.

In 2014, Enck et al. [8] proposed a dynamic malware detection tool, TaintDroid, which labeled a variety of sensitive data, and then monitored the flow path of these contaminated sensitive data in a sandbox environment in real-time to determine whether the application had malicious behaviors of privacy data leakage. RansomProber [9] can infer whether the user initiated the file encryption operation by analyzing the user interface widgets of related activities and the user's finger movement coordinates and has a good effect in detecting encrypted ransomware. Cai et al. [10] used a variety of dynamic features based on method calls and inter-component communication (ICC) intents to achieve better robustness than static analysis and dynamic analysis, which depends on system calls. Yerima et al. [11] proposed and investigate approaches based on stateful event generation and provided much better code coverage, which leads to more accurate machine learning-based malware detection.

There are also some studies combine static and dynamic features. Yuan et al. [12] used the requested permissions, suspicious API calls, and dynamic behaviors with a total of 202 features to build a complete deep learning model. Tam et al. [13] proposed CopperDroid, which can capture operations initiated in Java and the native code execution to reconstruct the behavior of Android malware based on the automatic dynamic analysis system in VMI (virtual machine introspection).

B. FEATURES OF ANDROID MALWARE DETECTION

1) Permission Features

According to Android mechanism, every Android application runs in a limited-access sandbox. If an application needs to use resources or information outside of its own sandbox, the application has to request the appropriate permissions. Therefore, malware can be found by viewing the permissions declared in the AndroidManifest.XML file. Permission features can be divided into two types: official permissions and custom permissions. There are 166 official permissions defined by Android [14], such as android.permission.INTERNET, which allows applications to open network sockets. All developers can request these permissions. By defining custom permissions, an application can share its resources and capabilities with other applications. For example, if a developer wants to prevent certain users from launching an activity in an application, the developer can define custom permissions to achieve this. After the permissions are defined, they can be referenced as part of the component definition.

Android's permission features can reflect the behavior of the application in a certain sense. SANZ et al. [15]used the permissions as features and combined them with machine learning algorithms to detect Android malware. Wang et al. [16] systematically analyzed the risk of each individual permission and the risk of a group of collaborative permissions by employing machine learning techniques. Talha et al. [17] implemented a permission-based Android malware detection system, APK Auditor, which can achieve 88% accuracy and 92.5% specificity. Li et al. [18] proposed a multi-level data pruning method, SIGPID, which includes negative-rate permission sorting, association-rule permission mining, and support-based permission sorting to extract significant permissions strategically. When using SVM as the classifier,they can achieve over 90% of precision, recall, accuracy, and F-measure.

2) Dalvik Opcode Features

Dalvik is a virtual machine that was used to run Android applications in early Android systems. Every time it runs, it dynamically interprets a part of Dalvik bytecode as machine code. After Android 5.0, the Dalvik virtual machine (DVM) was replaced by Android Runtime (ART), but the compilation method of the underlying opcode is still compatible. These opcodes can reflect the behavior pattern of an application to a certain extent by means of the underlying machine code, so they are often used as static analysis features.

Jerome et al. [19] used N-Gram-based opcodes as features to detect malware and classify malware families. McLaughlin et al. [20] proposed using a deep convolutional neural network to automatically learn from the original opcode sequence, thereby eliminating the need for manually designed malware features. Zhang et al. [21] extracted several global topology features from the Dalvik opcode graph of each sample to represent malware. This method achieves better detection efficiency and robustness. Pektaş et al. [22] extracted the instruction call graph from a malicious application and derived an instruction call sequence to represent Android malware. The accuracy of the proposed malware detection method reached 91.42%. The model proposed by Egitmen et al. [23] extracts skip-gram-based features from the instruction sequence of an application, and a word embedded vector is generated for each unique opcode to realize the high-level representation of the opcode sequence, and it is used as the input feature of the detection model.

3) Other Features

In addition to permissions, Dalvik opcodes, Android malware detection features also include API calls [24]–[26], control flow graphs (CFGs) [27], component information, and hardware information. For example, the API Getdeviceid() can be used to access sensitive data and obtain the user's device ID. Therefore, it is also an effective method to detect the maliciousness of the application by studying the application's API calls. Zhang et al. [28] represented opcodes with a bi-gram model and represented API calls with a frequency vector. Then, they used principal component analysis to optimize the representations and to improve the convergence speed.

Since attacks may specifically evade detection by avoid using certain permissions or API calls, employing a single kind of feature in malware detection may affect the results. Some works used combined features to detect malware. Drebin [29] performed extensive static analysis and collected as many application features as possible. These features are embedded in a joint vector space, which can automatically identify typical patterns that represent malware. ICCDetector [30] uses captured interactions between components within an application or across application boundaries as features to detect malware. Alazab et al. [31] combined request permissions and API calls. Compared with benign applications, malicious applications call a different set of API calls. Malware usually requests dangerous permissions to access sensitive data more frequently than benign applications.

The purpose of adopting multiple types of features is to improve the detection effect. However, the combination of multiple features will increase the feature dimensions, making the classifier consume much time in the operation process and not detect efficiently. Therefore, reducing the time consumed in malware detection is also a focus of research. Applying appropriate feature selection methods to reduce the features' dimensions is a solution to this problem.

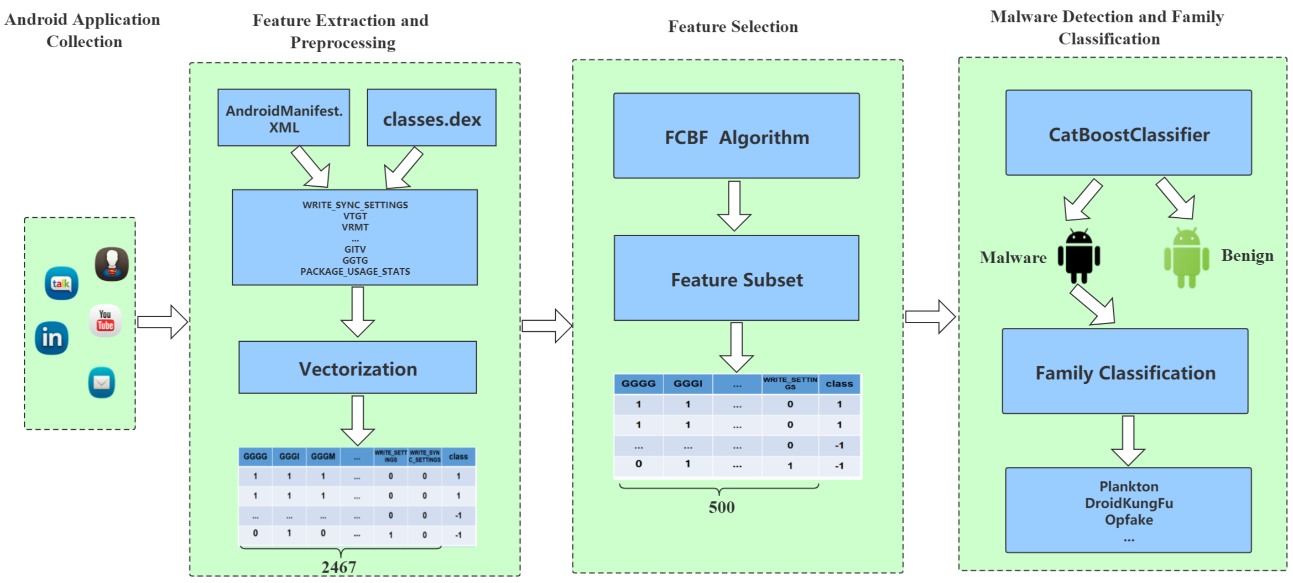

FIGURE 1. Framework of FAMD.

III. DESIGN AND IMPLEMENTATION OF FAMD

A. THE FRAMEWORK OF FAMD

FAMD is a fast Android malware detection framework based on multifeature combination. We combine the permission features and Dalvik opcode features from different levels of the operating system. To deal with the high dimensionality problem emerged after feature combination, the feature selection method is used to reduce the dimensionality, thereby reducing the classification consumption and achieving the purpose of being fast. Specifically, the FAMD framework will be divided into four parts: Android application collection, feature extraction and preprocessing, feature selection, malware detection and family classification, as shown in Fig. 1.

- **Android application collection.** The applications in this work are collected from an open source dataset and third-party markets. The collected samples are filtered by antivirus engine to ensure the purity of the maliciousness and normality. The details will be introduced in the experiment section.
- **Feature extraction and preprocessing.** We use decompilation tools to extract permissions and original opcode sequences from the AndroidManifest.XML file and the classes.dex file. Based on the N-Gram method, the specific length of the opcode sequence is extracted from the original opcode sequence, and the feature vector of each sample is constructed in combination with the permission features. Finally, we construct the feature matrix with each application as a row, and each of the extracted features as a column.
- **Feature selection.** Since the constructed feature vectors have high dimensionality which will result in high computational cost and overfitting, we employ the feature selection techniques to reduce dimensionality. FCBF algorithm is used to weight the features and construct the feature subset. The parameters and subset feature numbers will be decided by experiments.
- **Malware detection and family classification.** After distinguishing the malicious samples from benign ones, dividing malware into families is important to analyze the behaviors of malware. We use a machine learning algorithm based on the gradient boosted decision tree, CatBoost, as the classifier to detect malicious samples and classify the malware families. The evaluation metrics such as accuracy, precision, TPR, FPR are used to verify the effectiveness and performance of the framework.

B. FEATURE EXTRACTION AND PREPROCESSING

1) Extraction of Permissions

The purpose of setting permissions is to protect the privacy of Android users. Android applications must apply for permission to access sensitive user data (such as contacts and text messages) and certain system functions (such as camera and Internet). Depending on the function, the system may automatically grant permissions, or the user may be prompted to approve the request. Android divides permissions into four protection levels [32], which affect whether runtime permission requests are required.

- **Normal Permission.** This category of permissions covers situations in which the application needs to access data or resources outside its sandbox. These situations pose little risk to a user's privacy or the operation of other applications.

- **Dangerous permission.** Contrary to normal permissions, if an application should acquire this type of permissions, the user's private data will be exposed to the risk of tampering.
- **Signature permission.** This type of permissions is only open to applications with the same signature. Even if other applications know this open data interface and they also register permissions in the AndroidManifest.XML file, they still cannot access the corresponding data due to different application signatures.
- **SignatureOrSystem permission.** This permission category is similar to signature permission, but it not only requires the same signature but also requires similar system-level applications. This type of permissions is only used for prefabricated applications developed by general mobile phone manufacturers.

2) Processing of Dalvik Opcode

N-Gram [33] is a method based on statistical language models. It performs a sliding window operation of size N on the content of the text, forming a sequence of byte fragments of length N. Each byte segment is called Gram. The frequency of occurrence of all Grams is counted and filtered according to the preset threshold to form a key Gram list, which is the feature vector space of this text, and each element in the Gram list is a feature vector dimension.

The N-Gram model is based on the following hypothesis: the Nth word's appearance is only related to the previous $N-1$ words and is not related to any other words. The probability of an entire sentence occurring is the product of the probability of each word occurring. These probabilities can be obtained by directly counting the number of simultaneous occurrences of N words from the corpus.

In malware detection, the N-Gram method is often used to process malicious codes. The N-Gram features are usually extracted from the application opcode sequences. N is usually valued at 2, 3, and 4.

The current Dalvik instruction [34] set contains 230 instructions, including the "Move" instruction, "Invoke" instruction, "Return" instruction and so on. Existing studies have shown that methods based on N-Grams face the prospect of exponential growth in the number of unique N-Grams as the value of N increases. Therefore, in this paper, we simplify the opcodes by remove the irrelevant instructions, retain only the seven core instruction sets, and remove the operands. The seven instruction sets, M, R, G, I, T, P, and V, represent seven types of instructions, move, return, jump, judge, read data, store data, and call methods, respectively. The instructions are classified and described in Table 1.

According to the above mapping, we use the N-Gram to segment the opcode sequence extracted from the applications. The original opcode sequence: "move-object/from16, iget-object, invoke-virtual, goto, move-object/from16" is taken as an example. The "move-object/from16" sequence corresponds to the "M" instruction, "iget-object" corresponds to the "T" instruction, "invoke-virtual" corresponds to the "V" instruction, and "goto" corresponds to the "G" instruction. Therefore, the sequence is simplified as "MTVGM". Then the 3-Gram features of the sequence are {MTV},{TVG},{VGM}, the 4-Gram features are {MTVG}, {TVGM}, and the 5-Gram feature is {MTVGM}.

TABLE 1. Dalvik instruction mapping table

Instruction mapping	Instruction Set
M	"move"|"move/from16"|"move/16"|"move-exception"...
R	"return-void"|"return"|"return-wide"|"return-object"
G	"goto"|"goto/16"|"goto/32"
I	"if-eq"|"if-ne"|"if-lt"|"if-ge"|"if-gt"
T	"aget"|"aget-wide"|"aget-object"|"iget-wide"|"sget-object"...
P	"aput"|"aput-wide"|"iput-wide"|"iput-boolean"|"sput-object"...
V	"invoke-virtual"|"invoke-super"|"invoke-direct"...

3) Construction of Feature Vectors

Androguard [35] is a python-based Android analysis tool that can analyze an Android file structure through decompilation and extract static features. All permissions and opcode sequences of each application can be extracted from the AndroidManifest.XML file and the classes.dex file through Androguard. In this work, in order to limit the dimensionality of feature vectors and ensure the generality of extracted features, only 166 official permissions are extracted without considering custom permissions. For the extracted Dalvik opcode sequence, according to the above mapping table, an opcode sequence of a specific length is extracted. These features constitute the initial feature set.

The feature set is numerically simulated in the following way to construct feature vectors. Assuming an Android application a, the feature set constructed from all applications contains n features, and the feature set is represented by S; then, the feature vector of application a is represented by equation (1).

$$V_a = \{v_1, v_2, ..., v_n\}, v_i = \begin{cases} 1, & v_i \in a \text{ and } v_i \in S, 1 \leq i \leq n; \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Therefore, the feature vector can be expressed as $V_a = \{0, 1, 0, 0, 1, 0, 1, ..., 1\}$, where 1 indicates that the feature is included in the application and 0 indicates that the feature is not included. For sample labels, 1 represents malware, and -1 represents benign.

C. FEATURE SELECTION BASED ON FCBF

Feature selection is the process of selecting a subset of M features from N feature sets while meeting the condition $M \leq N$. The purpose of feature selection is to remove the redundant or irrelevant features from a set of features to reduce the dimensionality.

According to the execution process of the feature selection algorithm, feature selection can be divided into 3 categories: **Filter** methods rely on the general characteristics of the training data to select features with independence of any classifier. **Wrapper** methods use the classifier as a black box and the classifier performance as the objective function to evaluate the variable subset. **Embedded** methods want to reduce the computation time taken up for reclassifying different subsets which is done in wrapper methods. The main approach is to incorporate the feature selection as part of the training process.

FCBF is a fast-correlation filter algorithm proposed by Yu et al. [36], [37] in 2003. It has a wide range of applications in speech recognition [38], network traffic classification [39], and other fields because of its fast calculation. The FCBF algorithm employs symmetrical uncertainty (SU) to measure the correlation between two features. The theoretical basis is that if the SU of feature X and target Y is high, and the SU of other features and target Y is low, then feature X is more important and has a higher weight. When the value of SU between two features is 1, it means that X and Y are completely correlated; in other words, if $X \rightarrow Y$, then $Y \rightarrow X$. When the value of SU is 0, it means that X and Y are completely independent.

The SU uses entropy and conditional entropy to calculate the correlation of features. The entropy of X is:

$$H(X) = -\sum_{i} P(x_i) log_2(P(x_i)) \tag{2}$$

and the entropy of X after observing values of another variable Y is defined as:

$$H(X|Y) = -\sum_{j} P(y_i) \sum_{i} P(x_i|y_i) log_2(P(x_i, y_i)) \tag{3}$$

where $P(x_i)$ is the prior probabilities for all values of X, and $P(x_i|y_i)$ is the posterior probabilities of X given the values of $Y.IG(X,Y)$ represents the information gain:

$$IG(X|Y) = H(X) - H(X|Y) \tag{4}$$

Then, $SU(X,Y)$ between X and Y is:

$$SU(X|Y) = \frac{2IG(X,Y)}{H(X)+H(Y)} \tag{5}$$

An example illustrating the process of the FCBF algorithm is described as the following 7 steps.

Step 1: Calculate symmetric uncertainty $SU_{Xi,Y}$ between feature X_i and target Y.

Step 2: Set threshold δ, if $SU_{Xi,Y} > \delta$, add X_i to feature set S_{list} and arrange the features in descending order according to the $(SU_{Xi,Y})$ values. Suppose that six features $X_1, ..., X_6$ are obtained here, $SU_{X1,Y}$ are the maximum values and $SU_{X6,Y}$ are the minimum values.

Step 3: Select feature X_1 (the first feature in S_{list}) with the maximum value of $SU_{Xi,Y}$ as the main feature X_m.

Step 4: Select features X_2, X_3, X_4, X_5, X_6, whose symmetry uncertainty $(SU_{Xi,Y})$ is less than the main feature $(SU_{Xm,Y})$ in the S_{list}. Calculate the symmetric uncertainty $(SU_{Xi,X\,m})$ between the feature and X_m, and the symmetric uncertainty $(SU_{Xi,Y})$ between the feature and the category Y.

Step 5: If $SU_{Xi,X\,m} > SU_{Xi,Y}$, then this feature is proven to be a redundant feature, and the feature is removed from S_{list}. Here, we assume that features X_2 and X_4 are removed.

Step 6: Add X_1 to the feature subset S_{sub}, choose X_3 as the main feature X_m among the remaining features in S_{list}.

Step 7: If S_{list} is not null, repeat the process of Step 5, suppose that we remove X_6 and add X_5 to the feature subset S_{sub}. If S_{list} is null, S_{sub} is the terminal Subset, the selection is stop.

After the above process, we get the final feature subset S_{sub}. Compared with other algorithms, one of the advantages of the FCBF algorithm is the ability to remove redundant features. For two features X_1 and X_2, with mutual redundancy, suppose that X_1 has a higher correlation with target Y. After calculation, feature X_1 with the higher correlation with category Y is retained, and X_2 with the lower correlation will be removed. At the same time, the more relevant X_1 can be used to filter other features. For a dataset with N features and M instances, the time complexity is $O(MNlogN)$, so it is a fast filtering feature selection algorithm. For the features generated by the FCBF algorithm, we sort it in descending order and then select a certain number of features to form the subset of required features.

D. MALWARE DETECTION AND FAMILY CLASSIFICATION

After using the FCBF algorithm for feature selection, the constructed feature subset will be processed by the classification algorithm, and the maliciousness of the sample will be detected. CatBoost [40] [41] is a machine learning library open-sourced by Yandex in 2017. This algorithm is similar to XGBoost [42] and LightGBM [43] and is an improved algorithm based on the framework of the gradient boosting decision tree (GBDT) algorithm. CatBoost is based on the oblivious trees algorithm with few parameters, supporting categorical variables and high accuracy. Compared with other GBDT-based algorithms, it can process categorical features efficiently and reasonably. In addition, it can also handle gradient bias and prediction shift problems and improve the algorithm's accuracy and generalization ability. The CatBoost algorithm mainly proposes key methods from two aspects, dealing with category features and ordered boosting.

We usually need to process categorical features before building a model. Suppose we have a dataset $D = (X_i, Y_i)$, $i = 1, 2, ..., n$. $X_i = (x_{i,1}, ..., x_{i,m})$ is a vector with m features, including numerical features and categorical features, and $Y_i \in R$ is the label. The most common way to deal with categorical features in GBDT is to replace them with the average values of the tags corresponding to the categorical features. In the decision tree, the label average value will be used as the criterion for node splitting. This method is called greedy target-based statistics, and it is expressed by the formula below, where $[\cdot]$ denotes Iverson brackets, i.e., $[x_{j,k} = x_{i,k}]$ equals 1 if $x_{j,k} = x_{i,k}$ and 0 otherwise. This procedure obviously leads to overfitting.

$$\frac{\sum_{j=1}^{n}[x_{j,k} = x_{i,k}] \cdot Y_j}{\sum_{j=1}^{n}[x_{j,k} = x_{i,k}]} \tag{6}$$

CatBoost uses a more efficient strategy that reduces overfitting and uses the whole dataset for training. Let $\sigma = (\sigma_1, ..., \sigma_n)$ as the permutation, $x_{\sigma_p,k}$ is substituted with (7).

$$\frac{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}]Y_j + a \cdot P}{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}] + a} \tag{7}$$

We also add a prior value P and a parameter $a > 0$, which is the weight of the prior. Adding a prior is a common practice and helps to reduce the noise obtained from low-frequency categories.

Prediction shift is often a problem that plagues modeling. In each iteration of GDBT, the loss function uses the same dataset to obtain the gradient of the current model and then trains to obtain the base classifier. However, it will lead to gradient bias and overfitting. CatBoost replaces the gradient estimation method in traditional algorithms with ordered boosting, reducing the deviation of gradient estimation and improving the model's generalization ability. The principle of ranking improvement is as follows. Suppose that X_i is sorted by a random arrangement σ. To obtain an unbiased gradient estimation, CatBoost will train a separate model $M_i$for each sample X_i, and model M_i is obtained by training using a training set that does not contain sample X_i. Then, model M_i is used to estimate the gradient of the sample, and finally, this gradient training base learner is used to learn the final model.

IV. EXPERIMENTS AND EVALUATIONS

In this section, we discuss the parameter settings and classification results of the presented FAMD framework from 6 different parts of experiments. The parameter settings include N-Gram selection and FCBF algorithm parameter selection. In the classification, we compare the malware detection results with other classifiers, and the key feature distributions are also discussed in this part. The proposed method are compared with other state-of-the-art works, and we also evaluated the family classification results.

A. DATASETS AND EXPERIMENTAL ENVIRONMENT

The experiment uses two datasets: (1) The Drebin dataset, which contains 5,560 malicious samples and 5,666 benign samples. It is widely used as a benchmark dataset and is used to compare FAMD with other similar works. (2) The DS-1 dataset. It is collected by this work and contains a total of 25,737 applications, of which 12,989 are malicious samples and 12,748 are benign samples. The maximum size of a benign sample is 1.16 GB , while its minimum size is 8 KB. The maximum size of a malicious sample is 31.3 MB and its minimum size is 11 KB. We collected all of the benign samples from the third-party markets and used VirusTotal [44] to detect the maliciousness of each benign sample to construct a training dataset as pure as possible.

The experiments use a Dell Power Edge 720 server with Intel Xeon E5-2603 CPU and 64GB RAM. The Python version is 3.7.6, and the main libraries used include Numpy, Pandas, and Skfeature.

B. EVALUATION METRICS

The evaluation metrics are defined as follows. True positive (TP): the number of samples that are actually positive and predicted positive. False Positive (FP): the number of samples that are actually negative but predicted positive. False Negative (FN): the number of samples that are actually positive but predicted negative. True Negative (TN): the number of samples that are actually negative and predicted negative.

1) TPR

The percentage of samples correctly identified as positive out of the total positive samples.

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

2) FPR

The percentage of samples wrongly identified as positive out of the total negatives samples.

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

3) Accuracy

The percentage of correctly classified samples out of the total number of samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

4) Precision

The percentage of correctly predicted positive samples out of the total predicted positive samples.

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

5) F1-Socre

The combination of precision and recall metrics that serves as a comprise. The best F1-score equals 1, while the worst score is 0.

$$F1 - Socre = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{12}$$

6) ROC curve

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various values and threshold settings. It illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

7) AUC

The area under the ROC curve is AUC, and its value can be used to intuitively evaluate the quality of the classifier. The closer the AUC is to 1.0, the better the detection method will be. When it is equal to 0.5, it has no application value.

C. EXPERIMENTAL RESULTS

1) N-Gram Setting

For Dalvik opcodes based on the N-Gram, the value of N affects two aspects: classification accuracy and feature numbers. We use the DS-1 dataset to set the segmentation length of the Dalvik opcode. When the length is set to N=[2,3,4,5], the corresponding length of the N-Gram opcode sequence is extracted. The extracted features are input into the CatBoost classifier, and 10-fold cross-validation is selected to find the most appropriate length of N.

Due to the diverse designs of Android applications, especially malware applications that deliberately evade certain features. From some APK files cannot be extracted a single kind of features such as permissions. This results in many samples being ignored in classification. We compare the number of samples whose extraction failed when extracting features individually and when extracting combined features on the DS-1 dataset, as shown in Table 2.

TABLE 2. The number of sample failures while extracting different features

Feature	Malware	Benign
Permission	10	2689
2-Gram	93	0
3-Gram	93	7
4-Gram	93	7
5-Gram	93	8
Permission with 2-Gram	0	0
Permission with 3-Gram	1	6
Permission with 4-Gram	1	6
Permission with 5-Gram	1	7

It can be seen from Table 2 that when extracting the permission features combined with N-Gram opcodes, the features can be extracted from most of the samples, which is better than extracting a single kind of feature. When extracting permission features, benign samples are more difficult to extract. Since we extracted and discussed official Android permissions in this work, it may related to some samples that employ more customer permissions. The feature extraction of N-Gram opcodes is the opposite. There are more cases of relatively failed extraction in malware, such as the sample (SHA-1: 8d2795c2e790c54b401fd52eb56279f6af0a07fb), which is small in size and performs malicious behaviors through calling permissions.

We compare the accuracy and the number of constructed features with different N-Gram length of Dalvik opcodes in Table 3. It can be seen that as the value of N gradually increases, the classification accuracy is better but the growth trend is getting smaller. However, as N increases, the number of features will also increase obviously, leading to increased computational consumption.

TABLE 3. Accuracy comparison of different N values

Feature	Accuracy	Features Numbers
2-Gram	69.66%	49
3-Gram	88.44%	343
4-Gram	92.85%	2356
5-Gram	93.65%	16052

We combine the extracted permission features with the Dalvik opcode features and apply different values for N, and the results are shown in Table 4.

TABLE 4. Accuracy comparison of different feature combinations

Feature	Accuracy	Features Numbers
Permission	88.09%	111
Permission with 2-Gram	88.90%	160
Permission with 3-Gram	93.60%	464
Permission with 4-Gram	95.84%	2467
Permission with 5-Gram	96.21%	16163

The combination of two kinds of features achieves better accuracy than any single feature kind. The best accuracy is 96.21% when the features are "Permission with 5-Gram", and the second-best result is 95.84% with "Permission with 4-Gram".

According to the results in Table 3 and Table 4, as well as considering about the accuracy and feature dimensionality, we set the value of N to 4, and employ "Permission with 4-Gram" as features in the following experiments.

2) FCBF Algorithm Parameter Setting

Since the original feature set has high dimensionality, we use FCBF algorithm to perform feature selection to construct appropriate feature subset. We set the range of threshold δ

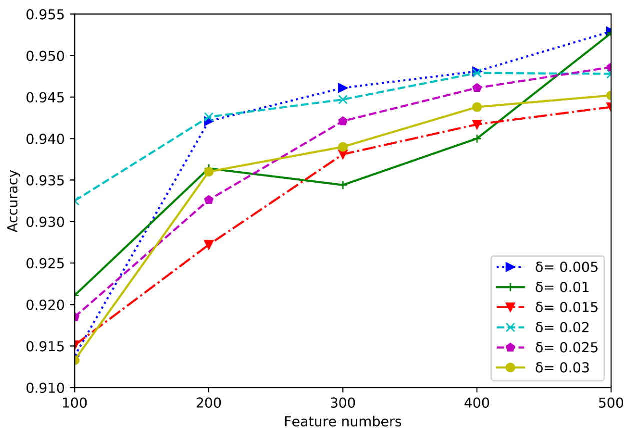

FIGURE 2. The accuracy of the FCBF algorithm using different parameters.

to [0.005, 0.03], with the interval of 0.005, and the feature

TABLE 5. Experimental results of different classifiers

Feature Numbers	Algorithm	Precision	Accuracy	F1-score	Recall
100	KNN	88.40%	89.29%	89.55%	90.74%
	RF	89.45%	90.76%	91.03%	92.66%
	XGBoost	87.61%	88.80%	89.12%	90.69%
	LGBM	90.65%	90.78%	90.90%	91.16%
	CatBoost	**91.18%**	**91.38%**	**91.51%**	**91.84%**
200	KNN	90.46%	91.48%	91.70%	92.97%
	RF	92.18%	93.40%	93.58%	95.02%
	XGBoost	92.98%	93.30%	93.41%	93.84%
	LGBM	93.86%	94.13%	94.22%	94.58%
	CatBoost	**93.65%**	**94.21%**	**94.32%**	**95.00%**
300	KNN	90.38%	91.48%	91.71%	93.07%
	RF	92.92%	93.94%	94.08%	95.28%
	XGBoost	94.12%	94.43%	94.52%	94.92%
	LGBM	94.03%	94.27%	94.35%	94.68%
	CatBoost	**94.39%**	**94.61%**	**94.69%**	**95.00%**
400	KNN	90.97%	92.03%	92.23%	93.53%
	RF	93.28%	94.22%	94.36%	95.46%
	XGBoost	94.25%	94.55%	94.63%	95.02%
	LGBM	94.51%	94.94%	95.02%	95.53%
	CatBoost	**94.55%**	**94.81%**	**94.89%**	**95.23%**
500	KNN	91.64%	92.57%	92.75%	93.89%
	RF	93.72%	94.57%	94.69%	95.69%
	XGBoost	94.78%	95.04%	95.12%	95.46%
	LGBM	95.01%	95.12%	95.18%	95.35%
	CatBoost	**94.96%**	**95.29%**	**95.36%**	**95.77%**

numbers are set in the range of 100 to 500. The result is shown in Fig. 2.

From Fig. 2, as the number of features increases, the detection results are getting better. The best accuracy is achieved when threshold δ is set to 0.005 and the number of features is set to 500, and that will be the chosen parameters of FCBF in this work.

3) Compare with Other Classifiers

The DS-1 dataset is used as experimental data, with 70% of the data is used as the training set and the rest is the test set. For all base classifiers, we use the grid search method to find each classifier's best parameters. The specific experimental results are shown in Table 5. The value of the AUC only counts the comparison when the number of features is 500, as shown in Fig. 3. It can be seen from Table 5 and Fig. 3 that as the number of features increases, the various experimental indicators of the Catboost classifier are better than those other classifiers in most cases. When the number of features reaches 500, CatBoost that we used can achieve 95.29% accuracy.

4) Analysis of the key features

We count the importance of the top 10 features ranked by FCBF and their distribution in malware and benign applications. The results are shown in Table 6. The top 10 features include both permissions and opcodes, which proves that these two kinds of features indeed have the classification ability.

In addition, there are significant differences in the feature distribution of malicious software and benign applications. For example, the permission "RECEIVE_SMS" takes 38.51% of malware and only 4.24% of benign applications, and the permission "READ_PHONE_STATE" takes 92.82% of malware and 54.13% in benign applications. In another words, compared with benign applications, malware take more attempts to obtain the user's SMS information and device identification.

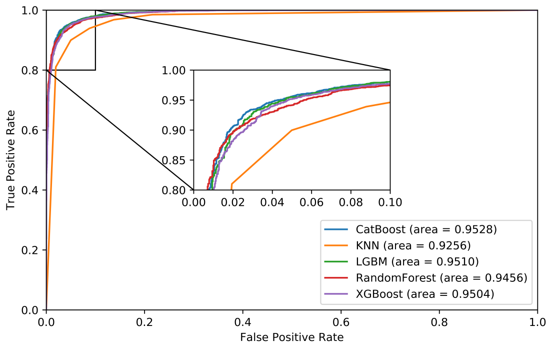

FIGURE 3. ROC of different classifiers.

5) Compare with Other Experiments

We compare the FAMD with other state-of-the-art works with Drebin dataset. Since the evaluation metrics in each paper are different, we have collected as comprehensive information as possible, as shown in Table 7.

It can be seen from Table 7 that FAMD can outperform most of other works in terms of accuracy. Moreover, FAMD only needs an average of 0.28s to analyze an appli-

TABLE 6. Distribution of the top-10 features in malware and benign application

Feature Name	Malware	Percentage of malware	Benign	Percentage of benign
Permission_RECEIVE_SMS	5002	38.51%	540	4.24%
Ngram_RMTG	3935	30.29%	1585	12.43%
Permission_READ_PHONE_STATE	12057	92.82%	6900	54.13%
Permission_WRITE_EXTERNAL_STORAGE	10864	83.64%	9452	74.14%
Permission_INSTALL_SHORTCUT	6718	51.72%	3032	23.78%
Ngram_GPMM	2611	20.10%	1276	10%
Ngram_MGVR	3215	24.75%	4614	36.20%
Ngram_VRIP	2231	17.18%	4138	32.46%
Ngram_PTGV	3185	24.52%	2165	16.98%
Ngram_GMRM	3747	28.85%	2043	16.03%

TABLE 7. Comparison with related work

Method	Accuracy	Precision	F1-score	TPR	FPR	AUC	Times
Drebin [29]	93.90%	-	-	94%	1%	-	0.75s
Opcode ngram [45]	96.88%	95.70%	96.30%	-	-	-	-
ICC-detector [30]	97.40%	-	-	93.10%	0.67%	-	0.79s
RoughDroid [46]	-	-	-	95.60%	1%	95.63%	-
Method-Level Behavioral Semantic Analysis [6]	96.00%	97.00%	96.00%	-	-	-	2.9S
FAMD	**97.40%**	**98.00%**	**97.38%**	**96.77%**	**1.97%**	**97.40%**	**0.28s**

cation. Hence, FAMD achieves the purpose of establishing a lightweight and efficient detection framework.

6) Malware Family Classification

In addition to malware detection, malware family classification is also concerned in this framework. We take the top 20 malware families in Drebin dataset and conducts the family classification with the presented methods. The precision, F1-score, and recall indexes are in Fig. 4 and the confusion matrix is in Fig. 5 to provide a graphical overview. Most of the samples are correctly classified into their respective families.

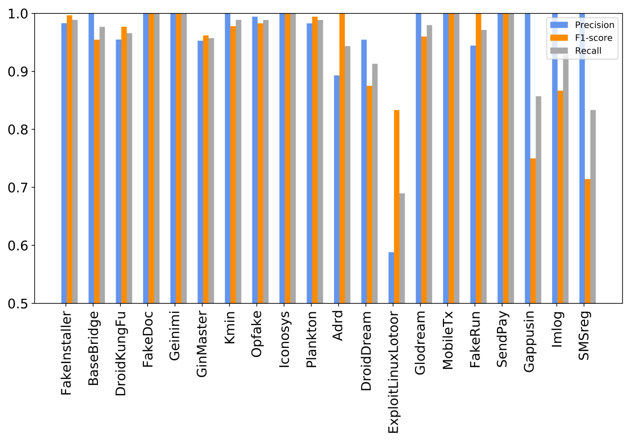

FIGURE 4. Classification results of top 20 malware families in Drebin dataset.

The overall accuracy of malware family classification is 97.38%, which shows the effectiveness and feasibility of the FAMD in family processing after malware detection. However, the ExploitLinuxLotoor family has a lower precision.

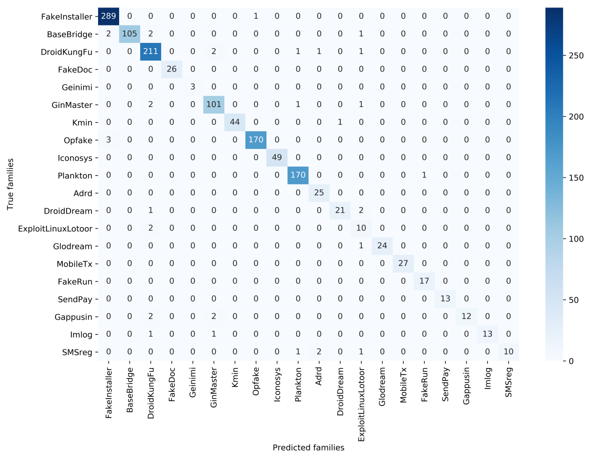

FIGURE 5. Confusion matrix of the top 20 Drebin malware families.

And for the Geinimi family, we only successfully extracted the features of 17 samples, in other words, from most of samples of this family cannot be extracted permissions and opcode features. This shows that our framework performs unsatisfied to detect malware that carries out certain activities, which is what we need to improve in our future work.

V. CONCLUSION

The number of applications that can be classified as malware continues to increase, new types of malware and camouflage techniques are constantly updating, effectively detecting malware in a relatively short time is of considerable significance to the third-party application markets and users. How to improve the detection accuracy and reduce the detection time are still the problems to be solved.

We present a fast Android malware detection framework, FAMD, which combines permission features and Dalvik opcode features from different operation levels to construct feature vectors. To reduce the feature dimensionality and time complexity of the method, the FCBF algorithm is employed for feature selection. As a classifier proposed in recent years, CatBoost is employed in this work to conduct malware detection and family classification.

In the experiments, we segment the opcodes with 4-Gram and vectorize the features combined with permissions. With the CatBoost as the classifier, the result achieves an accuracy of 97.40% in malware detection, and 97.38% in family classification. Compared with other state-of-the-art works, FAMD performs better comprehensively in accuracy and time consumption. It can be seen in the experiments that there is a clear difference in the distribution of certain key features in malicious applications and benign applications.

Since CatBoost is a supervised learning framework, this work is inadequate in detecting new emerging malicious applications, which we aim to improve in further work.

REFERENCES

[1] (2020) Smartphone Market Share. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os

[2] (2020) Mobile malware evolution 2019. [Online]. Available: https://securelist.com/mobile-malware-evolution-2019/96280

[3] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 235–245.

[4] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[5] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2018.

[6] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, pp. 69 246–69 256, 2019.

[7] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–34, 2019.

[8] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[9] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the face of android ransomware: Characterization and real-time detection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1286–1300, 2017.

[10] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: Effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2018.

[11] S. Y. Yerima, M. K. Alzaylaee, and S. Sezer, "Machine learning-based dynamic analysis of android apps with improved code coverage," *EURASIP Journal on Information Security*, vol. 2019, no. 1, p. 4, 2019.

[12] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 371–372.

[13] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors." in *Ndss*, 2015.

[14] (2020) Android Permission. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission

[15] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Álvarez Marañón, "Mama: manifest analysis for malware detection in android," *Cybernetics and Systems*, vol. 44, no. 6-7, pp. 469–488, 2013.

[16] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014.

[17] K. A. Talha, D. I. Alper, and C. Aydin, "Apk auditor: Permission-based android malware detection system," *Digital Investigation*, vol. 13, pp. 1–14, 2015.

[18] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[19] Q. Jerome, K. Allix, R. State, and T. Engel, "Using opcode-sequences to detect malicious android applications," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 914–919.

[20] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé *et al.*, "Deep android malware detection," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 301–308.

[21] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik opcode graph based android malware variants detection using global topology features," *IEEE Access*, vol. 6, pp. 51 964–51 974, 2018.

[22] A. Pektaş and T. Acarman, "Learning to detect android malware via opcode sequences," *Neurocomputing*, vol. 396, pp. 599–608, 2020.

[23] A. Egitmen, I. Bulut, R. Aygun, A. B. Gunduz, O. Seyrekbasan, and A. G. Yavuz, "Combat mobile evasive malware via skip-gram-based malware detection," *Security and Communication Networks*, vol. 2020, 2020.

[24] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *International conference on security and privacy in communication systems*. Springer, 2013, pp. 86–103.

[25] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2014.

[26] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1507–1515.

[27] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE access*, vol. 7, pp. 21 235–21 245, 2019.

[28] J. Zhang, Z. Qin, H. Yin, L. Ou, and K. Zhang, "A feature-hybrid malware variants detection using cnn based opcode embedding and bpnn based api embedding," *Computers & Security*, vol. 84, pp. 376–392, 2019.

[29] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[30] K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.

[31] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and api calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020.

[32] Permissions overview. [Online]. Available: https://developer.android.google.cn/guide/topics/permissions/overview

[33] W. B. Cavnar, J. M. Trenkle *et al.*, "N-gram-based text categorization," in *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, vol. 161175. Citeseer, 1994.

[34] Dalvik bytecode. [Online]. Available: https://source.android.com/devices/tech/dalvik/dalvik-bytecode

[35] Androguard. [Online]. Available: https://github.com/androguard/androguard

[36] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 856–863.

[37] B. Senliol, G. Gulgezen, L. Yu, and Z. Cataltepe, "Fast correlation based filter (fcbf) with a different search strategy," in *2008 23rd international symposium on computer and information sciences*. IEEE, 2008, pp. 1–4.

[38] D. Gharavian, M. Sheikhan, A. Nazerieh, and S. Garoucy, "Speech emotion recognition using fcbf feature selection method and ga-optimized

fuzzy artmap neural network," *Neural Computing and Applications*, vol. 21, no. 8, pp. 2115–2126, 2012.

[39] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2005, pp. 50–60.

[40] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," *arXiv preprint arXiv:1810.11363*, 2018.

[41] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in neural information processing systems*, 2018, pp. 6638–6648.

[42] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[43] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in neural information processing systems*, 2017, pp. 3146–3154.

[44] Virustotal. [Online]. Available: https://www.virustotal.com/gui/home/upload

[45] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family android malware," in *2015 10th International Conference on Availability, Reliability and Security*. IEEE, 2015, pp. 333–340.

[46] K. Riad and L. Ke, "Roughdroid: operative scheme for functional android malware detection," *Security and Communication Networks*, vol. 2018, 2018.

HONGPENG BAI received B.S. degree from Liaoning Shihua University in 2016. He is currently a postgraduate student in computer technology at the Changchun University of Science and Technology, China. His research interests include machine learning and malware detection.

NANNAN XIE received B.S. degree in software engineering from Jilin University in 2010, and Ph.D. degrees in computer system architecture from Jilin University in 2015, respectively. She worked as a postdoctoral researcher in Beijing Jiaotong University, China, from 2015 to 2017. She is currently a lecturer and a master's supervisor at Changchun University of Science and Technology. She has published about 20 scientific papers in various journals and international conferences. Her main research interests include network intrusion detection and mobile security.

XIAOQIANG DI received B.S. degree in computer science and technology from Changchun University of Science and Technology in 2002, and M.S. and Ph.D. degrees in communication and information systems from Changchun University of Science and Technology in 2007 and 2014, respectively. He was a visiting scholar at Norwegian University of Science and Technology, Norway, from Aug. 2012 to Aug. 2013. He is currently an professor and supervisor of Ph.D. in Changchun University of Science and Technology. His major research interests include network information security and integrated network.

QING YE received B.S. degree in computer science and technology from Northeast Normal University in 1992, and M.S. degree from Changchun University of Science and Technology in 1995. She is currently an professor at Changchun University of Science and Technology. Her major research interests include database and data mining, software engineering and information systems, and machine learning.