IEICE TRANS. INF. & SYST., VOL.E105–D, NO.3 MARCH 2022

PAPER

Android Malware Detection Based on Functional Classification

Wenhao FAN[†a)], **Dong LIU**[†b)], ***Nonmembers***, **Fan WU**[†], ***Member***, **Bihua TANG**[†], ***and*** **Yuan'an LIU**[†], ***Nonmembers***

SUMMARY Android operating system occupies a high share in the mobile terminal market. It promotes the rapid development of Android applications (apps). However, the emergence of Android malware greatly endangers the security of Android smartphone users. Existing research works have proposed a lot of methods for Android malware detection, but they did not make the utilization of apps' functional category information so that the strong similarity between benign apps in the same functional category is ignored. In this paper, we propose an Android malware detection scheme based on the functional classification. The benign apps in the same functional category are more similar to each other, so we can use less features to detect malware and improve the detection accuracy in the same functional category. The aim of our scheme is to provide an automatic application functional classification method with high accuracy. We design an Android application functional classification method inspired by the hyperlink induced topic search (HITS) algorithm. Using the results of automatic classification, we further design a malware detection method based on app similarity in the same functional category. We use benign apps from the Google Play Store and use malware apps from the Drebin malware set to evaluate our scheme. The experimental results show that our method can effectively improve the accuracy of malware detection.

key words: *Android, malware detection, functional classification, mobile security, HITS algorithm*

1. Introduction

Android operating system occupies a high market share. Android apps develop fast, but research shows that there are a large amount of malware endangering the safety of Android users [1]. At present, there is still space for Android malware detection technology to further improve the accuracy, and the existing research works do not make the best use of the app functional category information. Through the research of Android app functional classification, it is found that benign apps of the same category have strong similarity in implementation of their functionalities. Besides, permissions, APIs and the relationship between APIs of similar apps also have significant similarity. Based on this similarity, we use static features to classify the apps according to their functionalities. Malware detection methods take advantage of the difference between malware and benign apps. The difference in the same functional category is more obvious so the accuracy of malware detection will be higher in this case. What's more, we can use less features to detect malware in each functional category. When we get the functional category label of the app, we use a new method to measure the similarity between apps. This method calculates the similarity between apps to detect malware in the same functional category. We innovatively propose to apply HITS algorithm to the process of API feature selection. HITS algorithm is an algorithm applied in network retrieval. It can get the relevance ranking of web pages related to users' search keywords, so as to return the most desired information. The ranking mechanism of the algorithm makes use of the link relationship between web pages, that is, one web page contains hyperlinks to other web pages. Through the research, we found that the API in Android apps has a similar relationship. In the app functional classification, we increase the weight of the top APIs selected by HITS algorithm, which introduces the factor of the relationship between APIs and functionalities, and further improves the accuracy of app functional classification. On the basis of app functional classification, we propose a method to calculate app similarity, which can calculate the similarity between two apps by using the API feature and the relationship between apps and APIs. Then, we borrow and improve k-nearest neighbor (KNN) algorithm, using the benign or malicious labels of known samples to identify unknown samples. Due to the huge number of existing samples in the same category, it spends too much time calculating the similarity between the app to be tested and the existing samples. We combine the samples with high similarity. For some samples with high similarity, their benign and malicious labels are not same. We adopt the strategy of increasing the similarity threshold and discarding some samples to solve this problem.

HITS algorithm is based on the link between web pages, ranking the relevance between web pages and search keywords. The core idea is to first obtain the initial set of related web pages according to keywords, and then expand the set by the link between web pages. HITS algorithm sets an authority attribute and hub attribute for each web page in the set, then iterates the web pages by the link and the iteration formula. Finally, it filters out the top web pages according to the authority value. From the perspective of API, the API in Android apps is similar to web pages. Web pages are connected by hyperlinks, which is similar to the calling relationship between APIs. The core idea of HITS is that a good hub web page will point to many good authority web

Manuscript received June 12, 2021.
Manuscript revised October 11, 2021.
Manuscript publicized December 1, 2021.

[†]The authors are with the School of Electronic Engineering, and Beijing Key Laboratory of Work Safety Intelligent Monitoring, Beijing University of Posts and Telecommunications, Beijing, China.
a) E-mail: whfan@bupt.edu.cn
b) E-mail: liud@bupt.edu.cn

DOI: 10.1587/transinf.2021EDP7133

pages, and a good authority web page will be pointed to by many good hub web pages. We find that there is a similar logic in the calling relationship between the APIs. We can apply the HITS algorithm to the API filtering. HITS algorithm obtains the authority values of different APIs, and we set the corresponding weights for the APIs according to the authority values. Similar to web pages, the authority of API features represents the relevance between API and functionalities. The higher the authority, the more important the API is in the implement of functionality. We set high weight for them, so that they can play a greater role in classification, so as to improve the accuracy. In other words, we set weights for them according to the relationship between API and functionality. Through our app classification method, we have analyzed 12791 apps and extracted 17038 features from 16 app categories of Google Play Store, and finally achieved 86.6% classification accuracy.

KNN algorithm is a simple classification algorithm, which determines the category of the samples to be tested by counting the labels of the K nearest samples, and takes the category of most of the K samples as the result. Although the algorithm theory is simple, it often has good performance in practice. Inspired by KNN algorithm, we propose a new distance function, that is, the similarity between apps. The API and the relationship between the APIs are used to measure the app similarity. Similar to KNN algorithm, our scheme determines the category of the samples according to the label of the adjacent samples. The disadvantage of KNN algorithm is that it needs a lot of computation to find the nearest K sample points. For this problem, we use the similar sample merging strategy.

We make the following contributions in this paper:

Firstly, aiming at the problem of application functional classification, this paper proposes an Android application functional classification method based on HITS algorithm. HITS algorithm is used to filter the API features of applications in this method. It sets different weights for API features, and the API which plays a key role in the implementation of application's functionality has higher weight. In this way, this method improves the accuracy of functional classification. This method uses the filtered weighted API features, combined with a variety of static features, and uses the ensemble learning model to classify the applications according to their functions. The app classification model based on HITS algorithm is tested, which verifies the effectiveness of HITS algorithm in Android app classification.

Secondly, this paper proposes an Android malware detection scheme based on functional classification. The scheme include a new method to calculate application similarity. The types of APIs and the relationship between them are taken into account in this method. It builds relationship matrix for APIs according to apps' code logic and data flows and uses matrix operation to calculate the similarity between the two apps. The scheme classifies the applications and detects malware within the same functional category. This scheme uses KNN algorithm for reference, integrates the concept of similarity into the process of malware detection, and improves KNN algorithm for Android malware detection. The proposed scheme is compared with the existing Android malware detection methods to prove the effectiveness.

The reminder of this paper is organized as follows. Section 2 details the background and related works. Section 3 introduces the whole scheme. Section 4 describes the empirical evaluation. Section 5 presents our conclusions and future work.

2. Related Works

The existing Android malware detection technology is mainly divided into two categories, one is the innovation in application feature, the other is the innovation in machine learning algorithm.

2.1 The Innovative Research Work on App Feature

Li *et al.* [2] proposed to use permission for malware detection, and designed a three-tier screening model for permission. Abro *et al.* [3] proposed using permission and intent information to detect malware. Liang *et al.* [4] proposed to construct permission combination for malware detection. The above three methods ignore the differences in the use of permissions between apps of different functional categories. Peiravian *et al.* [5] proposed using API and permission information to detect malwares. Wu *et al.* [6] proposed using API, permission and intent information for malware detection, and classified the intention of malware. These two methods consider the API information, but the analysis of the relationship between APIs is not enough. Enck *et al.* [7] proposed using dynamic analysis method to detect malwares and track sensitive information flow of apps in real time. Xiao *et al.* [8] proposed using semantic model to process dynamic API call information. Ni *et al.* [9] proposed to extract the dynamic API and permissions of apps to detect malware. The average detection time of each app is 18 minutes. The above three methods use the dynamic features of the app to judge whether the app is a malware, but the dynamic analysis needs to run the app, which takes a long time, and does not cover the logic of the app code completely, so it is easy to miss useful features.

2.2 The Innovative Research Work on Machine Learning Algorithm

Arp *et al.* [1] first proposed to extract various static features and use SVM algorithm to detect malware. Aafer *et al.* [10] proposed that KNN algorithm can further improve the accuracy of malware detection. Zhu *et al.* [11] proposed to use rotation forest algorithm for malware detection. Yerima *et al.* [12] proposed a new classifier fusion method based on multi-level structure, which can effectively combine with machine learning algorithm to improve the accuracy. Gaikwad *et al.* [13] proposed a classifier fusion method to detect malicious intrusion. Wang *et al.* [14] proposed a

deep learning method to identify malware. Although the above methods are innovated from the perspective of machine learning, there are some shortcomings, such as insufficient use of applied features, insufficient depth of feature association mining, and weak interpretability.

Compared with the existing research works on Android malware detection, our proposed malware detection scheme based on functional classification fully considers the differences of different categories. At the same time, we propose an app similarity calculation method based on APIs and API relationship. Then we borrow and improve KNN algorithm to realize malware detection.

3. Solution

Our scheme includes Android app processing, feature filtering, application functional classification and malware detection. The overview of our scheme is shown in Fig. 1.

1. In the app processing stage, we use Apktool [15] to decompile the program package (APK file) in the dataset, analyze the AndroidManifest.xml and smali code file, and obtain the static features of the app. We use term frequency-inverse document frequency (TF-IDF) algorithm to preliminarily filter the features of each category, and use the API call relationship to expand the API set.

2. In the feature screening stage, HITS algorithm is used to screen API features. We assign authority and hub attributes to the APIs in the set, and use the calling relationship between APIs to update the attribute values iteratively. Finally, we select the API with the highest authority value as the API set to construct the API feature vector.

3. In the app functional classification stage, the API features filtered by HITS algorithm are combined with other static features to construct feature vectors for machine learning, and the corresponding weights are set for API features according to the result of HITS algorithm. The ensemble learning algorithm is used to train the automatic classification model. We adjust the parameters of the training model, and select the optimal parameters for experimental analysis.

4. In the malware detection stage, we calculate the similarity between each two apps in every functional category, and merge the samples with high similarity. Finally, we borrow and improve KNN algorithm to judge the benign or malware category of the app to be tested.

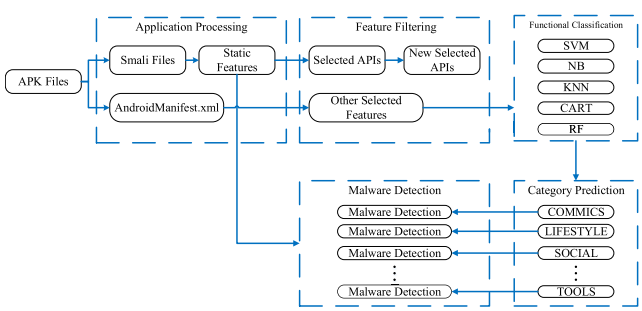


Fig. 1 The overview of our scheme.

3.1 App Processing

In the first stage, we need to collect the permissions, intents, hardware features and API related information from APK files. API is the key element of the apps' functionalities, and it accounts for the largest proportion in feature set. After extracting the APIs from smali folder, we will be able to pick up the implementation logic of the app's functionality. Apps in the same category have strong similarity in implementation. What's more, the permissions declared by an app are closely associated to the app's functionality. When the app is about to apply for hardware or user information resources on smartphones, it must declare relevant permissions in advance. Therefore, the quantity and type of the declared permissions in app are similar in the same category. Apktool is used to decompile APK files for getting these static features from Android Manifest.xml and smali folder.

In order to improve the efficiency of machine learning training and app detection, TF-IDF algorithm is employed to preliminarily screen the static features used by each category. TF-IDF is an algorithm for text feature filtering, which is used to filter the most suitable topic words from the article. It takes into account the frequency and the scope of text occurrence. TF-IDF algorithm can also be applied to filtering the app features. One static feature in an app can be regarded as a word in an article. With this algorithm, we can greatly reduce the quantity of the features.

After obtaining the API initially screened and the call relationship between APIs, we use the call relationship to expand the API set. The extension principle is to add all APIs which have call relationship with the APIs filtered by TF-IDF algorithm to the set.

The number of features filtered by TF-IDF algorithm is not enough. If we only increase the number of features filtered by TF-IDF algorithm, we will filter out some APIs that are only used by a few apps in this category, which will reduce the generalization ability of API features. Therefore, we take advantage of the call relationships to expand the initial API set, and then filter the set again through the subsequent algorithm to select the most relevant APIs for the app classification.

3.2 App Feature Filtering

The four main components of Android app are Activity, Service, Broadcast Receiver and Content Provider. Among them, activity is the foundation of Android app, and the logic of interaction between app and user is defined in activity. Activity has four states in its life cycle, which are Running, Pause, Stop and Destroy. Seven callback methods are defined in the activity, covering every step of the life cycle. The life cycle methods of activity will call various third-party libraries and APIs to realize the app's functionalities [16]. Android app development is based on Java programming language. In Java, method is made up of many lines of code, which are arranged in order to realize special

functionality. The code is the content of API. And the APIs have a great difference in implementation according to their different functionalities. Some APIs, such as the life cycle methods in activity, call a large number of methods from other classes, while others contain the concrete implementation of functionality.

HITS algorithm allocates two attributes to web pages, authority value and hub value. The authority value refers to the importance of the content in the web page, and the hub attribute represents the importance of the links in web pages. It is indicated that a web page with high authority value should have rich content and can be linked by many other pages. A web page with high hub value should be capable of hopping to many good and meaty pages, which corresponds to the core hub in the web pages.

We put the web page and API together for analogy. We discover that API has the same nature with web page in the relationships of API calls. The link between web pages resembles the call relationship between APIs. Besides, different types of API have differences on authority value and hub value. The authority value of APIs containing complex implementation of logic is higher than common APIs. Usually, the APIs with high hub value control the procedure of activity, such as life cycle methods. They do not contain complex logic, and most rely on calling other APIs to realize the functionality. On this basis, we consider applying HITS algorithm to select key APIs in app classification.

Therefore, according to the ideas of HITS algorithm, we set the authority and hub attribute for each API in the extension set. These two attributes are initialized to 1. Then we update them iteratively according to the relationships between API calls.

Nevertheless, there is a difference between the quantity of web page links and API calls. The link relationships in web pages are dense, and all links can be obtained. However, the relationships of API calls are relatively sparse. The source code of some Java based APIs cannot be obtained by decompiling, so the relationships of API calls collected are not comprehensive. In order to make up for this kind of call relationships, we expand the definition of the call relationship. We think there is a connection between the APIs invoked in the same method. For instance, if method A consists of method B and C, then we believe that there are call relationships between method B and C. This will increase the amount of the call relationships, and the frequently called API will get higher weight.

The premise of HITS algorithm is that the hub value and authority value are convergent, that is, the convergence of the algorithm. For the expanded API set, a matrix M is used to represent the relationships between APIs: $m_{ij} = 1$ indicates that the ith API calls the jth API, otherwise, it is 0. Vector H is designed to represent the hub values of all APIs, in which the ith element represents the hub value of the ith API. Vector A represents the authority value of all APIs, and the ith element represents the authority value of the ith API. The hub and authority values for all APIs are initialized to 1. Based on the above settings, we get the following results:

$$A_k = M^T H_{k-1} \tag{1}$$

$$H_k = MA_k \tag{2}$$

All the components of Z are 1.

$$A_k = (M^T M)^{k-1} M^T Z \tag{3}$$

$$H_k = (MM^T)^k Z \tag{4}$$

$M^T M$ and MM^T are symmetric matrices with n real eigenvalues. In $H_k = (MM^T)^k Z$, the principal eigenvector of Z and MM^T is not orthogonal, so the vector H will eventually converge to the principal eigenvector of MM^T. In order to ensure that vector H is a unit vector, we will standardize it after each iteration.

In $A_k = (M^T M)^{k-1} M^T Z$, the principal eigenvector of $M^T Z$ and $M^T M$ are not orthogonal, and vector A will eventually converge to the principal eigenvector of $M^T M$. Similarly, in order to ensure that vector H is a unit vector, we will standardize it after each iteration. So far, we have proved the convergence of HITS algorithm in APIs.

After the iteration, we acquire the APIs with high authority values in each category. These APIs are filtered out to represent the app and are closely related to the functionality. Then we combine the APIs filtered from each category for the construction of feature vectors. The process of HITS is shown in Algorithm 1.

3.3 App Functional Classification

After two rounds of screening, the quantity of features will be greatly reduced. Firstly, TF-IDF algorithm filters the static features collected from AndroidManifest.xml and generates the initial API set. Secondly, HITS algorithm filters the API features. These two kinds of features constitute

Algorithm 1 Process of screening APIs by HITS algorithm

Input: The set of call relationships; The expanded set of APIs; Max iterations,*m*; Iteration parameters,*min_delta*; The number of selected apis in each category,*k*;
Output: The set of selected APIs, S_n;

```
for each category c in categories do
    Initialize a vector H with all elements = 1;
    Initialize a vector A with all elements = 1;
    for t = 0 → m do
        for i = 0 → range(size(A)) do
            A[i]=0;
            for j = 0 → range(size(H)) do
                if API i calls API j then
                    A[i]+=H[j]
        Standardize vectorA;
        for i = 0 → range(size(H)) do
            H[i]=0;
            for j = 0 → range(size(A)) do
                if API j calls API i then
                    H[i]+=A[j]
        Standardize vectorH;
        if Change of results < min_delta then
            break;
    Add the top_k APIs from category c to S_n;
return S_n and their authority values;
```

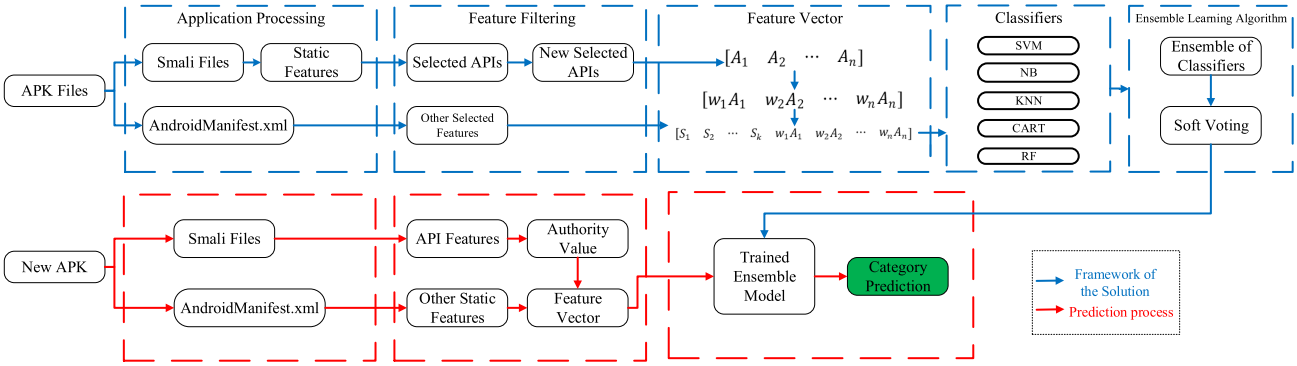


Fig. 2 The overview of the functional classification.

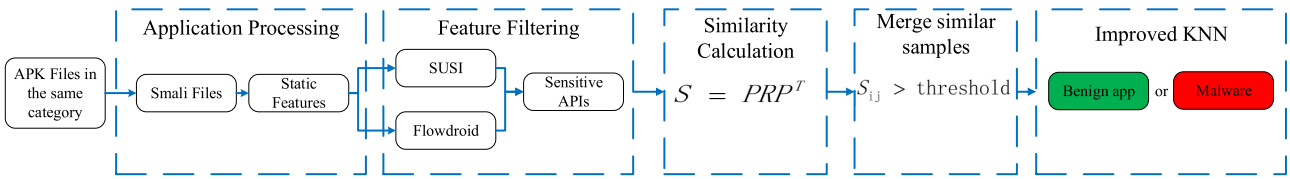


Fig. 3 The overview of the malware detection.

the final feature set. For API features, we set corresponding weights for them according to their authority values. In this way, the API which is closely related to the category functionality will get a higher weight, thus playing a greater role in the app classification. We employ the final feature set to match each app in the dataset. Then our method will generate a 17038-dimensional vector for each app. The first half of the vector is matched with the static feature selected from AndroidManifest.xml, and the second half is matched with API features.

When generating a feature vector for a new app, we first scan its AndroidManifest.xml. If it contains a feature which exists in the final feature set, the corresponding position of the feature in the vector will be set to 1, otherwise it will be set to 0. After that, we will traverse its smali folder. According to the fully qualified class name of the Activity registered in the AndroidManifest.xml, we scan the corresponding smali code file, so as to determine whether the current app contains the valid API feature. When valid API feature is found, the corresponding position in the feature vector will be set to its weights according to API's authority value, otherwise it will be set to 0. With this method, we generate the feature vectors for 12791 apps in the dataset. Then these vectors will be used to train the machine learning module and test the effectiveness of our method.

Compared with single machine learning algorithm, ensemble learning algorithm has higher fault tolerance [17]. We employ SVM algorithm, random forest (RF) algorithm, NB algorithm, KNN algorithm, classification and regression tree (CART) algorithm to build an ensemble learning model. We choose the Bagging algorithm. The characteristic of Bagging method is random sampling. It trains new models through repeated sampling, and finally uses soft voting to generate the final result. The experimental result shows that compared with weak classifiers, ensemble learning algorithm gets better results [17]. The structure of the model is shown in Fig. 2.

3.4 Malware Detection

We use the functional classification method to classify the malware set, and combine the classified malware with the labeled benign apps obtained from the Google Play Store to form a malware detection dataset. Then we calculate the similarity between the apps in the same functional category. The overview of this model is shown in Fig. 3.

In order to calculate the similarity from the perspective of malware detection, we extract the APIs related to the sensitive data flow in the app. The data flow inside the app can explain whether the app has malicious behavior. Firstly, we use SUSI [18] to extract the sensitive APIs, which are mostly related to sensitive information, sensitive permission and sensitive behavior of the app. Then we use FlowDroid [19] to extract the data flow of the app. FlowDroid can effectively extract the data flow that exist in the app. Using sensitive API and data flow, we can obtain the data flow related to sensitive API, and retain these data flow as sensitive data flow. Then we form a new API set, which includes all APIs in these data flow. These APIs include sensitive APIs and APIs related to sensitive data flow, which can fully reflect the possible sensitive or malicious behavior of the app.

After obtaining the final sensitive API set, we propose a method to calculate the app similarity. The process of calculating the similarity between apps in each functional category is shown in Algorithm 2. We find that the similarity between apps is not only reflected in the use of the same API, but also reflected in the similarity between APIs. The

Table 1 Strategy of merging samples.

Sample label	Solution
All are malware or all are benign apps	The new sample retains the original label
Some are malware and the others are benign apps	Increase the threshold of similarity and continue subdivision

Algorithm 2 Process of calculating the similarity

Input: The set of sensitive APIs; The set of sensitive data flows F; The API relation matrix R; The vector P; The set of apps; The similarity matrix S;
Output: The similarity between apps;
Initialize R with all elements = 0;
for each APIi in the set of sensitive APIs **do**
  for each APIj in the set of sensitive APIs **do**
    if APIi and APIj belong to the same relationship in F **then**
      $r_{ij} = 1$;
Initialize P with all elements = 0;
for $i = 0 \rightarrow$ range(size(The set of apps)) **do**
  for $i = j \rightarrow$ range(size(The set of sensitive APIs)) **do**
    if app_i includes API_j **then**
      $p_{ij} = 1$;
$S = PRP^T$; **return** S;

similarity between APIs includes the relationship between APIs, and it reflects the similarity between apps. We construct a relation matrix, and use it to express the relationship between APIs. For APIs that belong to the same data flow, the corresponding position in the relation matrix is 1. Then the similarity between two apps can be calculated by matrix multiplication.

We construct the API relation matrix R. That is, for the n-order matrix R, each row and each column correspond to each API in the final sensitive API set. For the element r_{ij}, if API_i and API_j has the extracted relationship, $r_{ij} = 1$, otherwise $r_{ij} = 0$.

In order to calculate the similarity between apps, we generate vectors $P[n]$ with the same dimension as the number of APIs in the set, and combine them into app matrix P. For the element p_{ij}, if app_i includes API_j, $p_{ij} = 1$, otherwise $p_{ij} = 0$.

Then the similarity matrix S can be generated by the matrix operation PRP^T, and the element s_{ij} represents the app similarity between app_i and app_j. Compared with other malware detection methods, our method can cover a wider range of API relationships, and does not add more elements to the vector of each app.

The purpose of calculating the app similarity is to find nearby data samples from the app to be tested, and judge whether the app is malware according to these samples. Each new app detection needs to calculate the similarity with each sample in the same functional category, and there are a large number of samples in the dataset. In order to reduce the amount of similarity calculation, we merge the samples in the dataset. The strategy is to set a similarity threshold, and merge the samples whose similarity exceeds the threshold. Because there are some similar samples which have different malicious or benign labels, we formulate corresponding strategies to solve this problem. The strategy is shown in Table 1.

The number of samples after merging is greatly reduced. When a new app arrives, the model only needs to calculate the similarity with the merged samples. Then it can screen out the nearest 30 samples. Finally, it determines the label of the sample to be tested according to the labels of nearby samples.

After setting the threshold, the amounts of samples in each similar sample set are different. Some sets are split after increasing the threshold, and the number of samples in the set is reduced. Therefore, we set weights for the newly generated samples. That is, the set with more samples has higher weight after merging, while the set with less samples has lower weight.

Through the above methods, we effectively reduce the amount of computation. We replace the distance in KNN algorithm with similarity between apps. The calculation of matrix operation is far less than the calculation of Euclidean distance between samples. At the same time, the relationship between APIs in the similarity calculation is fully considered to ensure the accuracy of classification.

4. Empirical Evaluation

Both our scheme and the baseline are implemented by Python and the experiments are conducted on a 3.8 GHZ×24 core CPU, 128 GB main memory PC. The PC is equipped with windows10 operating system. We employ the algorithms in the scikit-learn tool package [20] to train and test the machine learning model. Based on Python, scikit-learn tool package integrates common machine learning algorithms and some classic datasets, which is convenient for the experiments.

We choose Google Play Store as our input for app functional classification. The web crawler downloads 12792 apps from Google Play Store, and these apps are spread across 16 categories. The data we used in the experiments is described in Table 2. In the experiment, we choose 90% of the data as the training set and the remaining 10% as the test set.

4.1 Functional Classification Experiment

Number of APIS selected by HITS algorithm: HITS algorithm is employed to filter the APIs that are closely related to functionalities, and we set different weights to these APIs according to their authority values. In each category, the number of APIs filtered by HITS directly affects the total number of static features selected. Then it affects the classification accuracy of machine learning module. We select

Table 2 Data set.

Category	Number of apps
BOOKS_AND_REF.	795
COMICS	799
COMMUNICATION	794
ENTERTAINMENT	801
FINANCE	800
HEALTH_AND_FITNES	796
LIFESTYLE	803
MEDIA_AND_VIDEO	810
MUSIC_AND_AUDIO	805
NEWS_AND_MAGAZINES	790
PERSONALIZATION	793
PHOTOGRAPHY	804
SOCIAL	806
SPORTS	800
TOOLS	799
TRANSPORTATION	796

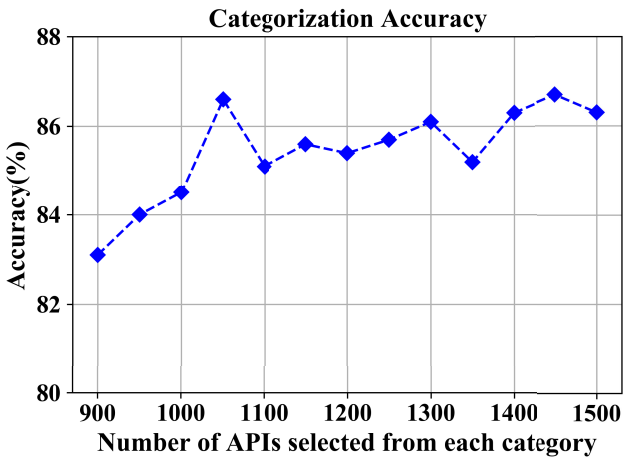


Fig. 4 Change of accuracy with quantity of APIs selected in each category.

500 APIs from each category to start the experiment, adding 100 APIs to each experiment. According to the preliminary results, we conduct more detailed experiments in the range of 900 to 1500, and the results are shown in Fig. 4. As can be seen from the figure, when the number of APIs filtered by each category is less than 1050, the classification results show an upward trend with the increase of the number. When the number exceeds 1050, the classification results fluctuate in a small range, and there is no obvious upward trend. Therefore, we finally selected 1050 APIs in each category to add to the final static feature set.

We use ensemble learning algorithm to experiment on 16 app categories. The experimental result of each category is shown in the figure. Here, we use the recall value as the accuracy of each category, that is, the ratio of the number of correctly predicted apps in each category to the number of apps used for test in that category. In multi classification problems, recall has the same value as accuracy.

As shown in Fig. 5, the classification accuracy of BOOKS_AND_REFERENCE and COMICS is low. The reason is these two categories are difficult to distinguish

Table 3 Descriptions of feature sets.

Feature type	Wang et al.	Our Solution
Request permissions	96	84
Filtered intents	126	34
Restricted API calls	34188	16800
Code-related information	5	0
Used permission	96	84
Hardware features	41	36
Suspicious API calls	78	0

Table 4 The precision results of each category.

Algorithm	Parameter
SVM	Linear kernel function,c=0.2
NB	MultinomialNB
KNN	11
CART	Default
RF	200

from other READING apps, and the main difference between them is the content theme. The categories with higher classification accuracy are NEWS_AND_MAGAZINES, SPORTS and PERSONALIZATION. These categories are quite different from other categories in functionality. Take SPORTS for example. In addition to applying for INTERNET, ACCESS NETWORK STATE and other common permissions, this kind of apps usually requests the permission of SENSOR_ENABLE. Besides, all functionalities of them are about users' health, so these apps are easy to be classified correctly. The apps in PERSONALIZATION provide user with personalized smartphone settings, so they include many APIs which is designed to change smartphone settings and are different from other apps in functionality.

Wang [17] uses ensemble learning algorithm to classify apps by extracting static features. On the basis of this method, we mine the relationships of API calls, introduce HITS algorithm to filter API features and set different weights to different APIs, so as to improve the classification accuracy. In order to prove the effectiveness of our method, we compare our method with Wang's method [17]. The number of features extracted by the two methods is shown in Table 3.

Our method uses fewer types and the number of features is less than the comparison method. Especially in the number of API features, we use less than half of the comparison method. In this case, we employ the same ensemble learning algorithm, and the final classification accuracy is 86.6%. The parameters of ensemble learning algorithm is shown in Table 4. Besides, we spend less time on training the machine learning module because we use fewer features. The experimental results show that the classification accuracy of the comparison method is 79.3%, which is about 7% lower than ours. The detailed classification results of each category are shown in the figure. As can be seen from Fig. 6, in most categories, the accuracy of our method is greater than, equal to, or slightly lower than the comparison method. Among the six categories with lower accu-

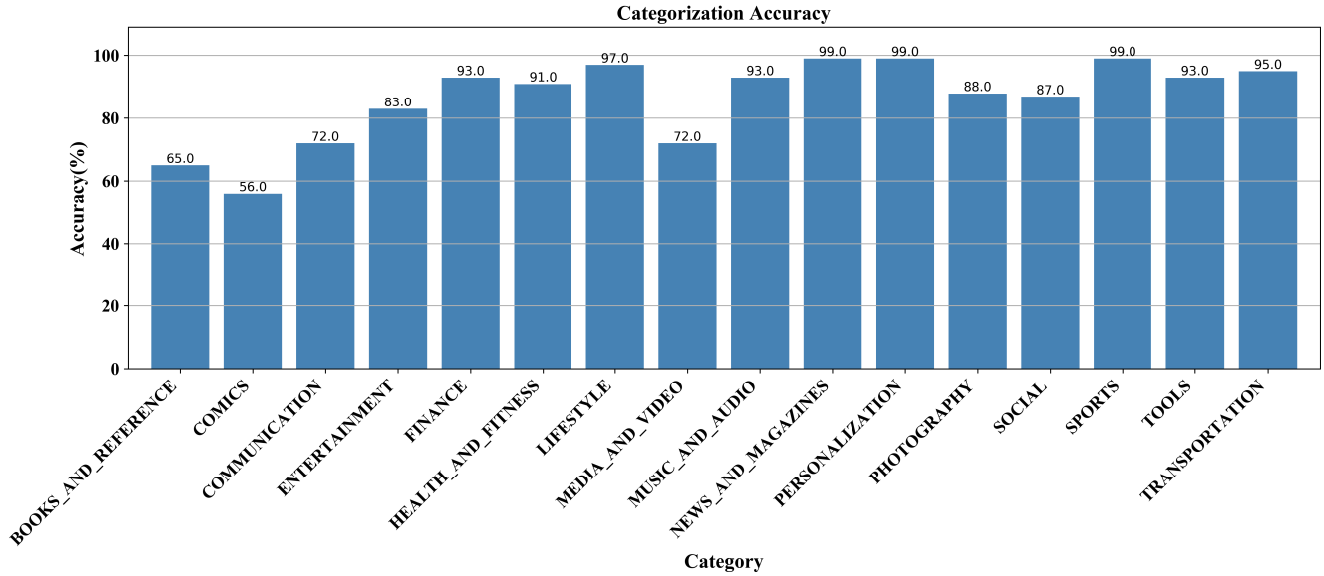


Fig. 5 Accuracy in each category.

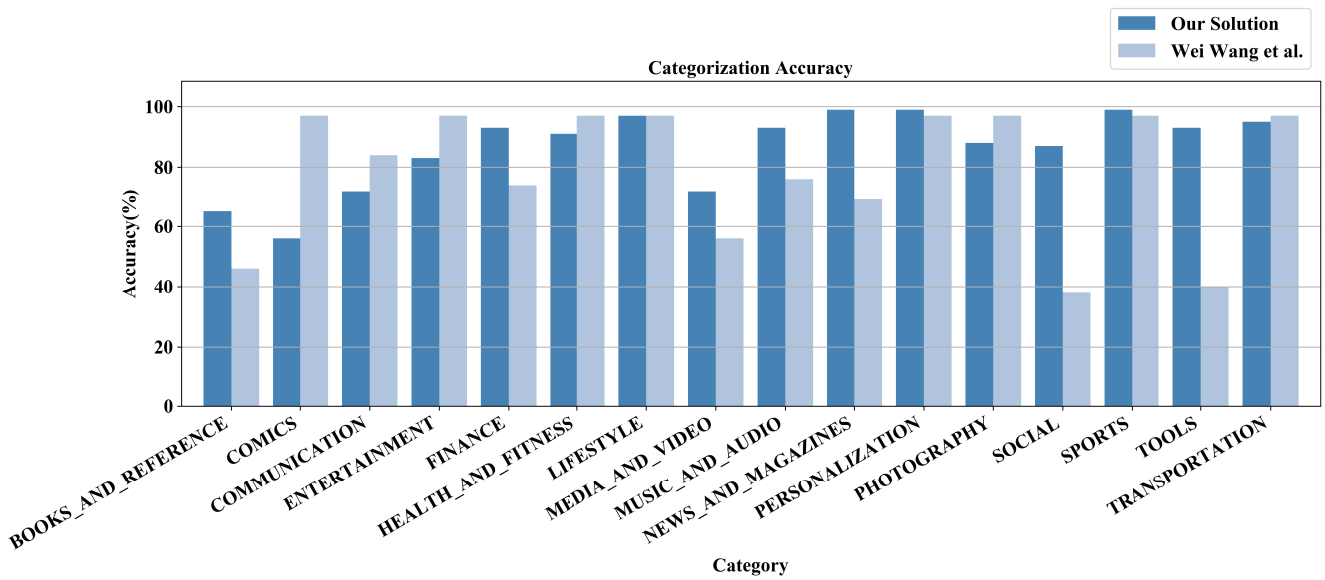


Fig. 6 Comparison of two methods.

racy than the comparison method, the accuracy of three categories has no significant difference in the classification results of the two methods. Only in the three categories of COMICS, COMMUNICATION and ENTERTAINMENT, the accuracy of the comparison method is over 10% higher than ours. After analysis, we found that in these three categories, the comparison method extracted more features than us. The main reason is that we control the quantity of APIs filtered out from each category. Consequently, the classification effect of some categories is not as good as the comparison method. The motivation of our method is to improve the overall accuracy of the classification. It is allowed that the accuracy of a small number of categories is lower than that of existing methods. Increasing the number of features in these categories will cause HITS algorithm to introduce a large number of additional APIs, which will reduce the detection efficiency. What's more, the introduction of these APIs will reduce the accuracy of other categories.

The precision results of each category are shown in Table 5.

Besides, there are some other functional classification methods, which use different static features and algorithms. We also implemented Karina Sokolova *et al.* [21] and AndroClass [22] as the comparative experiments. The experimental result is shown in Table 6. It can be seen that the effects of these methods are not as good as ours.

Table 5 The precision results of each category.

Category	Precision
BOOKS_AND_REF.	67%
COMICS	84%
COMMUNICATION	92%
ENTERTAINMENT	90%
FINANCE	79%
HEALTH_AND_FITNES	85%
LIFESTYLE	89%
MEDIA_AND_VIDEO	93%
MUSIC_AND_AUDIO	89%
NEWS_AND_MAGAZINES	96%
PERSONALIZATION	87%
PHOTOGRAPHY	85%
SOCIAL	83%
SPORTS	91%
TOOLS	47%
TRANSPORTATION	85%

Table 6 Descriptions of feature sets.

Method	Accuracy	Precision
Karina Sokolova	80.9%	81.8%
AndroClass	83.5%	87.4%
Ours	86.6%	90.2%

Table 7 Data set.

Category	malware	all
BOOKS_AND_REF.	296	1091
COMICS	315	1114
COMMUNICATION	302	1096
ENTERTAINMENT	345	1146
FINANCE	248	1048
HEALTH_AND_FITNES	264	1060
LIFESTYLE	278	1081
MEDIA_AND_VIDEO	343	1153
MUSIC_AND_AUDIO	301	1106
NEWS_AND_MAGAZINES	368	1158
PERSONALIZATION	345	1138
PHOTOGRAPHY	204	1008
SOCIAL	362	1168
SPORTS	250	1050
TOOLS	267	1066
TRANSPORTATION	274	1070

Table 8 The number of APIs filtered out by each category.

Category	Number of APIs
BOOKS_AND_REF.	487
COMICS	421
COMMUNICATION	314
ENTERTAINMENT	376
FINANCE	310
HEALTH_AND_FITNES	298
LIFESTYLE	345
MEDIA_AND_VIDEO	343
MUSIC_AND_AUDIO	389
NEWS_AND_MAGAZINES	396
PERSONALIZATION	397
PHOTOGRAPHY	395
SOCIAL	393
SPORTS	421
TOOLS	347
TRANSPORTATION	305

Table 9 The number of apps after merging.

Category	Number of apps
BOOKS_AND_REF.	474
COMICS	435
COMMUNICATION	495
ENTERTAINMENT	487
FINANCE	456
HEALTH_AND_FITNES	462
LIFESTYLE	421
MEDIA_AND_VIDEO	414
MUSIC_AND_AUDIO	401
NEWS_AND_MAGAZINES	403
PERSONALIZATION	408
PHOTOGRAPHY	423
SOCIAL	446
SPORTS	420
TOOLS	417
TRANSPORTATION	400

Table 10 Effect comparison of two strategies.

Strategy	Accuracy	Precision	Recall	F1-score
retaining all APIs	98.9%	98.9%	99.6%	99.2%
retaining public APIs	91.1%	89.9%	91.9%	90.9%

4.2 Malware Detection Experiments

The dataset consists of benign apps from Google Play Store and malware from Drebin dataset. We use Android functional classification method to distinguish the malware. We choose Accuracy, Precision, Recall and F1-score as the evaluation indexes of the experimental results. After functional classification, the amount of malware in each category and the number of all apps in each category are shown in Table 7.

We use SUSI and Flow Droid to obtain the APIs related to sensitive data flow in each functional category. The number of APIs filtered out by each category is shown in the following Table 8.

The similarity between apps is obtained after matrix operation. In order to reduce the computation, we merge similar samples. According to different situations of each category, we set different similarity thresholds, and merge the samples whose similarity exceeds the threshold. After merging, the number of samples in each category is shown in Table 9.

We set weights for the merged samples according to the number of samples before merging. Besides, we consider the weight when we judge whether the app is malicious. After merging, we design two strategies to determine the representation of new samples. The effect comparison is shown in Table 10. It is obvious that retaining all APIs is better than retaining only public APIs.

What's more, the detail experiment result of our scheme in each category is shown in Table 11. When an

Table 11 Experimental result of our method.

Category	Accuracy	Precision	Recall	F1-score
BOOKS_AND_REF.	0.991	0.991	0.996	0.994
COMICS	0.992	0.993	0.996	0.994
COMMUNICATION	0.990	0.988	0.999	0.993
ENTERTAINMENT	0.987	0.988	0.994	0.991
FINANCE	0.988	0.986	0.998	0.992
HEALTH_AND_FITNES	0.991	0.991	0.996	0.994
LIFESTYLE	0.994	0.991	1.00	0.996
MEDIA_AND_VIDEO	0.986	0.984	0.996	0.990
MUSIC_AND_AUDIO	0.987	0.988	0.995	0.991
NEWS_AND_MAGAZINES	0.990	0.989	0.996	0.992
PERSONALIZATION	0.990	0.992	0.994	0.993
PHOTOGRAPHY	0.986	0.990	0.993	0.991
SOCIAL	0.991	0.991	0.995	0.993
SPORTS	0.989	0.990	0.995	0.993
TOOLS	0.986	0.983	0.999	0.991
TRANSPORTATION	0.988	0.986	0.997	0.992
ALL	0.989	0.989	0.996	0.992

Table 12 Comparison with KNN algorithm.

Algorithm	Accuracy	Precision	Recall	F1-score
KNN algorithm	95.1%	95.3%	94.6%	95.0%
Our algprithm	98.9%	98.9%	99.6%	99.2%

Table 13 Comparison with detecting malware directly.

Functional classification	Accuracy	Precision	Recall	F1-score
No	95.6%	97.5%	93.6%	95.5%
Yes(Our scheme)	98.9%	98.9%	99.6%	99.2%

app is misclassified into a wrong category in our functional classification method, it will affect subsequent malware detection. However, in functional classification, misclassified apps are more likely to be classified as the other similar category. At the same time, the attributes extracted in the functional classification stage and malware detection stage are quite different. So those misclassified apps don't have a great impact on malware detection.

Compared with KNN algorithm, our algorithm is an improvement based on KNN algorithm. Therefore, we compare it with KNN algorithm directly. The experimental result is shown in Table 12. It can be seen that our scheme has higher detection accuracy than KNN Algorithm in most categories.

At the same time, we consider not to classify the apps firstly, and directly use our malware detection method for experiments. In this case, we find that more APIs are extracted, which results in a time-consuming process of calculating the similarity. Moreover, because there are more samples than before, the computational efficiency is greatly reduced. The experimental results are shown in Table 13.

We select two classic malware detection methods. We reproduced their methods and compared the result. The difference among them and experimental results are shown in Table 14.

Table 14 Comparison with other classic methods.

Related work	Algorithm	Accuracy	Precision	Recall	F1-score
Drebin [1]	SVM	94.5%	91.4%	87.3%	89.3%
PIndroid [23]	Ensemble learning	96.9%	94.9%	99.5%	97.2%
Our scheme	Improved KNN	98.9%	98.9%	99.6%	99.2%

5. Conclusion

We propose a scheme of Android malware detection based on functional classification. This scheme classifies the app firstly, and then detects the malware in the same functional category. The detection results show that our scheme is more accurate than the existing malware detection research works. We innovatively propose to apply HITS algorithm to the API feature screening process of functional classification, and propose a new method to calculate the similarity between apps. Using this method and borrowing KNN algorithm, we can detect Android malware in the same category. We have decompiled the apps, extracted and filtered features, built machine learning model, classified apps, and detected malware. Finally, we achieved a detection accuracy of 98.9%, which is about 2% higher than the existing Android malware detection method.

In the future work, we hope to consider more API relations and expand the feature range of similarity between apps. What's more, deeper analysis of code logic difference between malware and benign apps is meaningful. At the same time, we hope to mine the data flow of the app and analyze the detail relationship between the data flow and the malicious behavior of the app.

Acknowledgements

This work was supported in part by the National Natural Science Foundations of China under Grant No.61821001, and in part by the Fundamental Research Funds for the Central Universities.

References

[1] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," Network & Distributed System Security Symposium, 2014.

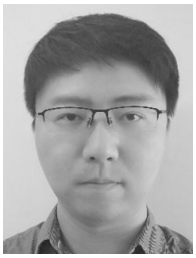
[2] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," IEEE Trans. Industr. Inform., vol.14, no.7, pp.3216–3225, 2018.

[3] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," IEEE International Conference on Wireless & Mobile Computing, pp.354–358, 2014.

[4] S. Liang and X. Du, "Permission-combination-based scheme for android mobile malware detection," IEEE International Conference on Communications, pp.2301–2306, 2014.

[5] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and API calls," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp.300–305, 2013.

[6] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," 2012 Seventh Asia Joint Conference on Information Security, pp.62–69, 2012.
[7] 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 Trans. Comput. Syst., vol.32, no.2, Article No.5, 2014.
[8] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A.K. Sangaiah, "Android malware detection based on system call sequences and LSTM," Multimedia Tools and Applications, vol.78, no.4, pp.3979–3999, 2019.
[9] Z. Ni, M. Yang, Z. Ling, J.-N. Wu, and J. Luo, "Real-time detection of malicious behavior in android apps," 2016 International Conference on Advanced Cloud and Big Data (CBD), pp.221–227, 2017.
[10] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," International Conference on Security and Privacy in Communication Systems, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol.127, pp.86–103, 2013.
[11] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model," Neurocomputing, vol.272, pp.638–646, 2018.
[12] S.Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for android malware detection.," IEEE Trans. Cybern., vol.49, no.2, pp.453–466, 2017.
[13] D.P. Gaikwad and R.C. Thool, "Intrusion detection system using bagging with partial decision treebase classifier," Procedia Computer Science, vol.49, pp.92–98, 2015.
[14] Z. Wang, J. Cai, S. Cheng, and W. Li, "DroidDeepLearner: Identifying android malware using deep learning," 2016 IEEE 37th Sarnoff Symposium, pp.160–165, 2016.
[15] Apktool, https://ibotpeaches.github.io/apktool/, 2019.
[16] Android Developers, http://developer.android.com/guide/components/fundamentals.html, 2019.
[17] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting android malicious apps and categorizing benign apps with ensemble of classifiers," Future Generation Computer Systems, vol.78, pp.987–994, 2018.
[18] P. Lam, E. Bodden, L. Hendren, and T.U. Darmstadt, "The soot framework for java program analysis: A retrospective," 2011.
[19] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y.L. Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," ACM Sigplan Notices, vol.49, no.6, pp.259–269, 2014.
[20] A. Swami and R. Jain, "Scikit-learn: Machine learning in python," Journal of Machine Learning Research, vol.12, no.10, pp.2825–2830, 2013.
[21] K. Sokolova, C. Perez, and M. Lemercier, "Android application classification and anomaly detection with graph-based permission patterns," Decision Support Systems, vol.93, pp.62–76, 2017.
[22] M.R. Hamedani, D. Shin, M. Lee, S.-J. Cho, and C. Hwang, "AndroClass: An effective method to classify android applications by applying deep neural networks to comprehensive features," Wireless Communications & Mobile Computing, vol.2018, pp.1–21, 2018.
[23] F. Idrees, M. Rajarajan, M. Conti, T.M. Chen, and Y. Rahulamathavan, "PIndroid: A novel android malware detection system using ensemble learning methods," Computers & Security, vol.68, pp.36–46, 2017.



Wenhao Fan received the B.E. and Ph.D. degrees from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2008 and 2013, respectively. He is currently an Associate Professor with the School of Electronic Engineering, BUPT. His main research topics include information security for mobile smartphones, parallel computing and transmission, mobile cloud computing, and software engineering for mobile internet.

Dong Liu received the B.E. degree from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2018, where he is currently pursuing the master's degree with the School of Electronic Engineering, BUPT. His main research interests include network and information security, android software, and machine learning.

Fan Wu received the B.E. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2004, and the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2009. She is currently an Associate Professor with the School of Electronic Engineering, BUPT. Her main research interests include network and information security, and wireless sensor networks.

Bihua Tang received the M.E. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1984. She is currently a Professor with the School of Electronic Engineering in Beijing University of Posts and Telecommunications. Her research interests include wireless sensor network and the Internet of Things.

Yuan'an Liu received the B.E., M.Eng., and Ph.D. degrees in electrical engineering from the University of Electronic Science and Technology, Chengdu, China, in 1984, 1989, and 1992, respectively. He is currently a Professor with the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, where he is also the Dean of the School of Electronic Engineering. His main research interests include network and information security, pervasive computing, wireless communications, and electromagnetic compatibility. Prof. Liu is a Fellow of the Institution of Engineering and Technology, U.K., the Vice Chairman of the Electromagnetic Environment and Safety of the China Communication Standards Association, the Vice Director of the Wireless and Mobile Communication Committee, Communication Institute of China, and a Senior Member of the Electronic Institute of China.