IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 26, 2017, accepted October 24, 2017, date of publication November 8, 2017, date of current version December 5, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2771470

Machine Learning-Based Malicious Application Detection of Android

LINFENG WEI[1], WEIQI LUO[1], JIAN WENG[1], YANJUN ZHONG[1], XIAOQIAN ZHANG[1], AND ZHENG YAN[2]

[1]Department of Computer Science, Jinan University, Guangzhou 510632, China
[2]Department of Information Security, Xidian University, Xi'an 710126, China

Corresponding author: Linfeng Wei (linfengw@gmail.com)

This work was supported in part by the National Science Foundation of China under Grant 61472165, Grant 61373158, Grant 61672014, and Grant 61502200, in part by the Guangdong Provincial Engineering Technology Research Center on Network Security Detection and Defence under Grant 2014B090904067, in part by the Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Important Scientific and Technological Achieve under Grant 2016B010124009, in part by the Natural Science Foundation of Guangdong under Grant 2016A030313350, in part by Special Funds for Science and Technology Development of Guangdong under Grant 2016KZ010103, and in part by the Guangzhou Key Laboratory of Data Security and Privacy Preserving.

ABSTRACT In this paper, we propose a machine learning-based approach to detect malicious mobile malware in Android applications. This paper is able to capture instantaneous attacks that cannot be effectively detected in the past work. Based on the proposed approach, we implemented a malicious app detection tool, named Androidetect. First, we analyze the relationship between system functions, sensitive permissions, and sensitive application programming interfaces. The combination of system functions has been used to describe the application behaviors and construct eigenvectors. Subsequently, based on the eigenvectors, we compare the methodologies of naive Bayesian, J48 decision tree, and application functions decision algorithm regarding effective detection of malicious Android applications. Androidetect is then applied to test sample programs and real-world applications. The experimental results prove that Androidetect can better detect malicious applications of Android by using a combination of system functions compared with the previous work.

INDEX TERMS Malicious applications of Android, machine learning, system function.

I. INTRODUCTION

Android has become the most widely used operation system today which takes about two-third of the mobile terminal market by the end of 2016 [1] and has attracted a lot of attention. It has been applied in more than 2.3 billion terminal platforms in the world, including computers, panel computers and mobile phones. A report in [2] showed that Android system has occupied 60.39% market share and is still expanding.

However, Android system suffers from attacks of various malicious applications due to its being an open platform. The cumulative samples of Android malicious application have reached 18.74 million [3], [4]. The newly increased number of malicious attacks threaten the security of Android system and lead to information leakage of Android users. Therefore, ensuring the security and effectiveness of Android applications is urgent.

At present, malicious application detection of Android based on machine learning becomes a hot topic in this field. Many researchers are devoted to improving the efficiency and effectiveness of detection by using machine learning to construct eigenvectors. Sanz *et al.* [5] proposed a malicious application detection method for Android based on application usage rights by using machine learning applied in static analysis. The machine learning can be used to train characteristics which are application usage rights, and get training models. Android malicious application process can be detected using the training model. However, a large number of Android applications have ''spillovers''. Moreover, a single feature will reduce the effectivenss of the system classification, which results in high false positive rates. In [6], a detection method based on contrast right mode is proposed. The enclamald classifier can be used to detect the existence of malicious applications. However, the deformation technology such as code obfuscation and dynamic loading, impairs the accuracy of the analysis. Edkrantz *et al.* [7] proposed a malware detection method based on parallel machine learning. The method converts the system call feature into vectors including the selected permission, the usage right and the sys-

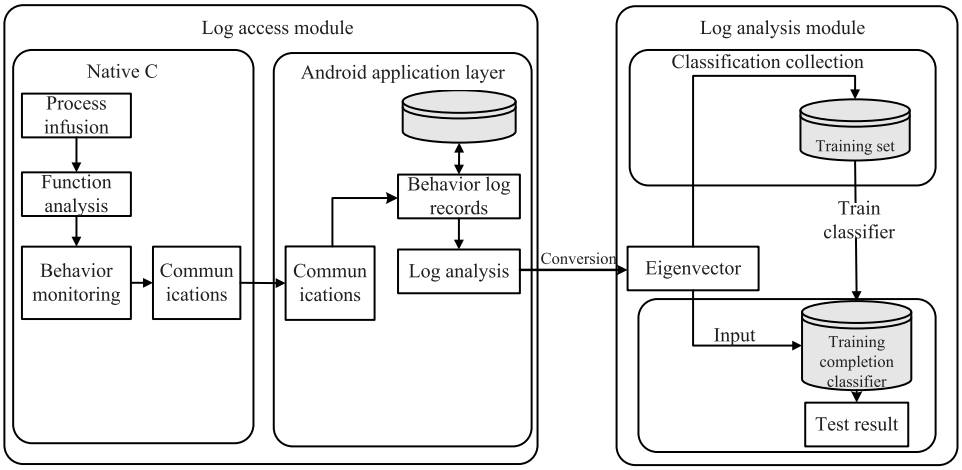


FIGURE 1. Schematic diagram of Androidetect detection system.

tem call sequence. The rbf and the shortest path graph kernel function are used to construct the dynamic model, which has high classification precision, but the false positive rate is high. In [8], a malicious software detection method is proposed based on system call. In this paper, the vector characteristics is generated by collecting the system invocation information and the corresponding frequency information variable construction model in the Android environment. The kNN classification algorithm is used to identify the normal behavior and the malicious behavior. The method can build the dynamic running characteristics of the application, but suffers from high false positive rate. Isohara *et al.* [9] proposed an Android malware detection method based on kernel behavior analysis. A rule base is created by extracting the names and parameters of malicious system calls. The invocation to rule base of the application is used to detect unknown malicious samples. The method is relatively simple, and therefore the test results are imprecise. The shortcomings of these two analytical methods are that they can not simulate the dynamic operation of the application or the constructed eigenvectors can not effectively detect the instantaneous attacks. Therefore, Qin *et al.* [10] proposed a MNDAM system, which can detect the abnormalities of Android system status, and analyze the existence of malicious applications. In [11], the DroidRanger tool [12] was developed to detect the variants of malware. Li *et al.* [13] proposed a dynamic stain detection method based on control dependency analysis. The malicious application software can be detected by analyzing the dependence of sensitive operation and pollution data. These methods can detect anomaly in the system, but can not determine which application has caused the problem [15], [16].

This paper presents an Android malicious application detection method based on machine learning, which identifies instantaneous attacks with low false positive rates. Our method constructs a feature vector based on the system call function and classifies Android applications based on source, where the feature vector is used as training data of the classifier. Compared with the above methods, the behavior description method of dynamic fine-grained application solves the problem of static detection against code obfuscation and poor encryption ability, and can also identify the instantaneous attack and better describe the application behavior in the stage of constructing eigenvector. In the application testing, the Android application is classified in accordance with the functional types to increase the success rate of detection and reduce false positives by improving the quality of data.

The structure of this paper is as follows. Androidetect system is given in Sec. II. Key technologies and algorithms is presented in Sec. III. Experimental design and analysis is showed in IV. In the end, the conclusions are provided in Sec. V.

II. ANDROIDETECT SYSTEM

This paper presents a malicious application detection method based on machine learning, which is implemented as the Androidetect tool that can automatically detect malicious applications. The system uses the process injection technology, Hook technology and inter-procedural communication that constructs the eigenvectors by extracting the characteristics of the Android application. An algorithm for application function class judgment algorithm is designed to establish the classification of normal and malicious applications. The two machine learning algorithms, namely the naive Bayesian and decision tree, are used to train and test classifiers.

The structure of Androidetect detection system is shown in Fig. 1. The system consists of a log access module and a log analysis module, where the log access module is used to obtain the behavior log corresponding to each sample and transform it into eigenvectors. The security threaten uses the transformed eigenvector to train classifier which detects malicious Android applications.

A. LOG ACCESS MODULE

1) **Code injection.** We adopt the code injection technology and Hook technology to complete the intercep-

tion process of a calling system function during the operation of malicious samples. In particular, the *.so* file is injected into the system_server to replace the function IOCTL and complete the interception.

2) **Function analysis.** We employ the Binder communication mechanism in Android processes to complete the analysis of system calls. The progress communication using the Binder mechanism is the process of system calling functions to perform sensitive behavior. According to the Binder protocol, by analyzing the command code BINDER_WRITE_READ, we can obtain two kinds of parameters of the intercepted system function *ioctl* command code and data to realize the analysis of the *ioctl* function.
3) **Behavior logging.** In this paper, the log protocol is used to output the analysis information by extracting and combining the underlying information. In the application layer, a specific program gets permission to open the service monitor log information and record log operations.
4) **Log analysis.** Since the output information is spliced by customized rules, we can import the information by using the Java string processing method to analyse and obtain data adopting the customize rules line-by-line.

B. LOG ANALYSIS MODULE

1) **Vector construction.** In the stage of feature description, the combination of system functions are used to describe the application behavior. Considering the relationship of system functions and permissions, sensitive permissions and sensitive APIs, they can also be considered as a combination of sensitive behaviors to describe the application behavior. At the same time, the combinations of sensitive behaviors are obtained dynamically from the log information to construct 34 different eigenvectors.
2) **Classifier training.** We use simple algorithms, naive Bayesian and decision tree to train the eigenvectors of the samples. The combination of sensitive behaviors in the log information are counted and used to construct the eigenvectors, where these eigenvectors are trained as input data.
3) **Application detection.** We adopt the training classifier to detect a large number of applications. We can classify the applications based on k nearest neighbor algorithm and 13 functional types, and then incorporate them into the training classifier which then use them to detect and determine security threaten.

III. KEY TECHNOLOGIES AND ALGORITHMS

A. FEATURE EXTRACTION TECHNOLOGY

The application behavior description method, process injection technology and Hook technology are combined to extract features from different types of Android applications including instantaneous attack behavior.

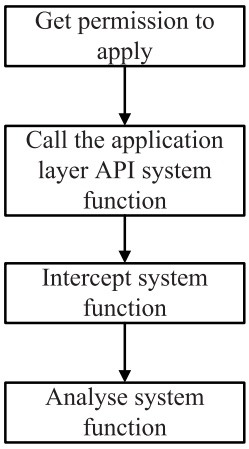


FIGURE 2. Schematic diagram of Android malicious behavior.

1) DESCRIPTION OF ANDROID APPLICATION BEHAVIOR

Different description of Android application behavior method affects the accuracy of behavior characteristics. Malicious behavior may send text messages until obtaining permission rights of SEND_SMS and RECEIVE_SMS before calling the functions sendTextMessage() and sendDataMessage() in the API layer. In order to prevent users from sending text messages in the background, malicious applications receive priority messages via registered SMS receiver SMS_RECEIVED, and call abortBroadcast() system functions to avoid sending text messages. The implementation principles of malicious behavior are shown in Fig. 2.

The schematic diagram of Fig. 2 shows that the Android application behavior can be described by permissions rights, API, and system functions. The research in [14] has shown the correspondence between system functions and permissions, and defined 137 permissions that may be applied. The work of [17] and [18] further classifies Android application permissions regarding sensitive permissions. These sensitive permissions can better describe Android malicious behavior. Some of the sensitive permissions are listed in TABLE 1.

The system function and the permission have the corresponding relations, the sensitive authority and the sensitive API also have the mapping relation [19]. The relationship between them can be understood as follows. The implementation of malicious behavior must first apply for sensitive permissions, and the granted permissions are used to call the application layer API, in order to intercept the system function. Therefore, the sensitive behavior of the application can be used to characterize system functionalities. As a combination of sensitive behaviors can be more dangerous than a single sensitive behavior, we select a set of single system behaviour and combined system behaviors, which leads to 34 items that need to be recorded, as shown in TABLE 2.

2) INTERCEPTION OF ANDROID APPLICATION BEHAVIOR

The interception of Android application behavior, in essence, is to replace system calls in an application. During an

TABLE 1. Some classical sensitive permissions.

Name	Permission description	Module	Classification
Make a call	Android.permission.CALL_PHONE allows the program to enter a phone number from a non system dialer	Telephone	Security
Access networks	Android.permission.INTERNET access network connections, which may generate GPRS traffic	Networking	Security
Send text messages	Android.permission.SEND_SMS sends text messages without user confirmation and consumes costs	Send message	Security
Read schedule	Android.permission.READ_CALENDAR allows the program to read the user's schedule information	Schedule	Privacy

TABLE 2. Sensitive Behaviors and their combination.

Single system function	Combination of system funcitons
getAccounts	setAudioSource+Internet
getAuthenticatorTypes	getContentResolver(telephony)+Internet
getLocation	getRiuiningAppProcesses+Internet
getCompleteVoiceMailNumber	getGroupIdLevel1+Internet
android.bluetooth.Ibluetooth.getState	getIccSerialNumber+Internet
android.bluetooth.Ibluetooth.isEnabled	gerMsisdn+Internet
android.bluetooth.IbluetoothManager.enable	getDeviceSvn+Internet
android.bluetooth.IbluetoothManager.disable	getCompleteVoiceMailNumber+Internet
getDataActivity	getContentResolver(settings)+Internet
sendmultiparttext	getContentResolver(sms)+Internet
setAudioSource	getContentResolver(media)+Internet
isAdminActive	getContentResolver(calander)+Internet
Itelephony.call	getInstal1edPackages+1nternet
sendtext	getAccounts+Internet
setWifiEnabled	getContentResolver(contacts)+Internet
	getContentResolver(browser)+lnternet
	getContentResolver(call-log)+Internet
	getLocation+Internet
	getDeviceId+Internet

application run, any request of calling a system function is sent to the system_server system process. For an important sensitive operation API, permissions rights are needed to pass through the system core, so it is useful to intercept calls to all sensitive APIs. During the operation of the code, the *ioctl* system function needs to be called, and process injection and Hook technology can be used to complete the behavior interception. At the same time, the Binder communication mechanism in Android processes is used to complete the analysis of system call function. This is implemented in the log capture module as shown in Fig. 3.

i) Injection of the *so* base and interception of behavior code. In the system, the log capture module employs the process injection technology, Hook technology and behaviour interception code. The process is shown in Fig. 4.

(a1) Call ptrace() to track and debug the target process system_server.

(a2) Call the MMAP() function to open up a large enough memory space in the target process space.

(a3) Copy shellcode into memory space.

(a4) Load customized *so* base. Call the Property function to write *ioctl*'s real address and replace *ioctl* by the new function address. Call the get_module_base function to load *libbinder* and complete the injection of *so* base.

(a5) Analysis of ELF files and interception of behavior code. Open the /system/lib/libbinder.so file to get the

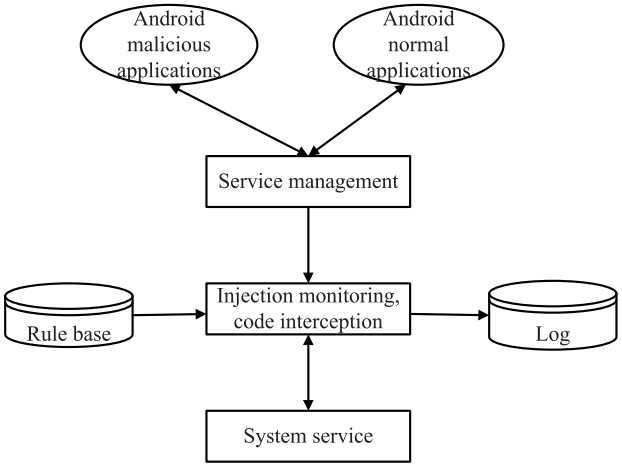


FIGURE 3. Schematic diagram of application system function interception.

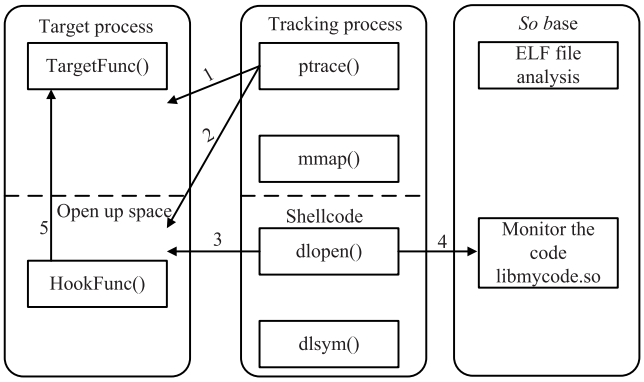


FIGURE 4. Schematic diagram of injection of *so* base and interception of behavior codes.

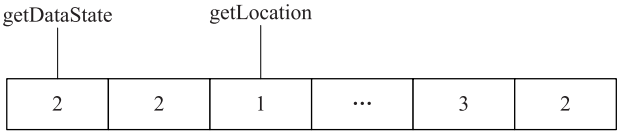


FIGURE 5. The basic form of eigenvector.

ELF head, the address, number and size of the Section Header zone table, and get the index number of the string table Section. The interception of the behavior code is completed.

ii) Analysis of *ioctl* function. The structure binder_transaction_data of *ioctl* function in the write_buffer data stores process communication data. Stripped of these data, we can get the application process ID and the system function name that is to complete the analysis of the *ioctl* function.

iii) Behavior log and analysis. After obtaining the *ioctl* function parameters, we can use specific rules to splice and the log output. The application layer is used to operate the log. Similarly, the application layer completes the analysis of log in accordance with the established rules.

3) CONSTRUCTION OF CHARACTERISTIC VECTORS

We can use the combination of system functions to construct the eigenvector of Android application, that is to construct a 34 dimensional eigenvector to describe the application samples. The basic form of a vector is shown in Fig. 5.

Specifically, we need to count the number of samples, the types of sensitive behavior and the number of times a sensitive behavior appearing in the sample after extracting the system function from the log information. Each vector corresponds to an application sample, and each dimension of the vector corresponds to how many times a sensitive behaviour or a combination of behaviours appear in the sample.

B. DETECTION ALGORITHM OF ANDROID APPLICATION

The detection of Android application is divided into three stages as follows.

(b1) Preparation. The eigenvectors are constructed by extracting the feature information of the training samples.

(b2) Classifier training. The eigenvectors constructed by the samples are used as input data, which can be trained by the classifier to obtain the classifier.

(b3) Application security detection. The trained classifier is used to classify the application and detect the security threaten.

1) TRAINING ALGORITHM OF CLASSIFIER

The classifier training algorithm is used to train the eigenvectors of the samples to obtain classifier. In order to validate the efficiency of application behavior description method, we adopts the Naive Bayesian algorithm and the decision tree algorithms to train the classifier respectively. The best classification performance algorithm is as the classifier training algorithm of the system.

a: NAIVE BAYESIAN ALGORITHM

Naive Bayesian is based on the Bias principle, which is to calculate the posterior probability of each category under concrete condition. The maximum posterior probability is considered as items to be classified subordinate category.

Assume a set of items to be classified $X = \{x_i|i = 1, 2, \ldots, l\}$. Every item x_i have n attributions $A_1, A_2, \ldots, A_n$ expressed as $x_i = \{a_1, a_2, \ldots, a_n\}$. Let the set of categories be $Y = \{y_h|h = 1, 2, \ldots, m\}$. Then judge the category of each x_i, that is

$$p(y_h|x_i) = argmax\{P(y_1|x_i), \ldots, P(y_m|x_i)\} \tag{1}$$

According to Bayes' rule, we get

$$p(y_h|x_i) = \frac{p(x_i|y_h) \cdot p(y_h)}{p(x_i)} \tag{2}$$

Considering the independence of the attributes in the item x_i, the conditional probability of each characteristic attribute in the class y_h is obtained from

$$p(x_i|y_h) = p(a_1|y_h) \cdot p(a_2|y_h) \ldots p(a_n|y_h) \tag{3}$$

Considering the higher computing performance of the naive Bayes algorithm and the decision of Android application security, a large number of applications need to be detected, so the algorithm can be used to train the classifier. The steps are as follows.

Step 1: Statistical analysis of the training samples, we get the set of category samples $\{y_1, y_2, \ldots, y_m\}$ and the corresponding condition probability set $(p(y_1), p(y_2), \ldots, p(y_m))$.

Step 2: The eigenvector set $\{x_1, x_2, \ldots, x_l\}$ can be constructed by running training samples and analyzing the log information of these samples, where the related eigenvector is $\{a_1, a_2, \ldots, a_n\}$ and the probabilities are $\{p(x_1), p(x_2) \ldots, p(x_i)\}$. We get the condition probabilities of each feature attribute $\{y_1, y_2, \ldots, y_m\}$ as follows.

$$\begin{array}{l} p(a_1|y_1) \cdot p(a_2|y_1) \ldots p(a_n|y_1) \\ p(a_1|y_2) \cdot p(a_2|y_2) \ldots p(a_n|y_2) \\ \vdots \\ p(a_1|y_m) \cdot p(a_2|y_m) \ldots p(a_n|y_m) \end{array} \tag{4}$$

Step 3: Combined Eqs. (2) and (3), we get the probability $p(y_1|x_i), \ldots, p(y_m|x_i)$.

Step 4: We get the maximum posterior probability of category y_h from Eq. (1).

$$p(y_h|x_i) = argmax\{p(y_1|x_i), \ldots, p(y_m|x_i)\} \tag{5}$$

Some behavior can not be performed in Android application process which will causes $P(A/y_h) = 0$ to affect the performance of the classifier. Therefore, the Laplace calibration is introduced in step 2, and the number of all characteristic attributes in each class is added 1 to improve the classification efficiency.

b: DECISION TREE CLASSIFICATION ALGORITHM

ID3 and C4.5 are widely used in decision tree classification algorithms. C4.5 uses information gain rates to select properties to overcome the shortcomings of the multi-attribute value selection in ID3 algorithm contributing to a higher efficiency. So C4.5 is used as a selection algorithm of the attribute decision tree classifier.

In the category item set X and Y, the X is divided into m subsets $\{X_h|h = 1, 2, \ldots, m\}$. Thus, the average information of X is

$$I(X) = -\sum_{h=1}^{m} P_h \cdot log_2 P_h \tag{6}$$

where $P_h = \frac{|X_h|}{|X|}$, $|X_h|$ and $|X|$ represent the number of elements in X_h and X, respectively.

In the set $\{A_1, A_2, \ldots, A_n\}$, suppose that A_j has q attribute values. Based on the attribute A_j, the classification item set X can be divided into q subsets $\{X'_1, X'_2, \ldots, X'_q\}$. The average information quantity of X is

$$I_{A_j}(X) = \sum_{j=1}^{q} \frac{|X'_j|}{|X|} I(X'_j) \tag{7}$$

Divide X by using the attribute set A_j, we get the information gain as follows.

$$G(A_j) = I(X) - I_{A_j}(X) \tag{8}$$

The information gain rate $R(A_j)$ is obtained by using C4.5 algorithm

$$R(A_j) = \frac{G(A_j)}{S(A_j)} \tag{9}$$

where $S(A_j) = -\sum_{j=1}^{q} \frac{|X'_j|}{|X|} \cdot log_2 \frac{|X'_j|}{|X|}$. The information gain rate $R(A_j)$ can be used as the basis for attribute selection. The process of training classifier using the C4.5 algorithm are as follows.

Step 1: Statistical analysis of the sample in the training stage, and the set of samples is obtained $\{y_1, y_2, \ldots, y_m\}$.

Step 2: Run the training samples, and analyze the log information of these samples to construct the eigenvector set $\{x_1, x_2, \ldots, x_l\}$ which corresponds to eigenvector $\{a_1, a_2, \ldots, a_n\}$.

Step 3: The eigenvector set $\{x_1, x_2, \ldots, x_l\}$ is divided into $\{X_1, X_2, \ldots, X_m\}$ according to $\{y_1, y_2, \ldots, y_m\}$.

Step 4: Calculating the average amount of information $-\sum_{h=1}^{m} P_h log_2 P_h$ of $\{x_1, x_2, \ldots, x_l\}$.

Step 5: Combined with Eqs. (5-7), we calculate the information gain rate of each attribute information $\{R(A_1), R(A_2), \ldots, R(A_n)\}$ in the set $A = \{A_1, A_2, \ldots, A_n\}$.

Step 6: The max information gain rate $R_{max}(A_j^*)$ is chosen from $\{R(A_1), R(A_2), \ldots, R(A_n)\}$ as split attribute to construct nodes of a decision tree.

Step 7: Cut the A_j^* attribute in set A, repeat the steps.

Step 8: Judge whether the set A is Ø, and if it is Ø, the decision tree is output. Otherwise, it returns to steps 5-7.

2) THE APPLICATION SECURITY DETECTION ALGORITHM

FUNCTIONAL CLASSIFICATION METHODS

The method used for classification of Android applications has a great impact on the classification effect at the classifier training stage, as well as on the detection result at the security detection stage.

In order to reduce false positive rates, the system adopts the application function classification method to divide Android applications into 13 categories, as shown in TABLE 3.

TABLE 3. Function categories of android application.

System security	Social communication	Audio and video
News reading	Life and leisure	Theme wallpaper
Office business	Photography	Shopping discount
Map travel	Education and learning	Financial management
Healthy care		

In the classifier training stage, we need to classify the adopted sample according to the functional classification methods, and then distinguish benign and malicious applications. In the security testing stage, we need to classify the applications according to the known categories. A large number of applications needs to be detected in the security detection stage. Since the sample class domains intersect and overlap, the system automatically discriminates application function categories based on the k nearest neighbor algorithm.

The k nearest neighbor classification algorithm works on the commonality of adjacent samples, that is, if most of k samples, which is nearest to a given sample to be classified, belong to a certain category, then the sample also belongs to the category. In the Android system permissions mechanism, we can find commonness based on the similarity of application permissions between similar applications.

Assume that the set of the category is $X = \{x_i|i = 1, 2, \ldots, l\}$. Each application item x_i to be classified applies the permission with an n dimension vector $x_i = \{x_1^i, x_2^i, \ldots, x_n^i\}$ (where n is the total number of permissions in the Android system, and in this test $n = 137$). The v-th

dimension element x_v^i in the permission vector of x_i represents the v-th application permission. If $x_v^i = 1$ holds, then the permission is applied, and vice versa.

In order to obtain a vector of estimated similarity, the latitude values in the vector are adjusted are as follows.

$$x_v^i = \frac{p_v^i}{n} log_2 \frac{D}{d+1}, \tag{10}$$

where p_v^i represents whether the i-th application has the v-th dimensional permission, n stands for the total number of Android system, D represents the number of applications to be tested in the sample, and d indicates that the number of application in the v-th permission.

Furthermore, the cosine of the vector denotes the similarity between the two applications as follows.

$$S_{x_i,y_j} = \frac{\sum_{v=1}^{n} x_v^i y_v^j}{\sqrt{\sum_{v=1}^{n} (x_v^i)^2} \cdot \sqrt{\sum_{v=1}^{n} (y_v^j)^2}} \tag{11}$$

where $S_{(x_i,y_j)}$ represents the similarity between x_i and y_j. The vector of x_i is $(x_1^i, x_2^i, \ldots, x_n^i)$, and the vector of y_j is $(y_1^j, y_2^j, \ldots, y_n^j)$. The function classification algorithm based on the application of k nearest neighbor classification method is as follows.

Step 1: Analyze the permission of application to be detected, we get $X = \{x_i | i = 1, 2, \ldots, l\}$, and the set of n-dimensional vectors$\{(p_1^1, p_2^1, \ldots, p_n^1), (p_1^2, p_2^2, \ldots, p_n^2), \ldots, (p_1^l, p_2^l, \ldots, p_n^l)\}$ generated by the corresponding set of permissions.

Step 2: The set of n-dimensional vectors is calculated combined with Eq. (8) to obtain a corresponding set of permission vectors of calculating the similarity

$$\{(x_1^1, x_2^1, \ldots, x_n^1), (x_1^2, x_2^2, \ldots, x_n^2), \ldots, (x_1^l, x_2^l, \ldots, x_n^l)\} \tag{12}$$

Step 3: Set $v = 1$, where v is the $v-$th element in the permission vectors of step 2.

Step 4: We calculate the similarity between the permission vector $(x_1^v, x_2^v, \ldots, x_n^v)$ and the other vector according to Eq. (11) to get the similarity set $S_{(v,v+1)}$, $S_{(v,v+1)}, S_{(v,2)}, \ldots, S_{(v,v-1)}, S_{(v,v+1)}, S_{(v,v+2)}, \ldots, S_{(v,l)}$.

Step 5: We choose the first K apps of highest similarity with the x_v application with the application x_v to be detected, where p represents the number of divided by K applications, r is application type $S_{(v,x_v(k,p))(r,w,m_r)|k\in\{1,2,\ldots,v-1,v+1,\ldots,l}$, i.e. $\{r|r = 1, 2, \ldots, p\}$, m_r represents the number of application types, i.e. $\sum_{(r=1)}^{p} m_r = k$, w is the w-th application of type r, i.e. $\{w|w = 1, 2, \ldots, m_r\}$.

Step 6: Calculate the average similarity between each type in the K applications and the x_v of the application to be detected in step 5.

$$u_{(x_v)} = \frac{\sum_{w=1}^{m_r} S_{v,x_v(k,p)(R,w,m_r)}}{m_r}. \tag{13}$$

Step 7: We choose the type of application of the highest similarity by analyzing u_{x_v}.

Step 8: If $v \neq n$ holds, then $v = v + 1$ and turn to step 4. Otherwise, the result of the classification is the output.

IV. EXPERIMENTAL DESIGN AND ANALYSIS

A. EXPERIMENTAL ENVIRONMENT

In this paper, we use C/C++ and Java to implement the malicious application detection tool Androidetect, where C/C++ is used in the application call function and Java is used for the other tasks. Androidetect detects 219 malicious application samples, where 102 applications are reading applications, and the remaining 117 applications are of unknown types, as are included in the virus database. The experiment is performed on the Mils mobile phone, where the baseband version of 8660-AAABQNBYA-g4271bc1, the Android system version of 4.4.4, and the kernel version of 3.4.0-g1ccebb5.

B. EXPERIMENTAL RESULTS AND ANALYSIS

1) ANALYSIS ON THE EFFECT OF BEHAVIOR INTERCEPTION ON MOBILE PERFORMANCE

Behavioral interception may lead to change in mobile performance, and ultimately affects the application classification and accuracy of application detection. In this section, we analyze the impact of behavioral interception on mobile performance.

In the experiment, we first install and run malicious applications that can send text messages and get private information through the background. The behavior interception through the inject, *librecorder.so* base files, *behaviour-recorder.apk* to complete process injection, *ioctl* function intercept and parameter analysis, and log record. The results in Fig. 6 show that the system can successfully intercept the application behavior.

```
Hook success
android.telephony.SmsManager.sendTextMessage pid:16383
android.telephony.SmsManager.sendTextMessage pid:16383
android.telephony.SmsManager.sendTextMessage pid:16383
android.telephony.TelephonyManager.getDeviceId pid:16383
```

FIGURE 6. Schematic diagram of the application behavioral interception.

In TABLE 4, the notations a and b represent the changes in the rate of CPU usage and memory usage before and after the injection of the System_server. The rate of CPU usage is below 1%, while the rate of memory occupancy is not obviously changed. The notation c represents the rate of occupation of resources, and the rate of CPU and memory resources are not high. Thus, application behavior interception does not affect moblie performance.

2) ALGORITHM ANALYSIS OF CLASSIFIER TRAINING STAGE

In TABLE 5, the confusion matrix of the classification result distribution is showed, where benign applications of the right classification TN, benign applications of the wrong classification FP, malicious applications of the right classification FN, malicious applications of the wrong classification TP.

TABLE 4. Occupation of resources in application behavior interception.

Resource usage	PID	The rate of CPU usage	Virtual consumption of memory	Actual usage of physical memory	Process
a	15530	0%	608644K	62448K	System_server
b	15530	0%	608804K	62588K	System_server
c	22222	0%	524500K	30876K	Com.zhongyj.behaviorrecorder

TABLE 5. The classification result distribution.

Inputs	Benign applications	Malicious applications
Benign applications	TN	FP
Malicious applications	FN	TP

TABLE 6. The classifier evaluation parameters.

Evaluate parameters	Equations	Meanings
TPR	TPR=$\frac{TP}{(TP+FN)}$	The proportion of malicious application of samples correctly classified
FPR	FPR=$\frac{FP}{(FP+TN)}$	The proportion of benign application samples of the wrong classification
ACC	ACC=$\frac{(TP+TN)}{(TP+TN+FP+FN)}$	The proportion of applied samples of the right classification

The classifier evaluation parameters are shown in TABLE 6.

In the experiment, the J48 decision tree classifier and the naive Bayesian classifier adopt 10-fold cross-validation to detect 200 samples, where 100 benign applications and 100 malicious applications respectively. The benign application categories include system security, lifestyle, shopping, and map tourism). The results are shown in TABLE 7 and TABLE 8.

Combined TABLE 6, 7 and 8, we get the bar graphs with the detection rate, false positive and classification accuracy in Fig. 7-9.

It implies that the classification accuracy of the two algorithms reach 82.5% and 86%, respectively, and the J48 decision tree algorithm is superior to the naive Bayesian algorithm in TPR, FPR and ACC. The system function is used to describe the behavior to distinguish benign and malicious applications.

3) ALGORITHM ANALYSIS OF IMPLEMENTATION STAGE

In the experiment, we still use the 10-fold cross-validation to test the selected 200 news reading (100 for benign and 100 for malicious applications, respectively), and verify the validity of the application function classification algorithm based on

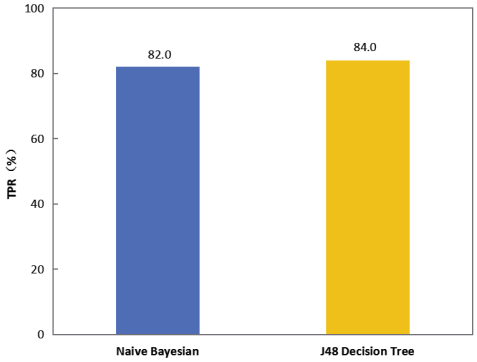


FIGURE 7. Schematic diagram of TPR.

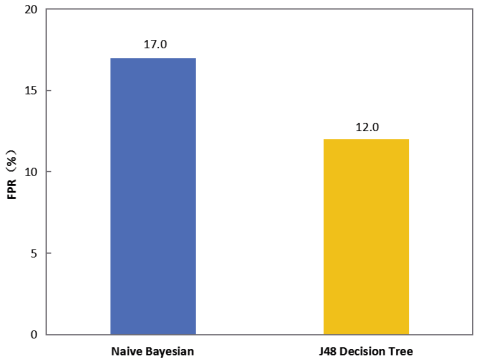


FIGURE 8. Schematic diagram of FPR.

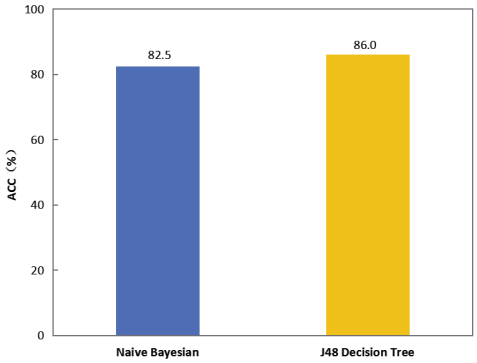


FIGURE 9. Schematic diagram of ACC.

k nearest neighbor algorithm. In addition, we also selected 180 hybrid application types of the training set (90 for benign and malicious applications, respectively) to verify the effect

TABLE 7. The results of J48 decision tree classifier.

Detailed accuracy by class									
/	TP rate	FP rate	Precision	Recall	F-measure	MCC	ROC area	PRC area	Class
	0.840	0.120	0.875	0.840	0.857	0.721	0.884	0.857	malicious
	0.880	0.160	0.846	0.880	0.863	0.721	0.884	0.836	benign
Weighted Avg.	0.860	0.140	0.861	0.860	0.860	0.721	0.884	0.847	/
Confusion matrix									
a	b	⟵classified as							
84	16	a=malicious							
12	88	b=benign							

TABLE 8. The results of the naive Bayesian classifier.

Detailed Accuracy By Class									
/	TP rate	FP rate	Precision	Recall	F-measure	MCC	ROC area	PRC area	Class
	0.820	0.170	0.828	0.820	0.824	0.650	0.907	0.908	malicious
	0.8300	.180	0.822	0.830	0.826	0.650	0.907	0.915	benign
Weighted Avg.	0.825	0.175	0.825	0.825	0.825	0.650	0.907	0.911	/
Confusion matrix									
a	b	⟵classified as							
82	18	a=malicious							
17	83	b=benign							

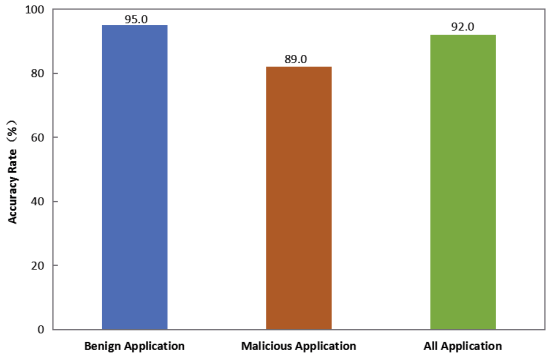


FIGURE 10. Schematic diagram of classification accuracy of classification algorithm.

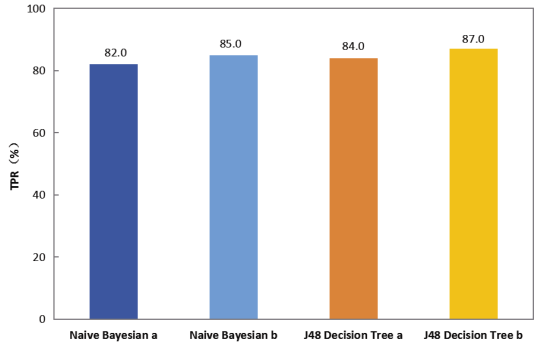


FIGURE 11. Schematic diagram of TPR comparison of *a* and *b*.

of data training and testing in accordance with the functional classification based on the type.

We first use the test set to classify the application function classification algorithm. If the result of judgement is the news reading, the classifier with the formal training set is used. If an error appears, the classifier of alternate training set is used.

In Fig. 10, the accuracy rate of 100 benign reading application classification reaches 95%. The accuracy rate of 100 malicious reading class application classification reaches 89%. The accuracy rate of the overall classification reaches 92%. Therefore, the proposed algorithm based on k nearest neighbor algorithm is effective.

The comparison bar graphs of TPR, FPR, and ACC, where a and b are two experiments in Fig. 11-13.

The results of TPR, FPR and ACC in experiment b are better than those in the experiment a, which proves that it is more effective to further divide the application according to the functional type. At the same time, using this classification method can more effectively reduce the probability that benign applications are wrongly determined as malicious applications.

C. COMPARISON WITH RECENTLY RELATED WORK

In order to further prove the effectiveness of the proposed method, the same test samples will be compared with some typical testing tools.

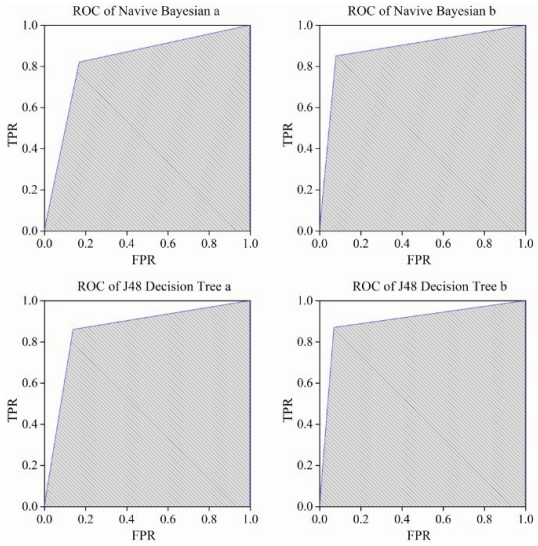


FIGURE 12. Schematic diagram of TPR and FPR comparison of *a* and *b*.

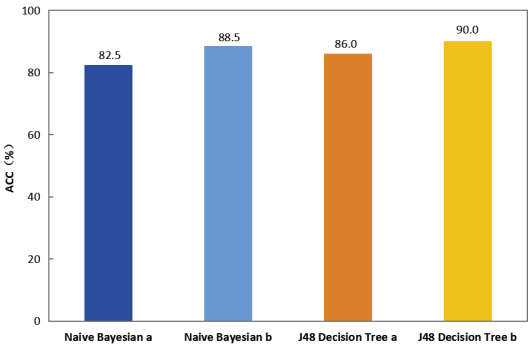


FIGURE 13. Schematic diagram of ACC comparison of *a* and *b*.

Comparisons with recently related work are shown in TABLE 9. In Andromaly system [20], we use the dynamic method to extract the API features. The disadvantage is fewer samples and not using real malicious application validation compared with BN, J48, K-means and other algorithms to detect malicious applications. In PUMA system [6], we adopt the application authority as characteristics, and use the random forest algorithm to detect malicious applications, but the result has high false positive rate. Peiravian and Zhu [21] proposed the combination of permissions and Android API calls as characteristic to improve the detection rate and the accuracy of classification. Amos *et al.* [22] chose the system state as a characteristic and used a real malware sample for verification.

To sum up, a variety of features are used in these systems to better reflect the behavior feature of malicious Android applications. The instantaneous attack can be accurately described by Androidetect system using a unified relationship of system functions, sensitive operation APIs and permissions. While other tools cannot determine the detection of abnormal behavior generated by which application. And Androidetect system uses the application of functional classification algorithm with better detection effects.

TABLE 9. Comparisons with recently related work.

Detection tools	Select characteristics	Analytical methods	Algorithms
Andromaly	API	Dynamic	BN, J48, K-means
PUMA	Authority	Static	Random forest algorithm
Literature tools	Authority, API	Combination of static and dynamic	SVM,C4.5
Literature tools	System status	Dynamic	No
Androiddetect tools	System call functions	Combination of static and dynamic	Naive Bayesian, C4.5, Application functional classification algorithm

TABLE 10. The relevant results.

Detection tools	TPR (%)	FPR(%)	ACC(%)
Andromaly	87.8	23.1	87.7
PUMA	91	19	86.37
Literature tools	92.8	21.2	93.6
Literature tools	87.8	23.1	87.7
Androidetect systems	87	7	90

Comparing with the relevant results, we find that Androidetect system has a high classification accuracy and low false positive rate in the detection of malicious Android applications. Androidetect system has a better detection result in the categories FPR and ACC, and a slightly lower TPR, as shown in TABLE 10.

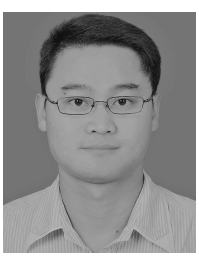
V. CONCLUSION

In this paper, the dynamic analysis technique is used to extract the feature of system functions to construct the eigenvectors. The classification model is established by naive Bayesian, J48 decision tree and Android application function type decision algorithm to realize the detection system Androidetect. The advantages of the system are as follows. First, the description method based on the system function can be used to identify the instantaneous attack. Second, the detection method based on the algorithm of the Android application function type can judge the source of the detected abnormal behavior. Finally, compared with the related work, Androidetect system has a better performance regarding FPR and ACC. In the future, we will improve the TPR in the Androidetect system.

REFERENCES

[1] H. Peng *et al.*, "Using probabilistic generative models for ranking risks of Android apps," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 241–252.
[2] Netmarketshare. *Operation Systems Market Share Reports*. Accessed: Apr. 18, 2016. [Online]. Available: http://www.netmarkethttp://www.netmarketshare.com/
[3] 360 Internet Security Center. *China Mobile Security Situation Report*. Accessed: Jan. 29, 2016. [Online]. Available: http://zt.360.cn/1101061855.php?dtid=1101061451&did= 1101593997

[4] K. Tian, D. D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged android malware with code-heterogeneity features," *IEEE Trans. Depend. Sec. Comput.*, to be published, doi: 10.1109/TDSC.2017.2745575.
[5] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "PUMA: Permission usage to detect malware in Android," in *Proc. Int. Joint Conf. CISIS-ICEUTE-SOCO Special Sessions*, 2013, pp. 289–298.
[6] X. Ping, W. Xiaofeng, N. Wenjia, Z. Tianqing, and L. Gang, "Android malware detection with contrasting permission patterns," *China Commun.*, vol. 11, no. 8, pp. 1–14, 2014.
[7] M. Edkrantz, S. Truvé, and A. Said, "Predicting vulnerability exploits in the wild," in *Proc. IEEE Int. Conf. Cyber Security Cloud Comput.*, Nov. 2015, pp. 513–514.
[8] Z. B. Cai and X. G. Peng, "Android malware detection based on system calls," *Comput. Eng. Des.*, vol. 34, no. 11, pp. 3757–3761, Nov. 2013.
[9] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for Android malware detection," in *Proc. IEEE Int. Conf. Comput. Intell. Secur.*, Dec. 2011, pp. 1011–1015.
[10] Z. Y. Qin *et al.*, "An Android platform for malicious software static detection method," *J. Southeast Univ.*, vol. 43, no. 6, pp. 1162–1167, Apr. 2013.
[11] *Analysis Reveals Flame Malware'S Process Injection Tricks*. Accessed: Aug. 14, 2012. [Online]. Available: https://threatpost.com/analysis-reveals-flame-malwares-process-injection-tricks-081312/76906/
[12] A. Mahindru and P. Singh, "Dynamic permissions based Android malware detection using machine learning techniques," in *Proc. 10th Innov. Softw. Eng. Conf. (ISEC)*, 2017, pp. 202–210.
[13] J. Z. Li *et al.*, "Android-based malware detection based on control-dependent analysis," *J. Tsinghua Univ. Natural Sci. Ed.*, vol. 54, no. 1, pp. 8–13, Jan. 2014.
[14] Google APIs. *Google APIsŠ Guide and Components*. Accessed: Jul. 11, 2012. [Online]. Available: http://www.android-doc.com/guide/components/index.html
[15] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in Android," *J. Parallel Distrib. Comput.*, vol. 103, pp. 22–31, May 2017.
[16] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Softw. Quality J.*, 2017, pp. 1–19.
[17] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," in *Proc. 17th ACM Conf. Comput. Commun. Security*, vol. 3. 2010, pp. 73–84.
[18] Z. Fang, W. Han, and Y. Li, "Permission based Android security: Issues and countermeasures," *Comput. Secur.*, vol. 43, no. 6, pp. 205–218, 2014.
[19] H. Bagheri, E. Kang, S. Malek, and D. Jackson, "Detection of design flaws in the Android permission protocol through bounded verication," in *Proc. 20th Int. Symp.*, Oslo, Norway, Jun. 2015.
[20] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for Android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.
[21] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in *Proc. IEEE Int. Conf. Tools Artif. Intell.*, Nov. 2013, pp. 300–305.
[22] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic Android malware detection at scale," in *Proc. 9th Int. Wireless Commun. Mobile Comput. Conf.*, Jul. 2013, pp. 1666–1671.

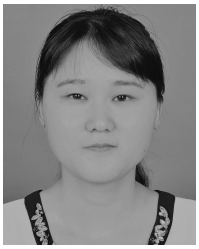


LINFENG WEI received the M.S. degree from the Department of Computer Science, Jinan University, China, in 2010. He is currently a Lecturer with the Information Science and Technology/Network Space Security Institute, Jinan University. He is also a Deputy Director with the Guangdong Province Network Security Detection and Protection Engineering Technology Research and Development Center. His research interests include cloud computing security, block chain security application, and mobile security.

WEIQI LUO received the Ph.D. degree in control theory and control engineering from the South China University of Technology, China, in 2000. He is currently a Professor with the Network Space Security Institute, Jinan University, Guangzhou, China. His research interests include network security, and management science and engineering.

JIAN WENG received the Ph.D. degree in computer science from Shanghai Jiaotong University, China, in 2007. He is currently a Professor with the College of Information Science and Technology, Jinan University, Guangzhou, China. His research interests include multimedia forensics and security and image/video intelligent analysis.

YANJUN ZHONG received the M.S. degree from the Department of Computer Science, Jinan University, China, in 2016. His main research is mobile internet security.



XIAOQIAN ZHANG received the M.S. degree from the Department of Mathematics, Jinan University, China, in 2013, where she is currently pursuing the Ph.D. degree with the Department of Computer Science. Her research interests include quantum secure communication, quantum computing, and quantum information processing.

ZHENG YAN received the B.Eng. degree in electrical engineering and the M.Eng. degree in computer science and engineering from Xi'an Jiaotong University in 1994 and 1997, respectively, the M.Eng. degree in information security from the National University of Singapore in 2000, and the Licentiate of Science and Doctor of Science in Technology degrees in electrical engineering from the Helsinki University of Technology in 2005 and 2007, respectively. She is currently a Professor with Xidian University, Xi'an, China, and a Visiting Professor with Aalto University, Espoo, Finland. Her research interests are in trust, security, and privacy; mobile applications and services; social networking; cloud computing; pervasive computing; and data mining.