DroidExaminer: An Android Malware Hybrid Detection System Based on Ensemble Learning

Zhongxiang Zhan[1], *Sai Ji*[1,2], *Wenying Zheng*[3*], *Dengzhi Liu*[4]

[1] *School of Computer Science, Nanjing University of Information Science & Technology, China*
[2] *Taizhou University, China*
[3] *School of Computer Science and Technology, Zhejiang Sci-Tech University, China*
[4] *School of Computer Engineering, Jiangsu Ocean University, China*
a213zhong@126.com, jisai@nuist.edu.cn, zhengwy0501@126.com, liudz@jou.edu.cn

Abstract

Android is an open-source mobile operating system, with more than 70% of the mobile market share, widely popular on various intelligent devices. At the same time, the number of new malicious applications keeps increasing every year. In this paper, we first discuss the advantages and disadvantages of various detection methods for malicious software. A single detection method can only cover specific types of malware. Therefore, we propose a system that combines static structural analysis and dynamic detection of malware. This system has dual detection capability, which consists of a client and a server. The client is a lightweight Android application that is used to obtain the relevant data information of the installation package. The server is responsible for static analysis of APK and dynamic running of monitoring logs to get the relevant feature information. Based on the feature information, the Bagging algorithm of ensemble learning is adopted, and the decision tree and random forest are combined to identify the malware accurately. We collected 4210 Android software samples, with malicious apps accounting for about 20% of the total. Cross-testing of malware detection on this sample set showed that DroidExaminer achieved approximately 96% accuracy in detecting malware. It can resist confusion and conversion techniques, and the test performance overhead is less. In addition, DroidExaminer can alert the user to the details of malware intrusion so that the user can prevent malware intrusion.

Keywords: Android malware, Software detection, Android software safety, Hybrid detection

1 Introduction

With the increasingly powerful functions of smartphones, they can replace many jobs of personal computers and enrich people's daily life. Android is a popular mobile operating system. According to the statistics of Statcounter [1], by December 2021, Android has occupied 70.01% of the market share of mobile operating systems. The source code of the Android system is open source, which is customized by major mobile phone hardware manufacturers and runs on various brands of mobile phones. Due to the openness of Android, it is more likely to become the target of malware attacks. Statistics show that the annual growth of Android malware is doubling. According to security firm ESET, Android banking malware continues to grow dramatically, increasing by 158.7% in 2021. Smartphones bring great convenience to people's work and life, but also hidden security risks such as malware and viruses, which damage the mobile phone system, steal private data, and consume users' network traffic.

The first step of detection is to analyze the characteristics of malicious behavior of Android applications and identify the operation mode of malicious applications. For example, decompress the Android application package, embed advertisements through malicious packaging, or load malicious code to consume traffic. Cam [2] proposed a system named uitRAAD that can be used to detect repackaged Android applications using representative graphs. He [3] proposed a novel method to detect repackaged Android malware based on Mobile Edge Computing (MEC). MEC servers can collect network traffic traces generated by both original and repackaged apps in large degrees and then analyze these traffic traces to detect repackaged malware. Tian [4] proposed a new Android repackaging malware detection technology based on code heterogeneity analysis. The code structure of the application was divided into multiple code subsets based on dependency, and each region was independently classified according to its behavior characteristics, thus realizing the detection method of malware based on multiple dependencies. However, software developers are limiting the repackaging of such malicious applications through hardened and shelled safeguards. Permpair [5] identified the dangerous functions of applications based on permission analysis, extracted permissions from the list files of applications, compared the graph structure of ma ware and benign samples, authenticated the permission declaration during program installation, and identified the malware with dangerous permission pairs. Kim [6] et al. obtained multiple static characteristic information such as code attribute string, function, permission, shared library, and environment from Android APK file and then detected and classified the malware through a deep learning algorithm. RTPDroid [7] is an approach to detect implicitly malicious behaviors and bugs brought by the Runtime

*Corresponding Author: Wenying Zheng; E-mail: zhengwy0501@126.com

DOI: 10.53106/160792642024012501009

Permission Model (RPM). Notions of user-aware contexts as well as user-aware call graphs are defined and utilized for the detection. Application developers prevent static analysis in several ways for software security. Static analysis is complicated when the core code is mixed up multiple times, or when the program is dynamically loaded and compiled into binaries. Surendran [8] proposed a novel TAN (Tree Augmented naive Bayes) based hybrid malware detection mechanism by employing the conditional dependencies among relevant static and dynamic features (API calls, permissions, and system calls) which are required for the functionality of an application. Marvin [9] as a hybrid detection technique, relies on a number of static API-based features to find multiple vulnerabilities in applications. These detection schemes are designed for malware analysts and are difficult for ordinary mobile device users to install on their devices that can detect and prevent malware.

Various detection methods can obtain the characteristic information of Android programs, then conduct sample training, and combine with machine learning classification algorithm to distinguish benign and malicious software. According to the running state of the application during detection, it can be divided into three types, namely static detection, dynamic detection and hybrid detection. Android malware is constantly evolving in an attempt to circumvent existing detection methods. The memory usage of application installation packages is increasing, and the resource consumption of static scanning and analyzing Android applications is rising. Some applications strengthen the confidentiality, confound the program code, and then harden and shell the APK package to prevent decompilation, making static detection more and more difficult. Dynamic detection depends on simulation environment execution, efficiency is relatively low, and a single detection method has certain limitations.

1.1 Contributions

Based on the existing detection methods and ensemble learning algorithm, we designed a detection system with broader coverage of malicious features and better performance. It can reduce the detection time and improve the accuracy of malware detection. The main contributions of the scheme are shown as follows:

- We design a hybrid Android malware detection system based on ensemble learning, which avoids the shortcomings of the existing static and dynamic single detection schemes and is a new scheme with dual detection capabilities.
- Improve the performance and accuracy of the detection system, and optimize the automated execution script of the Android simulation container. The cross-validation algorithm experiment is carried out through the mixed training data model of ensemble learning. With fewer hardware resources, the accuracy rate is 96%.
- The detection system will cover a broader scope of malicious feature detection, which includes the log monitoring of the malware through multi-level API and classification of the characteristic behaviors of the malware. In addition, the distributed design of the system enhances the scalability of the system, and the detection methods for new malware have the ability to update synchronously.

1.2 Related Work

The characteristic behavior of Android malware is constantly changing, and the harm way is more and more extensive. Existing research schemes have different emphases on the detection of malware. According to the running state of the program at the time of detection, there are three types, namely static detection, dynamic detection and hybrid detection. Machine learning is widely used in classification recognition, and the recognition accuracy of machine learning on different data sets is different. The researchers designed a variety of detection systems and achieved ideal detection accuracy on some data sets.

Static detection and dynamic detection have their respective focus. Static detection analyzes application code and file information to obtain static characteristics for classification and identification. Android application installation package is the APK file, through the decompile analysis tool can get AndroidManifest.xml, smali file and other information. Wang et al. [10] proposed a lightweight framework for Android malware identification. The proposed method combines network traffic analysis with a machine learning algorithm that is capable of identifying Android malware with high accuracy. Shen et al. [11] proposed a new technique to detect mobile malware based on information flow analysis. The structure, patterns and relations accurately capture the complex behavior exhibited by both recent malware and benign applications. Meng [12] proposed a precise semantic model of Android malware based on Deterministic Symbolic Automaton (DSA) for the purpose of malware comprehension, detection and classification. Drebin [13] is based on performing extensive static analysis, collecting as many application feature attributes as possible, and then classifying them through machine learning. Amin [14] proposed an anti-malware system that uses customized learning models, which are sufficiently deep, and are End to End deep learning architectures which detect and attribute the Android malware via opcodes extracted from application bytecode. Droidchameleon [15] is a systematic framework with various transformation techniques. Dynamic detection runs an Android application in a simulated environment (such as a sandbox [16-17]), and the runtime behavior characteristics obtained are called dynamic characteristics. Dynamic analysis includes system calls, application programming interface calls, network traffic, and CPU data. [18-20] are all based on the behavior detection of the Android runtime. Discover all kinds of malicious behavior characteristics of Android software in different states. DeepDroid [21] is a dynamic enterprise security policy enforcement scheme on Android devices. It is implemented by dynamic memory instrumentation of a small number of critical system processes without any firmware modification. [22-24] are some methods based on dynamic taint analysis of applications to monitor the use of sensitive data to distinguish malware. Maldeep [25] visualizes the feature information of the application, and identifies malware based on image difference comparison and machine learning algorithms. Liu

[26-27] proposes a method for secure mutual authentication and storage checking. It can be used for Android data integrity detection. Based on the simulation environment is a common method of dynamic detection. The operating system environment is relatively complex, resource consumption is large during detection, and detection accuracy is a challenge.

Hybrid detection consists of static analysis and dynamic analysis. It covers a wider detection range, reduces the limitations of a single detection method, and achieve an ideal state of balance. Madam [28] is a behavior-based detection method that analyzes features at the four levels of kernel, application, user and package to detect malicious behaviors of Android applications. DroidCat [29] is an application-level interface call and component communication dynamic characteristics analysis method to capture the function of the application execution structure and realize the classification and identification of malware. Whether static or dynamic, most recognition and classification methods rely on a variety of machine learning algorithms. Ensemble learning is a commonly used scheme to avoid the limitations of a single machine learning classification. A classification target with high accuracy is achieved by the ensemble learning algorithm. ICCDetector [30] uses the communication mechanism between components to identify the detection method of malware and uses a support vector machine and ten-double cross-validation for measurement experiments and performance evaluation. DroidFusion [31] fuses several different classification methods and achieves good performance and detection accuracy on experimental datasets. Ma et al. [32] proposed a combination method for Android malware detection based on the machine learning algorithm, and three detection models for Android malware detection regarding API calls, API frequency, and API sequence aspects are constructed. Droid detector [33] associates the features from the static analysis with features from the dynamic analysis of Android apps and characterizes malware using deep learning techniques. Feng [34] proposed an effective dynamic analysis framework, called EnDroid, in the aim of implementing highly precise malware detection based on multiple types of dynamic behavior features. Extracting behavior features through a runtime monitor, EnDroid is able to distinguish malicious from benign applications with an ensemble learning algorithm. As the characteristic behavior changes of Android malware become more and more complex, the combination of multiple algorithms and features of ensemble learning can better identify the malware.

2 Background

In this section, we will introduce the structure of an Android application, discuss the basic background knowledge of Android malware, and discuss Android malware characteristic behavior-related detection methods, which are essential knowledge needed for the design of the detection system.

2.1 The Structure of An Android Application

Using static detection of Android malicious applications, the Android program installation package decompiles multiple files, then analyzes the detailed information of each file, the information through the classification algorithm and machine learning methods to identify the malware. An Android application is a compressed package ending in APK, which includes information such as data area and central directory area. A signed installation package will add an APK signing block. A signature is a security mechanism to prevent the installation package from being tampered with or inserted by Trojan horses, effectively preventing the APK from being repackaged maliciously. The signature area information includes the block length, ID-value sequence, and fixed magic value information. The difference before and after the signature is in Figure 1.

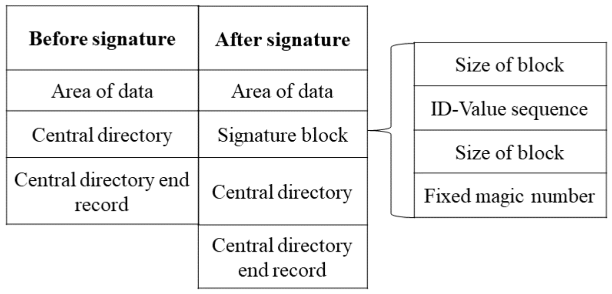

Figure 1. Differences before and after the installation package is signed

The Apk installation package contains assets, lib, res, and META-INF directories, and files AndroidManifest, Resources.arsc, and classes.dex. Table 1 lists the introductory information for each document. The above files need to be manipulated by the decompiler apktool. The Jadx tool can parse the decompiled class files.

Table 1. Apk file structure

File name	Describe
AndroidManifest.xml	Application declaration file
Classes.dex	Java source compilation file
Lib	Called library file
Res	Various resource files
Assets	Related file information
META-INF	Metadata

AndroidManifest.xml is an extensible markup language (XML) file. The AndroidManifest.xml has a start tag and an end tag for each piece of information. Applications request permissions using the tag of <permission>. There are also four types of components, which are activity, service, content provider and broadcast receiver. An activity represents a UI interface on the device that can interact with the user. Services are long-running background components without a user interface, and their function is to support background running tasks. Content providers are used to obtain system data resources. Broadcasting is a way for an operating system to interact data across processes. Each component has a different function and is isolated from the others. Android passes data using intents, which are messaging objects where one component requests an action from another. Typically, a component can use intents to start an activity, a service, or deliver a broadcast. There are two types of intents: explicit and implicit. The explicit intent is to start the component with the whole class name. Implicit intents do not require the name of the component.

The dex file is a compressed file after the java code source file is compiled into the class. Figure 2 shows the format information of the dex file. The code information of the application program can be obtained by parsing the dex file, and then the key API call graph can be extracted. The characteristic behaviors of the malware can be classified by sample training. With the prevention measures such as code confusion and dynamic loading, the detection accuracy of this detection method will be reduced.

Dex file structure			
header	File header	method_ids	Method identifier
string_ids	String identifier	class_defs	List of class definitions
type_ids	Type identifier	data	Area of data
proto_ids	Method prototype identifier	link_data	Data in a linked file
field_ids	Field identifier		

Figure 2. Dex compression file field information

2.2 Common Detection Methods

Android malware detection methods According to the running state of the application, the three main analysis methods are static detection, dynamic detection and hybrid detection.

Static methods parse program source code without executing the application, advance permissions, API calls, application library features and combine them with recognition algorithms or machine learning models to detect software. The advantage of static analysis is that it does not need to set a complex execution environment, but it cannot detect malicious software that uses transcoding, polymorphism, encryption technology and dynamic loading. Therefore, the static analysis method has a short time for the new family of malicious software.

The dynamic analysis method is to execute the application in the simulation environment, monitor the running characteristics of the program, and monitor various abnormal behaviors such as system calls, network access, file and memory modification, information access mode and high resource consumption. Dynamic detection needs to solve the performance problems of the running environment. Continuous runtime scanning of a running application puts a lot of pressure on the CPU of the device, resulting in a large amount of hardware resource consumption, significantly shortening battery life, and a challenge to the timeliness and stability of detection. The selection of machine learning technology has a significant impact on the accuracy rate. A single machine learning method has limitations, which only covers specific samples and is not widely used. After being extended to the ensemble learning method, the application can be accurately classified through experiments with various machine learning algorithms.

Static and dynamic methods are complementary and can be used in combination to increase code coverage. Hybrid detection is a solution that balances the advantages and disadvantages of static and dynamic methods, combining the two methods in different forms. The hybrid approach provides a better balance between resource and time efficiency, code coverage, method robustness and detection accuracy.

3 The Overview of DroidExaminer

DroidExaminer consists of two parts: a client and a server. An Android app runs on the smartphone, monitors the download of APK files, and the user uploads the APK to a server. The other part is the server side, which contains three services for processing data from the client, dynamic detection, and ensemble learning of the classification model. DroidExaminer has dual detection capabilities, the first detection is static, based on the application manifest file information, it is a fast and efficient detection process. Double detection is a dynamic detection process based on the container environment. The mobile application uploads the installation package to the server, installs it into the container environment through automated scripts and generates log files through automated testing. The following describes the detection methods and correlation of the two stages respectively.

3.1 First Detection

The first detection is a static detection method, which relies on static information related to Android application files. The Android app monitors the APK download file on the smartphone and detects file changes by tracking the phone's memory and SD card directory. If there is a new APK file, the detection service program is triggered. Users can upload the installation package to the server. On the server, the Apktool is used to decompile the APK file into the manifest and classes.dex, and then analyze and process the manifest. The manifest file contains the declaration of permissions and information about various components. The permission mechanism is one of the most important security mechanisms of Android. Sensitive information (location, phone and address book, etc.) and important resources (network, camera and bluetooth, etc.) need to apply for related system permissions in the manifest. Generate feature vectors based on permissions and component information.

The feature vector obtained by the first detection generates three text files, namely package.txt, permission.txt and component.txt documents. These three files hold the values of various attributes in the APK file and do not contain source code information. These attribute values will filter out duplicate data, build three related documents according to benign software and malicious software respectively, count the frequency of benign software and malicious software, and sort the attribute values according to the frequency. The values of these feature attributes are filtered according to the Chi-square test method and are finally used as input data for the ensemble learning model.

3.2 Double Detection

Double testing is a dynamic testing process. The server obtains the installation package (APK) file from the mobile phone. Start the automated script to install APK into the container environment. Run this Android application and call the automated test script. Android operating system is running on the container environment. In the dynamic detection stage, we developed an automated test script program to improve detection efficiency by reducing the repetition rate. In order to better detect the state of the running environment, open-source lightweight containers are used. We added a log tracking method to the key API of Android source code and then compiled and packaged it into a file named android.img. Android container environment loads this IMG file, and the API call information can be obtained when the application runs. According to this log information, feature vectors are generated and input into the ensemble learning model.

4 Feature Discovery and Computation

4.1 Feature Selection

In order to get enough log information, we have a hierarchical classification of Android's key APIs. The call resources from the API are divided into five types, and these types of APIs are tracked and monitored. These are application layer APIs, Android framework APIs, virtual machine APIs, system capability APIs, and special API types. Table 2 lists the details of each type of API.

Table 2. API type classification, these API classes and methods are more related to the behavioral characteristics of malware

API type	Invoke classes and methods	Description
Application layer API	ContentProvider	Some predefined content resources
	Context	Provides application context information
	Intent	The intention to transmit data
Android framework API	ActivityManager	Active interface management class
	PackageManager	Android package management class
	Telephony/SmsManager/LocationManager	Call, SMS and location services management classes
Virtual machine API	DexClassLoader	Load files such as class and dex externally
	Runtime.getRuntime.exec()	Runtime executes Linux commands at runtime
	LoadLibrary	Method of dynamically loading library files
System capability API	NetworkInfo/WifiManage/ConnectivityManager	Network and connection related classes
	DownloadMananger	Download management class
	HttpURLConnection	Universal network connection
	myPid() and killProcess() in the Process class	Process management class
	IO package	File read-write class
Special API	The Crypto class	Encryption, decryption, and key negotiation operation

The characteristic data of the application obtained from the two detections represents the structured form of the sample, and it is important to get the key information right. The representative characteristic set of malicious behavior is adopted. It can establish the recognition model with high accuracy. Application layer APIs are used by applications to obtain system content resources, program context information, and interaction information between multiple activities and services. Content providers can access sensitive information on mobile phones without obtaining the necessary authorization. The Context class provides global application information. Intents communicate between multiple active interfaces with MIME data information. The Android framework API provides all kinds of resource information such as active interface, installation package, phone and SMS. The virtual machine API monitors the outsourcing of loading classes and running dangerous Linux commands and parameters. This type of malware cannot be detected in static detection, but can only be detected through dynamic monitoring. The System Capability API is necessary to monitor process management, file reading and writing, and network data monitoring. Many malware programs send sensitive information over the network to external servers. Special APIs are encryption and decryption related calls that, if used for malicious encryption, will destroy the availability of the system. DroidExaminer's detection mechanism selects key features based on API calls. In addition to the common management class log tracking in Table 2, some methods to obtain key privacy data will be included in the tracking scope. For example, getDeviceId(), getSubscriberId(), sendTextMessage() and other calls to access sensitive data and resources of the phone, the malware tries to get this information without the user's consent and abuse it. These API calls will make the application suspicious, helping the application to be accurately labeled as malicious or benign. The calculation of collecting the API list and API call frequency of each application may be relatively large. We obtain the key API call information according to the five types, which makes the calculation cost lower, the feature set is strongly correlated, and the number of API calls to be considered is reduced. Some redundant data and irrelevant feature information are eliminated. These features are used for identification and classification. The label coding method we adopt is set to 1 if a feature exists, and 0 if it does not, thus the initial feature vector data set is constructed.

4.2 Feature Filtering

The feature data set obtained in the detection process is screened and filtered to improve the efficiency and accuracy of the learning model. The feature screening of classification problems usually includes the Chi-square test, F-test classification and mutual information regression. We use the Chi-square test algorithm to filter the feature data. This is a common statistical method used to determine whether there is a correlation between two features. The association of a feature *X* in an Android app with malware. We can use the following four-cell Table 3 data case to illustrate.

Table 3 is an enumeration of the data, it can be seen that the proportion of malice containing feature *X* is higher, when we cannot exclude the sampling error of this difference. Assuming that feature *X* is independent of whether it is a malicious application, the probability of an application belonging to the malicious category is *P*, *P* is the proportion of all malware to the total, which is 61.8%.

Table 3. Sample data for characteristic *X*

Group	Malware	Not malware	Total
Not include *X*	20	24	44
Include *X*	35	10	45
Total	55	34	89

X is a characteristic attribute. The data in the table indicates which software contains *X*. Calculate the correlation between X and malware.

Table 4. Theoretical data for characteristic *X*

Group	Malware	Not malware	Total
Not include *X*	27	17	44
Include *X*	28	17	45

The data in the table is recalculated according to the probability *P*, as a theoretical value for comparison.

The hypothesis of irrelevance generates a new four-cell Table 4 of theoretical values. The x^2 value is calculated by the following formula, where A is the actual value and T is the theoretical value.

$$x^2 = \sum \frac{(A-T)^2}{T}. \tag{1}$$

x^2 is used to measure the difference between the actual value and the theoretical value. After obtaining the x^2 value, query the Chi-square distribution critical value table. There is a value of the degree of freedom in the query, where the degree of freedom is 1. The possibility that an application contains feature *X* is less than 0.5% independent of whether it is a malicious application, indicating that the probability that feature *X* is associated with malicious applications is greater than 99.5%. The filtered feature set reduces the time spent on irrelevant features in subsequent classification steps and improves the efficiency of ensemble learning.

4.3 Ensemble Learning Model

Machine learning is used to classify Android characteristic data and judge whether the Android software is malicious or benign. There are many kinds of classification algorithms in machine learning, and different machine learning algorithms have different accuracy rates on the same data set. We use an ensemble learning algorithm to classify a large range of data sets. There are two paradigms of ensemble learning methods, namely serial ensemble method and parallel ensemble method, represented by AdaBoost and Bagging respectively. The serial ensemble method is the serial association of multiple learners before and after. The learning of the parallel ensemble method is independent of each other, and the final conclusion can be judged according to the results of the mutually independent base classifiers. These two methods are superior to the single classifier and can significantly reduce the error. We're going with a random tree, which is Bagging's way of leveling up. This method refers to random feature selection. When each decision tree selects the segmentation point, the random forest selects a feature subset at first, and carries out the traditional segmentation point selection on the subset. The training set is divided into several samples and the bagging sampling technique is used to carry out the partial sampling. By combining the classification results of multiple classifiers, the algorithm can classify the test samples. The algorithm has a better classification effect and generalization ability than a single classifier.

The detection classification stage consists of two parts, namely the training stage and the detection stage. In the training stage, the decision tree set and the corresponding weight value are generated through multiple training sample subsets. We sampled the total samples in a put-back way and segmented N sample training subsets altogether. The probability that each sample in the training set fails to be extracted is

$$p = (1-\frac{1}{N})^N. \tag{2}$$

When $N \rightarrow \infty$, $p \approx 0.368$, indicating that there is a 36.8% probability that the extraction process of each sample set is not extracted. The extracted feature attributes are selected to perform node splitting according to the decision tree generation algorithm, and pruning is not carried out in the splitting process, so as to obtain a decision tree from the training subset. The above steps are repeated for K times to establish K decision trees and generate a random forest.

The random forest was used for the classification and detection of the test sample set. All decision trees had K classification results, and the classification effect was evaluated. Finally, the test sample category was determined by the voting principle. Since the feature subset is randomly selected, the decision trees are independent of each other during algorithm training, which is a parallelization method. Each decision tree has a different effect on our subsequent voting, so add a concept of weight. For the decision tree T, T is applied to the sample set S for classification, and the true category of each sample is known. The number of samples for the correct classification of T is O. By comparing the classification conclusion with the real category, the accuracy P_T is calculated.

$$P_T = \frac{O_T}{S_T}. \quad \textbf{(3)}$$

P_T was used as the weight of the decision tree, and the samples were classified and weighted by the random forest classifier. In the detection process, T represents the random forest size, and $T_{(x)}$ represents the result after the classification test, with the value of 1 or 0. C is the final category of the sample, so C can be calculated as follows.

$$C = max(\sum_{t=1}^{T}(T_{(x)}P_T)). \quad \textbf{(4)}$$

Random forest is an important representative of the ensemble learning method. The calculation method used in this paper has been cross-tested on multiple samples and achieved the expected recognition accuracy. Due to the modular design adopted by DroidExaminer, more ensemble learning methods can be introduced in future experiments.

5 System Design

In this section, we'll cover the architecture and implementation details of DroidExaminer with the goal of designing a detection system suitable for real-world application scenarios. The system has desirable scalability and accuracy, and the detection ability is updated as the malware evolves. There is a good balance between system performance and detection accuracy. Android static detection relies on decompilation tools, which extract the APK file and retrieve the contents. Restore the relevant information to the source file according to the compressed file format. Parsing code files is time-consuming. As APK anti-decompile algorithms become more and more complex, the source file information obtained is incomplete, which greatly reduces the accuracy of static detection. The running sandbox environment of dynamic detection is complex. With the rapid upgrade of the Android operating system, many detection environments cannot meet the requirements of dynamic detection, resulting in a significant increase in the complexity of dynamic detection. Dynamic detection execution relies on automated scripts, which can be a major challenge to the functional coverage of an application. In the process of designing DroidExaminer system, both static and dynamic detection capabilities are realized. Figure 3 is the framework flow chart of DroidExaminer. Static analysis obtains the installation package from the mobile end and uploads it to the DPS module of the server. DPS analyzes the package information and manifest file information of APK. With the development of code obfuscation and APK shell hardening, it is more and more difficult to analyze dex static. Therefore, dex analysis is abandoned for static detection and supplemented by dynamic analysis. The second step in double-checking is to build a virtual runtime environment for dynamic testing. When the dynamic analysis is used, the classification time may be long, and this aspect of the design is automated. After the APK is uploaded to the server, it is automatically installed on the VM. The automated test script starts the application, covers as much functionality as possible, and generates the appropriate monitoring logs. Feature information is obtained from monitoring logs, ensemble learning and classification is performed, and the interpreted classification result is returned to the mobile terminal. As the features of Android malware are constantly evolving, a single machine learning classification method has a low classification accuracy for some malicious features. We adopt the ensemble learning method and modular design, which can update and extend the detection service on the server side.

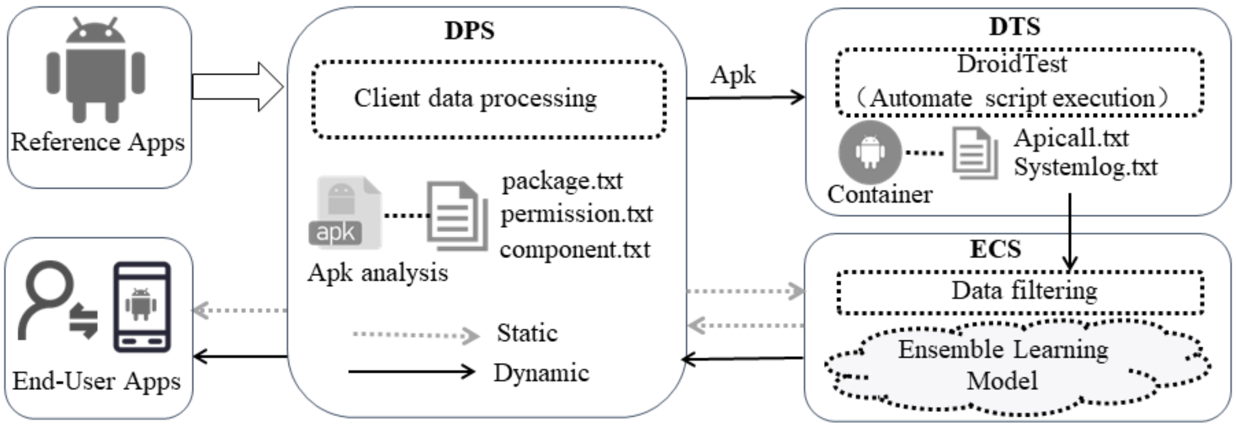

Figure 3. Workflow of DroidExaminer

(The dotted line represents the static detection process of hybrid detection, and the solid line represents dynamic detection.)

Considering the advantages and disadvantages of various detection methods, DroidExaminer is designed as a hybrid detection system that takes advantage of both static and dynamic detection. Optimize the execution process algorithm, and improve the efficiency and performance of the system. A scalable distributed service with four key components: an Android application, a Data processing service (DPS), a Dynamic test service (DTS), and an ensemble learning classification service (ECS). Classification service based on ensemble learning stochastic forest model. By using cross-training sample data experiments, the system achieves high efficiency and high accuracy.

The first component is an app that runs on an Android phone and interacts with the user. The app monitors new APK files in the phone's file directory and allows the user to select which apps to scan. In the Android operating system, APK is an executable program that resembles a compressed package. It contains two core files, the manifest file, which is the key file for static detection, and the DEX file, which is compiled from the executable code of the application. Our application sends the APK file to the server component, which returns the analysis results. In the case of malware identification, the information returned also includes the reason why the application is classified as malware.

The server consists of three components: Data processing service (DPS), dynamic detection service (DTS), and ensemble learning classification service (ECS). The DPS connects to the client app through HTTP. When it receives the APK file, it also obtains the information about the APK file on the mobile terminal. The APK file is quickly and statically parsed into three txt files for static detection. Dynamic detection transfers the APK file to the DTS, which installs the APK in the virtual environment and runs it, simulating the real detection scenario of the mobile Android operating system. The image file of the Android operating system ends with IMG. We modified the key API of the framework layer of the Android system source code, added a certain format of log information, and then recompiled the source code into the IMG image file to mount and run on the virtual machine. The APK installation package is installed into the virtual environment through a python script, and the automatic script DroidTest is used for dynamic coverage test. The log file information is generated. The feature vector is advanced from the log file, and the feature information extracted twice is sent to ECS for identification and classification, and the classification results are fed back to the mobile application. The Android phone will receive the result information twice. The first classification detection is to realize the fast response in the real scene, and the detection conclusion is usually reached in about 10 to 15 seconds. The second test is a dynamic test that usually takes between five and eight minutes, depending on the inherent complexity of the Android app.

We classify and describe the signature and permission information of static detection and the API call log information of dynamic detection. The following describes the design of four key capabilities of the system. One is to monitor the APK file changes of file directories on the mobile terminal, monitor the file directories downloaded by default through the FileObserver class, and prompt the user whether to detect the downloaded APK file. Users can also select the APK file in the corresponding directory to upload the APK to the server for detection.

The second key capability is that DPS gets information about APK installation package signatures, permissions in manifest files, active components and so on. This information generates three files: package.txt, permission.txt, and component.txt. We removed the duplicate attributes, each as a line message, and the three files will be uploaded to the ECS.

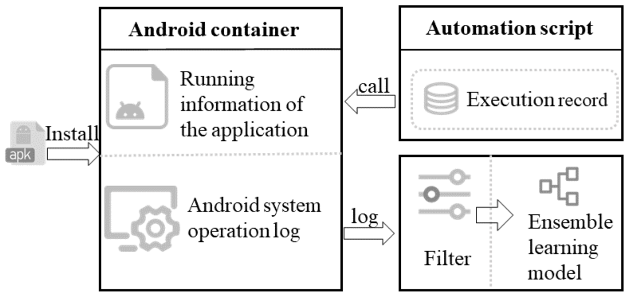

Figure 4. Dynamic detection process diagram

The third key capability is the dynamic detection process for APK installation packages to be uploaded to the server's virtual environment. Figure 4 is the flow chart of dynamic detection. Includes a container, an automated execution script, and an ensemble learning classification model. We optimized the automatic execution algorithm. When the server obtains the APK file, it triggers the automatic detection algorithm program and installs the APK into the Android container environment. First, determine whether the service program of the container environment is running. In the case of normal operation, install the APK file into the container, and then start the Android application to execute the automated test program. Through the operation of the program, obtain the call API log and system feature information of the program. In the execution algorithm, we add a repetitive operation filter judgment. An activity interface is considered to have been fully tested if the repeated click operations reach 96% within a certain period of time. Meanwhile, the click range of the operation is recorded to reduce the time for the next repeat execution and the number of clicks within the same range. The maximum coverage of all functions of the application, effectively improves the efficiency of dynamic detection.

The fourth key capability of the server is the ensemble learning classification and recognition module. The ensemble learning classification stage consists of two parts. The role of the training part is to generate the decision tree set and the corresponding weight value. Multiple cross-sampling was conducted according to the feature vector set of sample data to form multiple training subsets, and the decision tree set was generated according to the data of training subsets. The detection module is to classify and identify the test samples. In order to effectively define the classification effect, it defines a group of detection effect indicators.

- True positive (TP): The application is malicious software. It was detected as malware.
- False positive (FP): The application is not malicious

software. It was identified by faulty detection as malware.

- True negative (TN): The application is actually benign software. It was detected as benign software.
- False negative (FN): The application is actually malware. It was incorrectly detected as not malware.
- True positive rate (TPR): The proportion of the amount of malware correctly detected to the actual malware.

$$TPR = \frac{TP}{TP + FN}. \quad \textbf{(5)}$$

- Precision (PR): The proportion of correctly identified malware in all identified malware.

$$PR = \frac{TP}{TP + FP}. \quad \textbf{(6)}$$

- Accuracy (Acc): The proportion of correct identification in the total number of tested samples.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}. \quad \textbf{(7)}$$

- F-score: A measure of test accuracy that takes into account both test accuracy and recall rates.

$$F\text{-}score = \frac{2 * TPR * PR}{TPR + PR}. \quad \textbf{(8)}$$

Among them, the true positive rate, accuracy rate and F score are three very important indicators used to evaluate the reliability of the system. The higher the three values, the better the classification effect.

6 Performance Evaluation

6.1 Experimental Apparatus

The client of the DroidExaminer system is a mi10 mobile phone. A server is a computer. The computer's CPU is an eight-core AMD processor, with 16GB of RAM and 512GB of the hard drive. It is installed with Linux kernel 5.10. The Android runtime container uses the anbox tool. The Android operating system uses Android7.0, which is packaged as an android.img file with modified API source code, and then loaded into the anbox emulator to run. Software development includes Android applications, server-side programs, automated execution scripts and ensemble learning simulation programs. During the simulation test phase, a server can detect six applications and generate log information after each execution.

Sample data sets are from common malware storage sites. Samples of the malware used were collected between July 2021 and June 2022, while benign apps were downloaded from Google's official Store and Xiaomi's App Store at the same time. A total of 6240 applications were collected and the number of malicious samples trained was 2,030. The number of other programs used for detection was 4,210, of which 839 were malware and 3,371 were benign. With far fewer malware applications than benign ones, this sample set is reasonably proportional.

The application running on the phone contains a monitoring Service and an Activity interface. The server application container environment is based on the anbox tool for Linux, which is a lightweight Android running container. The code is open-source and can be modified for extensibility. Ensemble learning relies on the open-source machine learning algorithm library Scikit learn, which provides implementations of a variety of machine learning algorithms, including the random forest algorithm we will be using. Scikit learn supports feature processing and process design methods, and many existing machine learning libraries are implemented with Scikit learn compatible interfaces.

6.2 Evaluation

We evaluate the performance and accuracy of the detection system. DroidExaminer improves the dynamic detection performance and achieves a high detection accuracy. Due to the modular design idea of the system, the scalability of the detection system is better. In the experiment, it is estimated that about 70% of the time is spent in the feature information acquisition stage, 20% of the time is used for classification recognition, and 10% of the time is consumed in the communication transmission between modules.

The double detection capability of the system, in order to quickly respond to users at the first time, the time of the first detection is controlled in a short time range. The second detection time is strongly related to the size of the installation package and the dynamic detection performance. Automated script execution through Droidtest to improve detection performance. Compared with the direct use of commonly used test tools monkey and MonkeyRunner, we evaluated three aspects: CPU consumption, memory consumption and functional coverage of the same running time. As shown in Table 5, DroidTest achieves a wide range of application function testing on the basis of less CPU and memory consumption. Droidtest's memory usage is only 0.2% more than Monkey, much less than MonkeyRunner. This is due to the addition of the repeated coverage calculation method, in order to reduce the detection time and achieve wider functional coverage.

Table 5. Automated test tool comparison

Tool	CPU	Memory	Scope
DroidTest	5.8%	4.7%	89.3%
Monkey	7%	4.5%	76%
MonkeyRunner	11%	7.3%	84%

The recognition of the classification module is calculated according to the three indicators of true positive rate, accuracy rate and F score mentioned above. We take integer statistics on the test sample data of more than 4000. The sample data is divided into a stage every 500 for training tests. Figure 5 shows the changes in the three measures during the testing of the data set. The sample data reached more than 2000, and the recognition accuracy reached 96%, indicating that our system model design was reasonable. It has high accuracy in recognition. Our detection system is in the experimental stage. In the future, we can introduce more efficient ensemble learning and recognition model to further improve the accuracy.

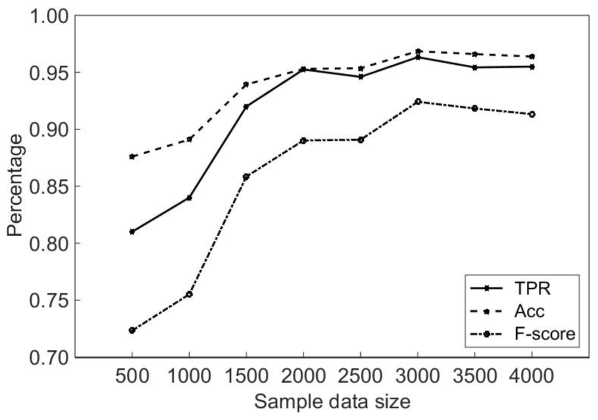

Figure 5. True positive rate, Accuracy rate and F-score

DroidExaminer takes distributed deployment and scalability into consideration in system design, and can update the training sample data set of the classification model according to the future evolution of Android malware to achieve better detection results. In our experiment, the random forest algorithm of ensemble learning is adopted, and the weight of the decision tree is considered to distinguish the difference between a strong classifier and a weak classifier. On the other hand, in the selection of feature attributes, more relevant information is selected to enhance the differentiation of Android applications. Multiple cross-experiments show that the classification model has better detection and classification accuracy. Our sample set is not complete enough for a large number of malicious software. Due to the optimization of automatic script execution performance, the dynamic detection efficiency of the system is improved and the consumption of hardware resources is reduced. The DroidExaminer system achieves the balance between resource consumption and detection efficiency, which improves the application value of the detection system.

7 Conclusion

The malware of Android smartphones is constantly changing, and various detection schemes have different detection scopes for the malware. DroidExaminer is divided into the first detection scheme with fast response and the second dynamic detection scheme with high accuracy. Obtain representative feature sets and filter the data that is strongly associated with malware. Classification recognition by ensemble learning can improve accuracy and reduce false negative rates. DroidExaminer's modular design is conducive to the expansion and update of the detection system. The system has a high degree of automation, taking into account the constant change and development of Android malware. We tested the data set and achieved a desirable accuracy. DTS service and ECS service are standardized designs, and interface invocations in different scenarios. Dynamic detection can check the situation of code reflection and dynamic code loading, and future research will cover a broader range of API calls.

The data characteristic analysis and detection results of this experiment achieve the expected effect. According to the analysis of the experimental process, we can still make some improvements. The data selection range of the sample is not large enough, so the sample size can be expanded. We use the parallel algorithm, which can be extended to the serial algorithm of ensemble learning. Comparative testing on the weight difference and priority of feature attributes can get a better detection effect on large-capacity data. The performance of dynamic detection can be further improved. DroidExaminer supports the scalability of distributed services. Other ensemble learning and malware detection algorithms can be introduced in future research.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grants No. 62102169, No. 61922045, No. U21A20465, No. 62172292, and Science Foundation of Zhejiang Sci-Tech University (ZSTU) under Grants No. 22222266-Y.

References

[1] Statcounter GlobalStats, *Mobile operating system market share worldwide. 2020*, Recuperado de: https://gs. statcounter.com/os-market-share/mobile/worldwide, December, 2021.

[2] N. T. Cam, N. H. Khoa, T. T. An, N. P. Bach, V.-H. Pham, Detect repackaged android applications by using representative graphs, *2021 8th NAFOSTED Conference on Information and Computer Science*, Hanoi, Vietnam, 2021, pp. 102-106.

[3] G. He, L. Zhang, B. Xu, H. Zhu, Detecting repackaged android malware based on mobile edge computing, *2018 Sixth International Conference on Advanced Cloud and Big Data*, Lanzhou, China, 2018, pp. 360-365.

[4] K. Tian, D. Yao, B. G. Ryder, G. Tan, G. Peng, Detection of repackaged android malware with code-heterogeneity features, *IEEE Transactions on Dependable and Secure Computing*, Vol. 17, No. 1, pp. 64-77, January-February, 2020.

[5] A. Arora, S. K. Peddoju, M. Conti, Permpair: Android malware detection using permission pairs, *IEEE Transactions on Information Forensics and Security*, Vol. 15, pp. 1968-1982, October, 2019.

[6] T. Kim, B. Kang, M. Rho, S. Sezer, E. G. Im, A multimodal deep learning method for android malware detection using various features, *IEEE Transactions on Information Forensics and Security*, Vol. 14, No. 3, pp. 773-788, March, 2019.

[7] J. Zhang, C. Tian, Z. Duan, L. Zhao, Rtpdroid:

Detecting implicitly malicious behaviors under runtime permission model, *IEEE Transactions on Reliability*, Vol. 70, No. 3, pp. 1295-1308, September, 2021.

[8] R. Surendran, T. Thomas, S. Emmanuel, A tan based hybrid model for android malware detection, *Journal of Information Security and Applications*, Vol. 54, Article No. 102483, October, 2020.

[9] M. Lindorfer, M. Neugschwandtner, C. Platzer, Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis, *2015 IEEE 39th annual computer software and applications conferenc*e, Vol. 2, Taichung, Taiwan, 2015, pp. 422-433.

[10] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, Z. Jia, A mobile malware detection method using behavior features in network traffic, *Journal of Network and Computer Applications*, Vol. 133, pp. 15-25, May, 2019.

[11] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, L. Ziarek, Android malware detection using complex-flows, *IEEE Transactions on Mobile Computing*, Vol. 18, No. 6, pp. 1231-1245, June, 2019.

[12] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, A. Narayanan, Semantic modeling of android malware for effective malware comprehension, detection, and classification, *Proceedings of the 25th International Symposium on Software Testing and Analysis*, Saarbrücken, Germany, 2016, pp. 306-317.

[13] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Drebin: Effective and explainable detection of android malware in your pocket, *21st Annual Network and Distributed System Security Symposium, NDSS 2014*, San Diego, CA, USA, 2014, pp. 23-26.

[14] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, S. Anwar, Static malware detection and attribution in android byte-code through an end-to-end deep system, *Future generation computer systems*, Vol. 102, pp. 112-126, January, 2020.

[15] V. Rastogi, Y. Chen, X. Jiang, Droidchameleon: evaluating android anti-malware against transformation attacks, *Proceedings of the 8th ACM SIGSAC symposium on information, computer and communications security*, Hangzhou China, 2013, pp. 329-334.

[16] B. Kondracki, B. A. Azad, N. Miramirkhani, N. Nikiforakis, The droid is in the details: Environment-aware evasion of android sandboxes, *Proc. Network and Distributed Systems Security Symposium (NDSS)*, *NDSS 2022*, San Diego, California, USA, 2022, pp. 1-16.

[17] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, J. Hoffmann, Mobile-sandbox: combining static and dynamic analysis with machine-learning techniques, *International Journal of Information Security*, Vol. 14, No. 2, pp. 141-153, April, 2015.

[18] K. Tam, S. Khan, A. Fattori, L. Cavallaro, Copperdroid: Automatic reconstruction of android malware behaviors, *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*, San Diego, California, USA, 2015, pp. 1-15.

[19] H. S. Galal, Y. B. Mahdy, M. A. Atiea, Behavior-based features model for malware detection, *Journal of Computer Virology and Hacking Techniques*, Vol. 12, No. 2, pp. 59-67, May, 2016.

[20] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, L. Cavallaro, Droidscribe: Classifying android malware based on runtime behavior, *2016 IEEE Security and Privacy Workshops (SPW)*, San Jose, CA, USA, 2016, pp. 252-261.

[21] X. Wang, K. Sun, Y. Wang, J. Jing, Deepdroid: Dynamically enforcing enterprise policy on android devices, *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*, San Diego, California, USA, 2015, pp. 1-15.

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

[23] M. Sun, T. Wei, J. C. Lui, Taintart: A practical multi-level information-flow tracking system for android runtime, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016, pp. 331-342.

[24] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, 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, June, 2014.

[25] Y. Zhao, C. Xu, B. Bo, Y. Feng, Maldeep: A deep learning classification framework against malware variants based on texture visualization, *Security and Communication Networks*, Vol. 2019, pp. 1-11, April, 2019.

[26] D. Liu, Z. Li, C. Wang, Y. Ren, Enabling Secure Mutual Authentication and Storage Checking in Cloud-Assisted IoT, *Mathematical Biosciences and Engineering*, Vol. 19, No. 11, pp. 11034-11046, August, 2022.

[27] D. Liu, Z. Li, D. Jia, Secure Distributed Data Integrity Auditing with High Efficiency in 5G-enabled Software-Defined Edge Computing, *Cyber Security and Applications*, Vol. 1, Article No. 100004, December, 2023.

[28] A. Saracino, D. Sgandurra, G. Dini, F. Martinelli, Madam: Effective and efficient behavior-based android malware detection and prevention, *IEEE Transactions on Dependable and Secure Computing*, Vol. 15, No. 1, pp. 83-97, January-February, 2018.

[29] H. Cai, N. Meng, B. G. Ryder, D. D. Yao, *Droidcat: Unified dynamic detection of android malware*, Department of Computer Science, Virginia Polytechnic Institute & State, Tech. Rep. TR-17-01, 2016.

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

[31] S. Y. Yerima, S. Sezer, DroidFusion: a novel multilevel classifier fusion approach for android malware detection, *IEEE Transactions on Cybernetics*, Vol. 49, No. 2, pp. 453-466, February, 2019.

[32] Z. Ma, H. Ge, Y. Liu, M. Zhao, J. Ma, A combination

method for android malware detection based on control flow graphs and machine learning algorithms, *IEEE Access*, Vol. 7, pp. 21235-21245, January, 2019.

[33] Z. Yuan, Y. Lu, Y. Xue, Droiddetector: android malware characterization and detection using deep learning, *Tsinghua Science and Technology*, Vol. 21, No. 1, pp. 114-123, February, 2016.

[34] P. Feng, J. Ma, C. Sun, X. Xu, Y. Ma, A novel dynamic android malware detection system with ensemble learning, *IEEE Access*, Vol. 6, pp. 30996-31011, June, 2018.

Biographies

Zhongxiang Zhan is currently working toward the M.E. degree at the Nanjing University of Information Science and Technology (NUIST), Nanjing, China. His current research interests include information security, software security and malware detection.

Sai Ji received his Ph.D. degree from the Nanjing Aeronautics and Astronautics University (NUAA), Nanjing, China, in 2014. He is currently a Professor at Taizhou University. His current research interests are in the areas of computer measurement and control, structural health monitoring, applied cryptography and WSNs. He has published more than 60 journal/conference papers. He is a Principle Investigator of three NSF projects. He is a member of ACM, CCF and IEEE.

Wenying Zheng received the M.E. degree in Electronic Engineering from Chosun University, Gwangju, Korea, in 2009, and the Ph.D. degree in Computer Science from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2022, respectively. She is currently working with the School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China. Her research interests include cloud storage security, security systems and network security.

Dengzhi Liu received the M.E. degree and Ph.D. degree from the Nanjing University of Information Science and Technology, in 2017 and 2020, respectively. He is currently an Associate Professor with the School of Computer Engineering, Jiangsu Ocean University, China. He mainly focuses on the security and privacy issues in data storage and transmission. He has authored more than 50 research papers and published in international conferences and journals. His current research interests include cloud computing security, cyber security, and data security.