A Privacy Leak Detection Mechanism based on Service Binding

Boyang Wang[1], Jinling He[2], Yuanhan Du[2], Ming Tang[2], and Xiaolong Xu[3,*]
[1]Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing, China
[2]State Grid Jiangsu Electric Power Company Limited, Nanjing, China
[3]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China
[2022040504, xuxl]@njupt.edu.cn; jinling.he@foxmail.com; duyuanhan@126.com; tangming930702@163.com
*corresponding author

Abstract—With the development of mobile technology, users gradually store financial information and personal privacy data on mobile terminals, which makes mobile security particularly important. In order to effectively improve the detection speed of application privacy leaks, this paper proposes a concept of service binding, binding a minimum privacy permission set for each service provided by the application. Use the service binding concept to make the static analysis phase track only the sensitive data flow paths between APIs that apply the service's privacy permissions. In order to make up for the lack of static analysis to predict the behavior of application privacy leakage, this paper proposes a static analysis that effectively combines machine learning algorithms, abstracts the path characteristics of the static data streams obtained from static analysis into feature vectors, and further uses machine learning algorithms for learning classification. The experimental demonstration and performance analysis results show that the privacy leak detection mechanism based on service binding has faster detection speed and higher accuracy than FlowDroid.

Keywords-privacy leakage, static analysis, service binding, machine learning

1. INTRODUCTION

In the past 10 years, the expansion of mobile devices in the consumer sector and even in the corporate sector has been surprisingly rapid and quantitative. As early as 2014, ANDREESSEN HOROWITZ analyst Benedict Evans said: "Mobile is Eating the World." This has never been an exaggeration. The number of mobile devices in the world has long surpassed the total population of the world many years ago. Since Apple introduced the first iPhone in 2007 to revolutionize smartphones for only 16 years, the market size of the mobile industry has grown rapidly. According to statistics from Statista, the global smartphone shipment in 2010 was around 0.34 billion units, however, this figure climbed to 1.28 billion by the end of 2020, and it is expected to reach 1.43 billion by 2023.

Today, the main mobile operating systems are Android and iOS. Android's source code is released by Google under an open-source license, which allows developers to develop secondary according to their own needs. IOS system has the advantages of smooth operation experience, exquisite system interface and high security which are incomparable to other mobile phone systems. Although these two mobile phone operating systems both provide users with a lot of applications, the difference between them is that most applications of Android are free while that of iOS takes charge but has higher security. According to the latest data from Statista, the number of applications in the Google Play App Store reached 3.482 million, and the number of applications in Apple's App Store was 2.226 million. The cumulative number of App downloads in the App Store reached 218 billion times. It also shows that mobile users usually spend 87% of their time on mobile apps.

From the perspective of mobile security, mobile platforms are more and more likely to be attacked, and the risk is also increasing. In addition, Android and IOS have adopted a coarse-grained rights management mechanism, that is, the user is responsible for the mechanism for applying and granting application-level permissions. Due to the lack of awareness and professional knowledge about mobile platform privacy leakage, the mobile platform has become a high-risk area for privacy leakage. Therefore, mobile applications need an effective privacy leak detection mechanism.

The existing mobile terminal privacy leak detection technologies for application software mainly include LeakMiner [1], TrustDroid [2], FlowDroid [3], TaitDroid [4], AppFence [5], and Kynoid [6]. They avoid privacy leakage of mobile platforms to a certain extent for specific application scenarios. Unfortunately, these schemes have found the following problems.

(1) Existing solutions generally detect privacy permissions applied during application installation, but services used by actual users only involve partial privacy rights. This takes a number of unrelated privacy rights into consideration, which not only increases the time spent in the evaluation phase but also reduces the accuracy of the test results.

(2) Static analysis only analyzes whether there is a sensitive data flow path in the application, while some existing service applications require users' private information to provide normal services, such as Baidu Map, Meituan Takeaway, Taobao, etc. These applications also have sensitive data flow paths. Sensitive data streams can usually reflect a user's own characteristics and habits, and profiling can be performed through sensitive data streams, but it is difficult to effectively distinguish whether an application is legitimately using user privacy or leaking user privacy by analyzing sensitive data streams alone. The dynamic analysis method needs to

2835-2823/24/$31.00 ©2024 IEEE
DOI 10.1109/ISSSR61934.2024.00018

execute the program and extract the features related to malicious code by collecting the runtime information of the program. However, this method can only obtain one execution track at a time and cannot ensure that all sensitive behaviors are included, so it has a high misjudgment rate.

In order to solve the above problems, we propose the concept of service binding, use the FlowDroid analysis tool to detect privacy leaks in the application software, and use the random forest algorithm for data analysis of the static analysis results. Our main contributions are as follows:

(1) We propose a concept of service binding, which binds a minimum privacy permission set for each service provided by the application. The minimum privacy permission set refers to the privacy permission that the service needs to apply for. This algorithm obtains the set of privacy permissions that need to be managed according to the services that users need to use, which not only reduces the number of privacy permissions detections, but also improves the detection speed. At the same time, the misjudgment rate of privacy leakage behaviors caused by the detection of considering redundant privacy permissions is reduced.

(2) For static analysis, only the sensitive data flow path in the application can be detected. It is difficult to effectively determine whether the application has a privacy leak behavior. This paper proposes a static analysis that effectively combines with machine learning algorithms, abstracts the path features of the sensitive data streams into feature vectors, and further uses machine learning algorithms to learn to classify and improve the accuracy of the detection results.

(3) Based on the concept of service binding, we select FlowDroid as a APK static analysis tool, and Random Forest Algorithm as a classification algorithm to implement a prototype system for privacy leak detection on the Android platform.

2. Related Work

Current privacy leak detection technologies for mobile terminals can be mainly classified into two types: static analysis and dynamic analysis.

Static analysis techniques directly avoid the problem of incomplete path coverage caused by dynamic analysis by statically scanning the application rather than executing the application dynamically. Static analysis can be divided into two types: 1. Feature matching of program code. 2. Static flow analysis. At present, there are the following studies at home and abroad:

Arzt [3] proposed FlowDroid to perform accurate static pollution analysis on Android applications. It takes the Android apk file as input for processing and performs static pollution analysis, handles callbacks by simulating the complete Android life cycle. It is context-sensitive, but also stream, field, and object-sensitive. Using the SuSi framework [7], Source and Sink of the target Android version can be determined. Arzt et al. detect Source, Sink, and EntryPoint by parsing the manifest and dalvik executables (dex). Then they perform a pollution analysis to find the path from Source to Sink and report all discovered paths.

Yue [8] proposed a strategy for replacing reflective call statements with non-reflective call statements and implemented a DyLoadDroid tool that can effectively analyze the dynamic loading and reflection mechanism of Android. At the same time, experiments have verified its effectiveness in dealing with the taint analysis of Android dynamic loading and reflection mechanisms.

Zhao [9] proposed a new privacy scanning technique to solve the problem of matching privacy information with behaviors in privacy policy texts and codes. The method trains a BiLSTM (Bidirectional Long Short Term Memory) model based on the attention mechanism, by which we can determine whether the text contains private information or not. Then, the statements where the privacy information is located are analyzed and the negative and behavioral words in the statements are extracted and recorded.

Wu [10] proposed TraceDroid for detecting Android malware, TraceDroid uses a static analysis approach to derive the possible execution trajectories of an application and identify sensitive information, and also utilizes a dynamic analysis approach to gather runtime information to discover disguised malware. This hybrid approach achieves better performance than purely static or dynamic analysis methods. The ability to obtain more network traffic data and provide better characterization of network behavior using hybrid analysis methods. Finally, the transmission data will be used for model training.

Sentana [11] proposed a static analysis method to analyze the permissions, as well as tracking the third-party libraries used by each app through decompilation and analyzing the impact of third-party libraries on privacy. Subsequently, malware was analyzed and some malware was captured. In addition, the authors performed network measurements to investigate the behavior of the network while the app was running. Finally, the authors conducted an app compliance and user reviews analysis to analyze privacy policy compliance and user reviews of the apps, and used reviews as one of the important parameters of security issues and the likelihood of user privacy breaches in cryptocurrency wallet apps.

Dynamic analysis refers to analyzing the function of a function, clarifying the logic of a code, and excavating possible privacy leaks by observing the state of a program during its execution, such as register contents, function execution results, memory usage, and so on. The most famous of these is the TaintDroid framework [4] for the Android platform. The modified Android virtual machine taints and tracks sensitive data. It can track multiple Sources and Sinks simultaneously. When sensitive information leaves the system, the running mobile phone user will be informed. The detection system based on this framework has AppFence [5]. Cui [12] proposed a network traffic analysis framework for detecting privacy leaks in Android

Apps, which utilizes dynamic hooking techniques to add additional code to functions responsible for executing HTTP(S) requests, and then saves unencrypted data and corresponding code execution traces. In order to study privacy leakage between host apps and third-party libraries with fine-grained accountability, the authors also propose an unsupervised approach that identifies third-party libraries by correlating requests and code execution traces, and uses this approach to differentiate traffic between host apps and third-party libraries. Schindler [13] proposes a method for identifying privacy leaks in third-party Android libraries that combines dynamic and static analysis methods, using the FlowDroid static analysis method and the Frida dynamic analysis method, and finally adding mitm-proxy to detect suspicious data sinks. Hao [14] presents a new clustering-based approach to detect third-party libraries in Android applications and perform privacy leakage analysis on third-party libraries. The approach uses fuzzy matching to cluster feature vectors into different clusters and uses cluster prediction to detect third party libraries and also uses dynamic instrumentation to monitor sensitive APIs and perform privacy breach analysis.

3. Privacy Leak Detection Mechanism Based on Service BindinG

3.1. Related definitions

Definition 1: Application. Apps App installed on phones and tablets.

Definition 2: Benign application. There is no privacy disclosure in the program.

Definition 3: Malignant applications. There is a privacy breach in the program.

Definition 4: Application Services. Some functional services are provided by the application, such as shopping, video, reading, etc.

Definition 5: Privacy Rights. Permissions involving mobile terminal user's privacy data. It can be divided into two categories. One is the Source privacy right. Some APIs it manages can read the local privacy data of the mobile phone. The other is Sink privacy, which manages some APIs that can transfer data to external systems. For example: Internet, SMS, etc.

Definition 6: Set of minimum privacy permissions. The application service requires normal application privacy permissions.

Definition 7: Service Binding. The application service corresponds to a set of minimum privacy rights, and the security of the application service determines the security of the set of corresponding minimum privacy rights.

Definition 8: Sensitive data flow path. Refers to the path of sensitive data between APIs from Source permissions to APIs related to Sink permissions, in the form of $API_{Source} \rightarrow API_{Sink}$.

Definition 9: The characteristic abstraction of all sensitive data flow paths of an application service can be defined as $S=\{r_1,r_2,r_3,...,r_n,c\}$, where r_i represents a sensitive data flow path and subscript n represents the total number of sensitive data flow paths. c indicates the nature of the application, such as benign or malignant.

3.2. System Model

Since dynamic analysis requires the execution of a program, this will bring a certain load to the mobile terminal. In addition, dynamic analysis does not ensure coverage of the data flow path and has a high misjudgment. Therefore, dynamic analysis is rarely used alone. Compared to dynamic analysis, static analysis does not require the installation and execution of applications. It only requires static scanning of the application execution code. At the same time, static analysis can effectively solve the problem of incomplete data path coverage. Therefore, most academic studies use static analysis to detect whether an application has a privacy breach. For the static analysis, it is impossible to judge whether there is a privacy leakage problem according to the path of the sensitive data flow. In addition, when the existing solution detects the application privacy leak, it generally detects the privacy permission applied during application installation, but the actual application service used by the user only involves some privacy rights. Based on this, we propose an application service privacy leak detection framework. This solution combines the machine learning algorithm with the sensitive data flow path of the detection application for further analysis, and only detects the privacy permissions of the application service binding, which can improve the speed of application privacy detection.

As shown in Figure 1, there is a combination of static analysis and detection record acquisition in the horizontal direction. The static analysis is responsible for privacy leak detection of the application, and the detection record acquisition is to avoid duplicate detection of the same application service. Therefore, in this solution, a unique identifier (signature_package name_version number) is created for each application, and the detection result of each application service is timely uploaded to the database server. When the next scheme accepts a detection task, it first determines whether the application service of the application to be detected has a detection result in the database. If a detection result already exists in the database, the detection result is obtained and the final detection result of the privacy permission preset rule is given in combination with the permission setting. If the database does not exist, privacy leak detection is performed on the execution code of the application service according to the normal flow. The structure of the application service detection record table stored in the database is shown in Table 1. The detection identification attribute is whether the application has been detected, 1 for yes, and 0 for no. Secure application services refer to application services that do not leak privacy.

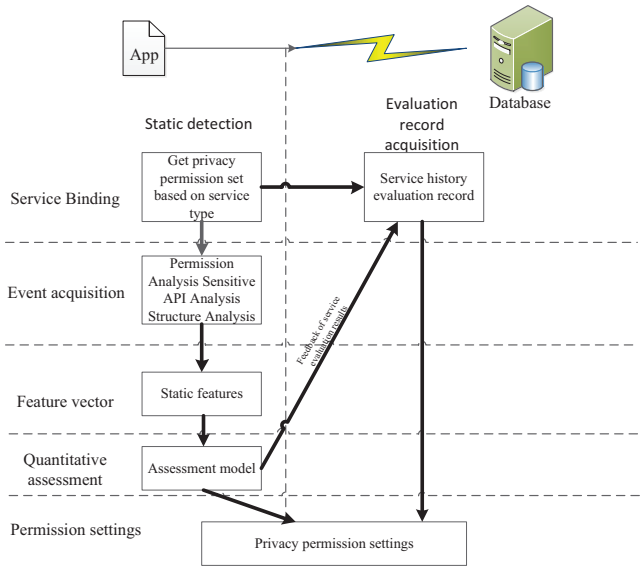

Figure 1. The overall framework of the program.

In the vertical direction, privacy leak detection is divided into multiple layers such as service binding, event acquisition, feature vector, quantization detection, and permission setting, which not only guarantees the detection mechanism performance, but also satisfies the personalized settings of different users' privacy rights.

The service binding refers to the minimum privacy permission set that needs to be applied for the service type binding in advance and determines whether the privacy permission set bound by the application service is secure by detecting the application service. A service is a function provided by an application, for example: video, shopping, music and reading. The minimum privacy permission set is the privacy permission that needs to be applied for the normal operation of the storage service. Privacy rights refer to some access control policies that manage user-sensitive data. There are generally read address books, read user accounts, and read calendars.

Event acquisition refers to obtaining the application service's sensitive data flow path through static analysis, mainly through the smut analysis to track the flow of sensitive data in the application and record the path of sensitive data flow between functions. Static analysis is to statically scan the application instead of dynamically executing the application. Therefore, the path coverage insufficiency caused by dynamic analysis is directly avoided. Therefore, we can statically analyze the application offline to identify the application potential in privacy leaked in advance. So, we choose to use static analysis to detect the application's privacy leak behavior.

The eigenvectors refer to the quantification of the sensitive data flow path into feature vectors. Because it is impossible to determine whether there is a privacy leak behavior based on whether there is a sensitive data flow path. Therefore, we quantify the path of the sensitive flow from the static analysis and continue to use the machine learning algorithm for quantitative detection. We consider the application privacy leak detection as a classification problem and count the sensitive data flow paths of all malicious applications and benign applications. Then we analyze the characteristics of benign applications and malicious applications with different sensitive data flow paths, select applications with statistical differences, and construct a feature sample matrix through the characteristics. The quantification definition of the feature sample matrix is shown in Table 2.

Table 1. Application Service Test Record

Application ID	Detection ID	Security Application Service
Zelyy_com.zelyy.finance_1.6.0	1	[Financial management]
Bolton_com.fanli.Android.apps_6.5.0	1	[shopping]
…	…	…

Table 2. Feature sample matrix

feature / app name	APISOURCR-->APISINK	…	Application properties
app1	1	…	GOOD
app2	1	…	BAD
app3	0	…	GOOD
…	…	…	…

As shown in Table 2, the columns of the feature matrix contain sensitive data flow paths and application properties owned by malicious applications and benign applications to mark whether the application is benign or malignant. If the detected application has a sensitive data flow path listed, it is set to 1, otherwise, it is set to 0. If the application has a privacy breach, then its application is BAD. Otherwise, it is GOOD. The permission setting is to meet the different privacy requirements of the user. This solution provides the user with the opening rules for autonomously presetting 13 privacy rights. When the solution completes the detection of the leakage of the application service privacy, it will further obtain the opening rules of the permission setting and the privacy permission preset. Finally, it gives a scientific suggestion to the user to assign the privacy permission to use the application service binding.

3.3. Workflow

The process of testing the application by the solution is as follows:

(1) It is reasonable to obtain the type of service to be tested and the type of service to be verified. The type of application service detected by the solution is the application service that the user needs to use. The solution default user knows what services the application can provide. In order to improve the stability of the solution, after the user chooses to use the application service, we will verify whether this application has this application service. If it is owned, the solution will be executed normally. Otherwise, the plan prompts the user to continue choosing the type of service to use. The method

of application service verification is to decompile the application permission application file and acquire the permission set of the application. If the application request permission set contains the minimum permission set corresponding to the application service selected by the user. Then, the user-selected application service is reasonable, and vice versa, it is unreasonable.

(2) Acquire the application's unique identifier (signature_package name_version number) and use the unique identifier and application service to query the history detection result from the database. If there is, then the opening rule of the detection result combined with the permission setting gives a scientific suggestion for the problem of giving the user privacy permission. The implementation of the program ends, if there is no detection result in the database, in execution (3).

(3) Get the minimum set of privacy permissions that need to be detected according to the application service. Use static analysis to get the path of sensitive data flow between privacy-priority APIs.

(4) The path of the sensitive data stream is quantized into feature vectors according to the feature vector definition.

(5) Construct a feature sample matrix based on benign sample application and malignant sample application.

(6) Using machine learning algorithms to learn and classify the applied feature vectors. The classification result refers to whether the application is benign or malignant. Malignancy refers to applications that have privacy breaches. On the contrary, it refers to application security.

(7) In combination with the detection result, the permission setting sets a preset opening rule for the privacy permission. Finally, it gives reasonable suggestions for the application of privacy rights for application services used by users.

4. System Implementation

4.1. System Modules

This system is mainly aimed at privacy leak detection of Android platform applications. System execution collects the minimum set of privacy permissions required for the normal operation of various types of application services. We refer to the Google Play, Application treasure, and SnapPea classification of the application. We use Android applications in accordance with the services provided by the classification. A total of 16 service types are listed in Table 3. We downloaded 20 of the most downloaded applications for each service type. The AndroidManifest.xml of all applications is decompiled by the APK tool to obtain the permission of the application, and the intersection of all application permission sets under a single service type is collected as the minimum privacy permission set and saved locally. Our comprehensive privacy data aggregates thirteen important privacy privileges for the importance of users. 13 privacy privileges include: reading address books, reading user accounts, reading calendars, reading obscure addresses,

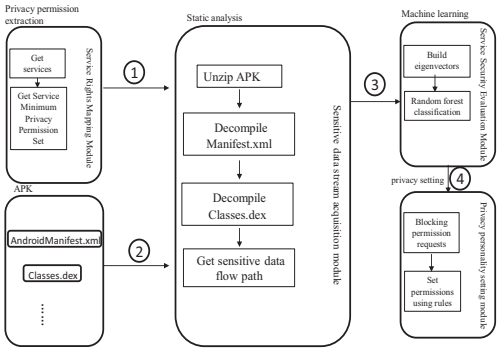

Figure 2. System implementation architecture

Table 3. 13 privacy rights

Authority	API
Reading address book	Android.permission.READ_CONTACTS
Reading user accounts	Android.permission.GET_ACCOUNTS
Reading the calendar	Android.permission.READ_CALENDAR
Read fuzzy address	Android.permission-group.LOCATION
Read the exact address	Android.permission.ACCESS_FINE_LOCATION
camera	Android.permission.CAMERA
recording	Android.permission.RECORD_AUDIO
Read phone status	Android.permission.READ_PHONE_STATE
Reading call records	Android.permission.READ_CALL_LOG
Reading SMS	Android.permission.READ_SMS
The internet	Android.permission.INTERNET
Read log	Android.permission.READ_LOGS
Read Browser History Access	com.Android.browser.permission.READ_HISTORY_BOOKMARKS

reading exact addresses, cameras, recording, reading phone status, reading call logs, reading logs, reading browser history visits, reading text messages, networking. The system implementation architecture is shown in Figure 2, and the binding relationship between application services and minimum privacy permissions is shown in Table 3. The system function module is introduced as follows:

(1) Service Rights Mapping Module

According to the input of the service type, when the user or the detected Android application applies for permission, the module obtains the minimum privacy permission set of the corresponding service from the local file according to the service type. The mapping relationship between the application service and the minimum privacy permission set is shown in Table 4. In order to ensure no duplicate

detection, the module will record the privacy leak detection result of each Android application service to the database server. Each Android application has its own unique identifier (signature_package name_version number), and we will uniquely identify the application in the database to find out whether the application service that has been detected by the
Android application has already recorded the detection result. If the historical detection result of the application service is found, the detection result read from the database can be directly submitted to the privacy personality setting module.

(2) Sensitive data stream acquisition module

This module is responsible for obtaining the minimum privacy permission set that the application has assigned to the normal operation of the privacy right and the to-be-detected service, obtaining the corresponding API function according to the privacy right, and then analyzing whether there is sensitive data flow path between these APIs through the taint. This module mainly quotes FlowDroid [15] for static analysis of applications. The FlowDroid open source project includes five subprojects: soot, heros, jasmin, soot-infoflow, soot-infoflow, and soot-infoflow-Android. Their dependencies are shown in Figure 3.3. FlowDroid stores an important SourcesAndSinks.txt file locally. This file is pre-defined by SUSI's Android system. Source and Sink APIs are prone to privacy leaks. There are 91 and 125 respectively. FlowDroid's sensitive data flow path analysis process is as follows:

a) FlowDroid collects source points, sink points, entrypoints, and callbacks before performing dataflow analysis.

b) FlowDroid will decompile and parse the APK's Manifest.xml file, generate the variable processMan, get its packagename and assign it to this.appPackageName="package name", get the entry point of its program. The entry point of the Android application is the four major components (Activity, Broadcast, Provider, Service) as defined.

c) The program calls initializeSoot(true) to decompile classes.dex to generate the stable file. Then, it calls the dummyMainMethod method body function to collect the callback function.

d) The program calls calculateSourcesSinksEntrypoints("SourceAndSink.txt") to collect the source and sink points.

e) The flow of FlowDroid has been improved here. We obtained the minimum set of privacy permissions for applications that have been granted privacy permissions and service requests that need to be detected. At the same time, the APIs recorded in SourceAndSink.txt are filtered to retain only the APIs related to the normal operation of the service to be detected.

f) After the preparation work is done, the program calls the runInfoflow() function to construct the data flow diagram (ICFG diagram), and analyzes whether there is a data flow path between each Source and Sink according to the ICFG diagram and saves the result in InfoflowResults.

Table 4. Service Privacy Rights Mapping Table

Application service	Minimum privacy set
navigation	{ Read precise location, network }
news	{ Read precise location, account, network, camera, recording, read phone status }
Financial management	{ Network, read phone status }
Social	{ Read precise location, network, camera, recording, read phone status }
video	{ Network, camera, read phone status }
shopping	{ Read precise location, network, camera, recording, read phone status }
communication	{ Network, camera, read phone status }
Office	{ Read address book, network, read phone status }
Safety	{ Network, camera, read phone status }
tool	{ Read precise location, network, camera, read phone status }
photography	{ Read precise location, network, camera, recording, read phone status }
life	{ Reading logs, network, camera, reading exact location, reading phone status }
system	{ network }
beautify	{ Reading logs, network, camera, read phone status }
music	{ Reading logs, network, camera, reading precise location, recording, reading phone status }
read	none

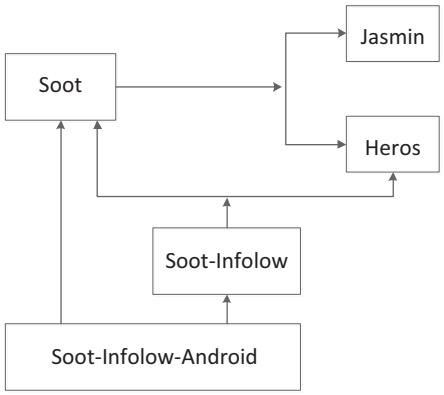

Figure 4. Dependencies between FlowDroid subprojects

(3) Service Security Detection Module

The sensitive data stream path obtained by the sensitive data stream acquisition module is constructed according to the abstract definition of the application service feature, and then the locally constructed feature sample matrix is called. This module refers to the random forest algorithm [16] to perform the feature vector of the application service, and then analyzes it to achieve application type classification.

The base classifier of the random forest is a decision tree. The bootstrap method is used to obtain the training set for constructing each decision tree from the training samples, the decision tree is constructed in a binary recursive manner, and the constructed decision tree is constructed into a forest. When it is necessary to classify the test samples, the samples

are passed to each tree in the forest. Each tree gives a classification result, and the most popular class among all trees in the forest is selected as the final decision.

Given a training data set D with M features, the process of the random forest algorithm can be described as follows:

a) Using the bootstrap method to generate K training subsets of the same size as the original sample set. The essence of the bootstrap method is a self-help method, a non-parametric statistical method: N times of random and repeatable sampling of the observed information (in this case, the original sample) to obtain the training for constructing each decision tree set. Bootstrap makes full use of given observation information and does not require models, other assumptions, or adds new observations. It has the characteristics of stability and high efficiency.

b) For each data set, a decision tree is constructed using a binary recursive approach. At each node of the decision tree, one feature subspace is randomly sampled from all features, and all possible splitting methods are calculated based on this feature. We use the best splitting method for nodes (for example, the maximum Gini metric). Then the nodes will continue to split until the tree reaches the preset stop condition. In this paper, the default stop condition is that the depth of the decision tree reaches the set value.

c) Then we combine unpruned trees, integrate a random forest, and use tree-to-tree voting as a sorting decision for random forests. When the sample to be classified is input, the classification result of the random forest output is determined by the simple majority vote of the output result of each decision tree.

(4) Privacy Personality Settings Module

Users can set up rules for privacy permissions in advance to meet the actual different privacy requirements of users. Users can turn on rule settings for a certain privacy permission. For example, set open conditions and open times. When the detection result of the service risk detection module is safe, the scheme will then check whether the privacy permission opening rule is set in the permission setting module. If the user sets the privacy permission in advance, then the rule is executed according to the setting rule. If no privacy permission setting rules are enabled, the solution will display the detection results on the system interface immediately, including whether the service is secure and which privacy rights can be assigned.

4.2. System Operation Process

System workflow chart shown in Figure 4, the specific steps are as follows:

Step 1: Decompile Manifest.xml to obtain Application Permission Set (APS).

Step 2: Get APK Unique ID (signature_package name_version number).

Step 3: Get the application service type from user feedback. Obtain the corresponding service security privacy permissions (SSPP) from the local file.

Step 4: Verify that the user feeds back the validity of the application service, and determine whether the application permission set includes the minimum privacy permission set bound by the application service. If included, go to step 5. Otherwise, the user is prompted to re-select the application service.

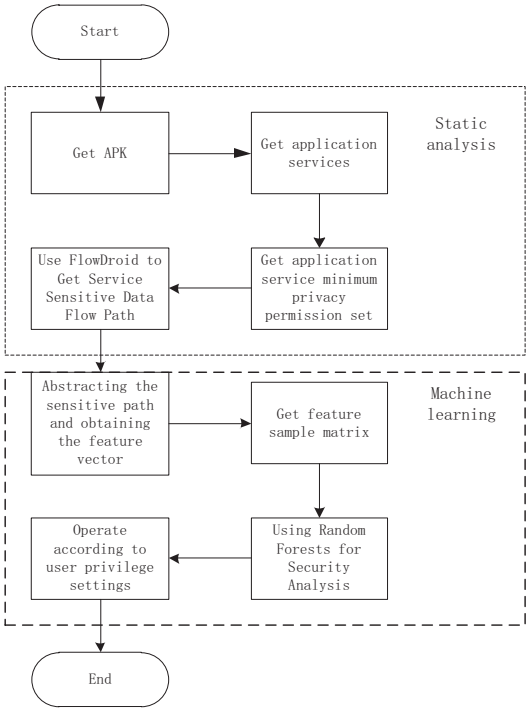

Figure 4. System work flow chart

Step 5: Obtain the security service set (SSS) and the evaluation mark (EM) of the application from the database according to the APK unique identifier. The SSS is used to save the application detection security service set. The EM is used to mark whether the application has been detected. If EM=1 and SSS are empty, indicating that the application has been detected and the security service is empty, then this application is not recommended for installation. If EM=1, SSS is not empty, it is judged whether the current service is in the collection. If yes, it is directly given to the privacy personality setting module, and the processing is performed according to the user's pre-set permission. If it does not exist, go to step 6. The application service test record table in the database is shown in Table 1.

Step 6: Remove the SSPP from the privacy permission granted by the application, and filter out the privacy permission that needs to be detected.

Step 7: Enter the sensitive data stream acquisition module, and use FlowDroid to trace the sensitive data flow path between the APIs related to privacy permissions.

Step 8: The path of the sensitive data flow obtained in the above step is converted into a feature vector by using the definition of the applied feature vector. This feature vector

can be used as a test set of random forest classification algorithms.

Step 9: Invoke the saved feature sample matrix file locally as a training set of the random forest classification algorithm. By analyzing the characteristics of benign applications and malicious applications that have differences in the path of sensitive data flows, those feature-construction feature sample matrices with statistical differences are selected.

Step 10: Use a random forest algorithm to analyze whether this application is secure, and at the same time decide whether the privacy of the application service binding is safe. If the result of this service test is safe, then this service is added to the SSS of this application in the database as a cache for the next test. If this service test result is unsafe, the feedback to the user that the user's selected application service is insecure and suggests rejecting the assignment of relevant privacy rights.

Step 11: The permission privacy right is combined with the preset opening rules of the 13 privacy rights by the privacy personality setting module and displayed on the system interface in time to provide the user with feedback on the security of the service and the privacy rights of the service application.

4.3. System Interface

A privacy violation detection system based on service binding includes:

(1) Application Service Evaluation

The main function is to perform privacy leak detection on application services. The execution process is:

a) Select the application to be detected APK

b) Select the application service to use

c) Click Start Evaluation. The evaluation result will show the detection results, including the security status of the application service security and the corresponding service privacy permission.

(2) Personality settings

The main function is to preset 13 privacy permission opening rules to satisfy the user's individual needs, including settings for privacy permission opening conditions and opening times. When the user sets the permission opening rule, you can click OK to complete the setting work.

5. Experimental Verification

5.1. Experimental Index

The system uses FlowDroid's excellent Android application static analysis open source project to construct the tested feature vectors based on the static analysis results of FlowDroid, and then uses the random forest classification algorithm to classify Android applications. This experiment mainly verifies the following two aspects of performance indicators:

(1) The accuracy of privacy leak detection results, recall rate, and F-measure.

The three evaluation indexes used in the experiment [20] were used to test the effect of this experiment. According to the actual situation of the application, the results of the experiment will result in four situations as follows.

a) True Positive, a benign application is judged to be a benign application. This type of application is denoted as $TP(M)$. M is the application feature vector entered during analysis.

b) True Negative, the malicious application is judged to be a malicious application, and this type of application was recorded as $TN(M)$.

c) False Positive, a malicious application is judged to be a benign application, and this type of application was recorded as $FP(M)$.

d) False Negative, a benign application is judged to be a malicious application, and the application was noted as $FN(M)$.

The formula for calculating the accuracy A is

$$A=\frac{|TP(M)|+|TN(M)|}{|TP(M)|+|TN(M)|+|FP(M)|+|FN(M)|}. \quad (1)$$

The formulas for measuring the accuracy P_B and recall rate R_B of benign applications are

$$P_B=\frac{|TP(M)|}{|TP(M)|+|FP(M)|}. \quad (2)$$

$$R_B=\frac{|TP(M)|}{|TP(M)|+|FN(M)|}. \quad (3)$$

The formulas for detecting the malicious application's accuracy P_M and recall rate R_M are

$$P_M=\frac{|TN(M)|}{|TN(M)|+|FN(M)|}. \quad (4)$$

$$R_M=\frac{|TN(M)|}{|TN(M)|+|FP(M)|}. \quad (5)$$

$|TP(M)|$, $|TN(M)|$, $|FP(M)|$, and $|FN(M)|$ respectively indicate the number of applications within the determination result.

The formulas for calculating the F-measure are

$$F_B=\frac{2P_BR_B}{P_B+R_B}. \quad (6)$$

$$F_M=\frac{2P_MR_M}{P_M+R_M}. \quad (7)$$

(2) Privacy leak detection time

We mainly examine the comparison between PDDMSB privacy leak detection mechanism and FlowDroid's mobile application privacy leak detection time consumption proposed in this chapter.

5.2. Data Set

We download a total of 5,000 malicious applications from VirusShare and Contagio's official website. Since many of the sample apps are just different versions of the same app,

Table 5. Dataset Application Source And Size Composition

size (MB)	Malignant applications			size (MB)	Malignant applications		
	Virus Share	Cont agio	total		VirusS hare	Cont agio	total
0~5	112	44	156	0~5	112	44	156
5~10	108	49	157	5~10	108	49	157

total	220	93	313	total	220	93	313

Table 6. Sample Application Collection

name	App properties	App service	Application introduction
mushroom Street	benign	Shopping	Mushroom Street, an e-commerce website focused on fashion women consumers, provides girls with products suitable for young women in the fields of clothing, shoes, bags, accessories, and beauty makeup.
Android Wallpaper	benign	beautify	Focus on millions of HD wallpapers for the Android platform.
Micro-lock screen	Malignant	beautify	A mobile phone lock screen APP.
Book artifact	Malignant	read	A newest, fastest, and most comprehensive novel guide reading assistant, covering the hottest series of novels on major websites, is a must-read app for reading.
...	…	…	…

there are many malicious apps whose APKs have been damaged, We filter and replace a total of 3000 malicious application APKs. And 456 benign application APKs are downloaded from Google Play and App. The FlowDroid project is a static analysis tool and his analysis ability is good enough. However, it can only be considered a laboratory product, FlowDroid is very sensitive to the size of the application of the analysis of the APK. When the APK file to be detected is only a few megabytes, FlowDroid can quickly analyze the sensitive data flow path. When the APK file to be detected is large, it needs to take up a lot of memory and take a long time. This is the reason why FlowDroid can't be used in the market. Therefore, due to the limited performance of the experimental machine, the experiment finally selected the APK file below 10M as the analysis object.

The data set for this experiment finally contained 769 Android applications that existed in the real world, of which 313 malicious applications came from the datasets VirusShare and Contagio. There are 456 benign applications from Google Play, App Store and other app stores. Table 5 shows the source and size of the entire data set. Table 6 shows the sample application set.

5.3. Experimental Results and Performance Analysis

Experiment 1: Comparing privacy leak detection results with PDDMSB and FlowDroid accuracy, recall rate, F-measure

Since FlowDroid has a long time to analyze static flow, we did not perform service inspection on all application APKs. We have chosen to verify the validity of the test results from different service type dimensions. We selected a total of 160 sample APKs from 769 sample APKs, of which 5 benign APKs and 5 malignant APKs were selected for each service type. A total of 16 service types are shown in Table 3. We have done 16 experiments in total according to the number of service types. Each set of experiments verifies the accuracy of current service test results, recall rate, and F-measure. A total of 10 APKs for each group were tested using the PDDMSB's prototype system. Record the results of each APK test by group $|TP(M)|$, $|TN(M)|$, $|FP(M)|$, and $|FN(M)|$. Finally, the accuracy rate, recall rate, and F-measure of each group were calculated. In addition, we used 16 sample APKs only for static analysis using FlowDroid, recorded the data and compared the detection accuracy of the two detection mechanisms: recall rate and F-measure. The comparison experiment results are shown in the Fig. 5~Fig.8. According to the above experimental results, the accuracy rate of the system detection results based on the service binding-based privacy detection mechanism is close to 94%, and the accuracy of the system detection results achieved using FlowDroid as the static analysis is close to 89%.

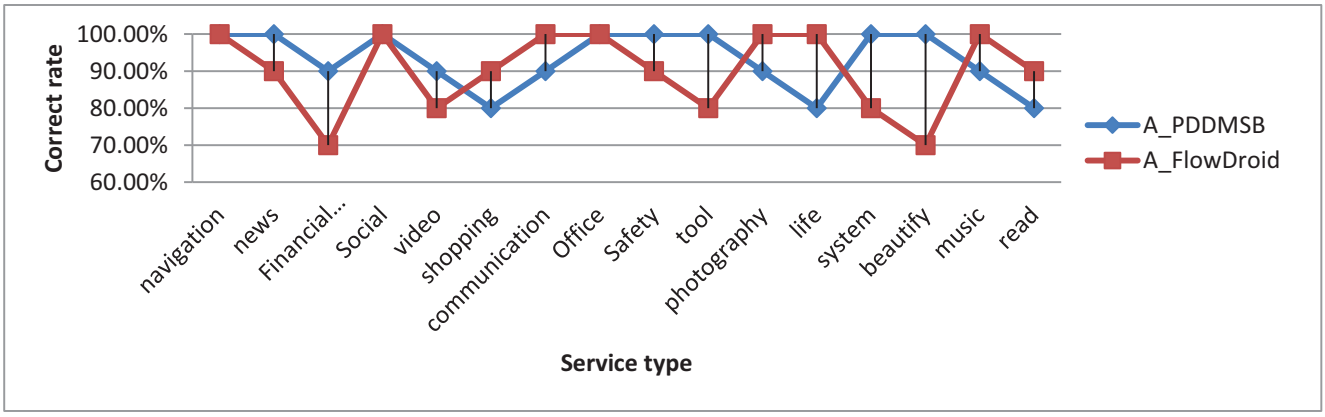

Figure 5. Plan detection accuracy

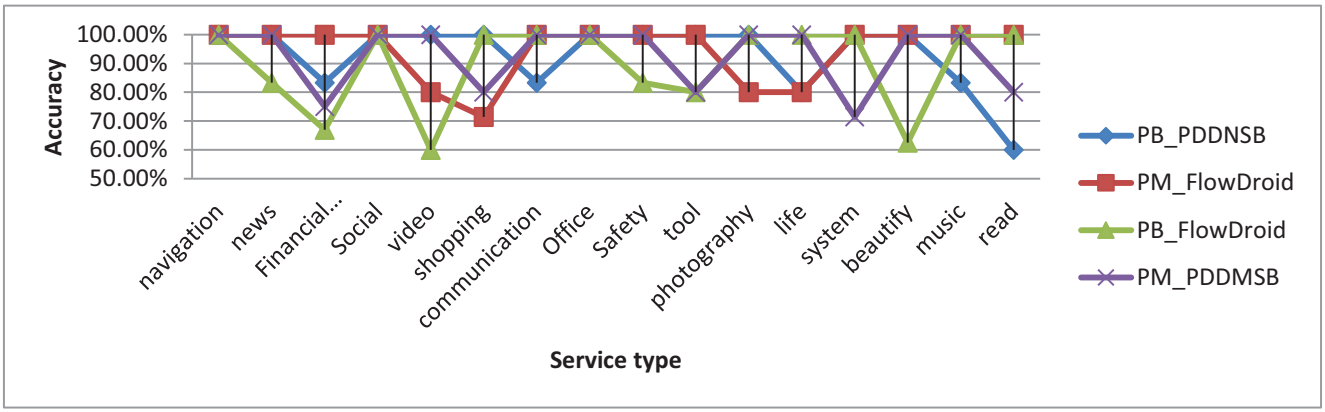

Figure 6. The program detects the accuracy of benign APKs and malicious APK

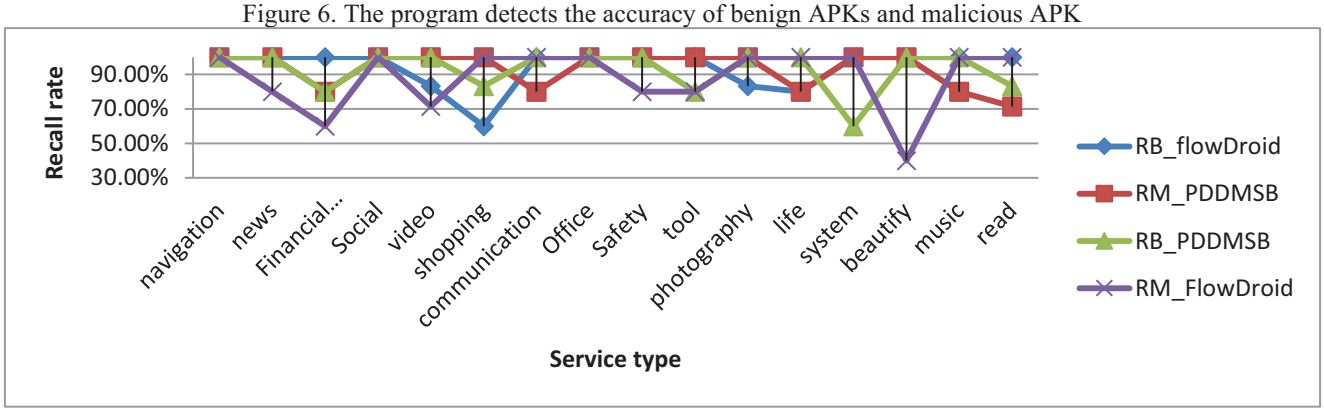

Figure 7. The program detects recalls of benign APKs and malicious APKs

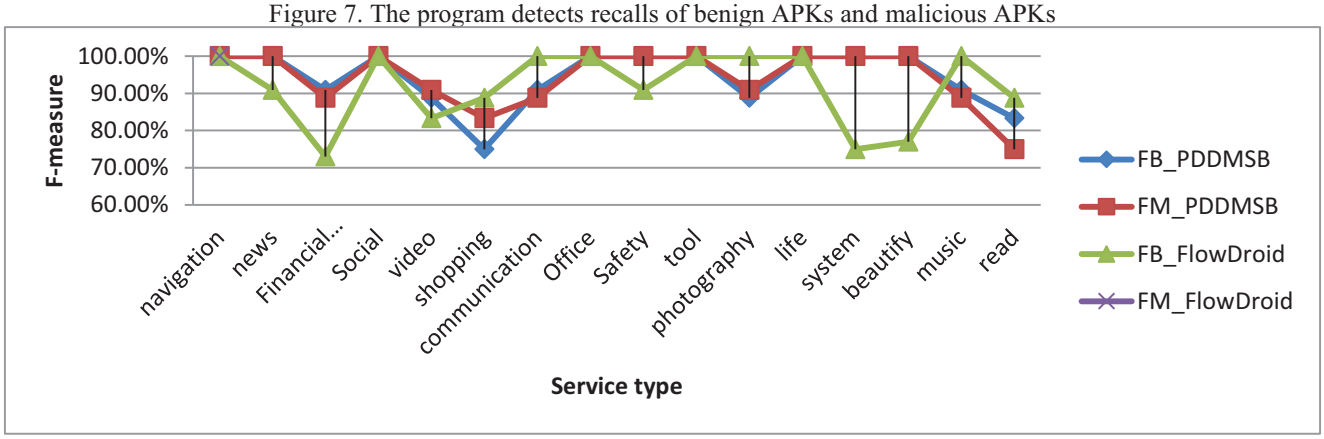

Figure 8. Protesting F-measure against benign APK vs. malicious APK

We can conclude that the solution-based privacy leak detection mechanism proposed by this scheme significantly improves the accuracy of experimental test results. Analyzing the reason shows that FlowDroid will mark an application with a privacy leak path as a malicious application. Therefore, the probability of misinterpreting a benign application as a malicious application increases, leading to a decrease in the overall accuracy of the detection result and a decrease in the benign application recall rate. Therefore, the accuracy rate of the test results of this program is higher than that of FlowDroid test results.

Experiment 2: Compare privacy leak detection time with PDDMSB and FlowDroid

Also, since FlowDroid takes a long time to analyze the static flow of the application, we do not perform service inspection on all application APKs. In this experiment, 10% of APK applications were randomly selected from the total sample set, for a total of 77 applications. There were 46 benign applications and 31 malignant applications. A total of two sets of experiments were performed. FlowDroid was also used for static flow analysis, but one set was based on application granularity and the other set was based on service granularity. The experimental data was grouped by

application size and the average detection consumption time for each group was calculated. The experimental results are shown in Figure 9.

From the experimental results, when the application size is 0~4M, the detection consumption time based on the application granularity is the same as the detection consumption time based on the service. However, as application size increases, service-based detection consumes significantly less time. Since the privacy leak detection mechanism based on the service granularity starts to detect, some APIs corresponding to the privacy rights that are not related to the application service are filtered out, which effectively reduces the privacy permission API that the FlowDroid needs the taint analysis. Therefore, the experimental results are in line with the expectation.

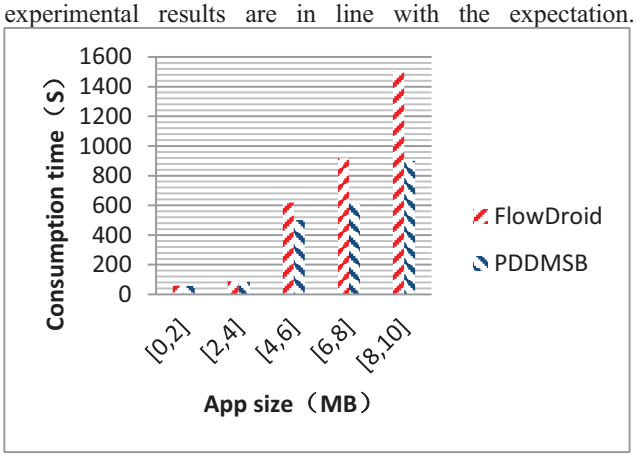

Figure 9. Two detection mechanisms detect elapsed time

6. CONCLUSION

The main work of this paper is to propose a risk detection solution based on service binding for Android application privacy right management. Service binding refers to the granulation of traditional assessment objects based on the application level into service-based. This project completes the classification of Android applications by services, and statistically obtains the minimum privacy set required for the normal execution of each service type application. At the same time, it completes the secondary development of the granular analysis of the FlowDroid tool and completes the numerical conversion of the sensitive data flow path of the Android application into a feature vector. Finally, through experimental verification, the accuracy, recall, and F-measure of PDDMSB and FlowDroid privacy leak detection results were compared. Besides, the experiment verifies that this scheme can effectively reduce the detection time. In the subsequent work, we will consider how to replace random forests with neural networks (e.g., deep neural networks) in the service security assessment module, and utilize deep learning methods to achieve the classification of feature vectors of Android applications. Subsequently, we will develop a unified application solution for different Android versions because the API functions corresponding to the privacy permissions of different Android versions will change, resulting in the invalidation of the statistical sources and sinks in the SourcesToSinks.txt file. For different operating systems, e.g., iOS and Windows, we will explore the way applications set privacy permissions and use private data, and apply the method proposed in this paper to these operating systems to realize privacy permission management.

ACKNOWLEDGMENT

We would like to thank the reviewers for their comments to help us improve the quality of this paper. This work was supported by the Key Technologies and Applications of Resource Collaborative Scheduling for Cloud-Edge-Network Integrated System Project of State Grid Corporation of China under Grant no.5700-202318292A-1-1-ZN.

REFERENCES

[1] Z. Yang and M. Yang, “LeakMiner: Detect information leakage on android with static taint analysis,” in 2012 third world congress on software engineering, 2012, pp. 101–104.

[2] Z. Zhao and F. C. Colon Osono, “‘TrustDroidTM’: Preventing the use of SmartPhones for information leaking in corporate networks through the used of static analysis taint tracking,” in 2012 7th international conference on malicious and unwanted software, 2012, pp. 135–143.

[3] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, “FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps,” in Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, in PLDI ’14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 259–269.

[4] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, “TaintDroid: An information flow tracking system for real-time privacy monitoring on smartphones,” Commun. Acm, vol. 57, no. 3, pp. 99–106, Mar. 2014.

[5] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, “These aren’t the droids you’re looking for: Retrofitting android to protect data from imperious applications,” in Proceedings of the 18th ACM conference on computer and communications security, in CCS ’11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 639–652.

[6] D. Schreckling, J. Köstler, and M. Schaff, “Kynoid: Real-time enforcement of fine-grained, user-defined, and data-centric security policies for Android,” Inf. Secur. Tech. Rep., vol. 17, no. 3, pp. 71–80, 2013.

[7] S. Arzt, S. Rasthofer, and E. Bodden, “Susi: A tool for the fully automated classification and categorization of android sources and sinks,” 2013.

[8] W. W. Hongzhou Yue, Yuqing Zhang, “Android static taint analysis of dynamic loading and reflection mechanism,” Comput. Res. Dev., vol. 54, no. 313–327, 2017.

[9] Y. Zhao, G. Yi, F. Liu, Z. Hui, and J. Zhao, “A Framework for Scanning Privacy Information based on Static Analysis,” in 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), Guangzhou, China: IEEE, Dec. 2022, pp. 1135–1145.

[10] Y. Wu et al., “TraceDroid: Detecting Android Malware by Trace of Privacy Leakage,” in Wireless Algorithms, Systems, and Applications, vol. 13471, pp. 466–478, 2022.

[11] I. W. B. Sentana, M. Ikram, and M. A. Kaafar, “An Empirical Analysis of Security and Privacy Risks in Android Cryptocurrency Wallet Apps,” in Applied Cryptography and Network Security, vol. 13906, pp. 699–725, 2023.

[12] H. Cui, G. Meng, Y. Zhang, W. Wang, D. Zhu, T. Su, “TraceDroid: A Robust Network Traffic Analysis Framework for Privacy Leakage in Android Apps,” in Science of Cyber Security, vol. 13580, pp. 541–556, 2022.

[13] C. Schindler, M. Atas, T. Strametz, J. Feiner, and R. Hofer, “Privacy Leak Identification in Third-Party Android Libraries,” in 2022 Seventh International Conference On Mobile And Secure Services (MobiSecServ), Gainesville, FL, USA: IEEE, Feb. 2022, pp. 1–6.

[14] X. Hao, D. Ma, and H. Liang, “Detection and Privacy Leakage Analysis of Third-Party Libraries in Android Apps,” in Security and Privacy in Communication Networks, vol. 462, pp. 569–587, 2023.