Privilege Escalation Attack Detection Method for Android Applications

Hui Li[1,a], Limin Shen[2.b.*], Chuan Ma[2.c], Meimei Wang[2.d], Honglei Tan[3,e] and Hongwei Zhang[4,f]

[1]School of Information Science and Engineering, YanshanUniversity; School of Business Administration, Hebei Normal University of Science &Technology, Qinhuangdao, China
[2]School of Information Science and Engineering YanshanUniversity, Qinhuangdao, China
[3]School of Continuing Education YanshanUniversity, Qinhuangdao, China
[4]School of Business Administration, Hebei Normal University of Science &Technology , Qinhuangdao, China
a. lh_23@163.com, b. shenllmm@sina.com, c. tianyi_mc@126.com,
d. wantong_ysu@163.com, e. thl_23@163.com, f. chengxuedong007@126.com
**corresponding author: Limin Shen*

Keywords: Privilege escalation attack, privacy data, component communication, role model, sensitive path pairs.

Abstract: For the problem of the user's privacy data obtained through conspiracy attack with privilege escalation in a number of applications of the Android system, the detection method was proposed based on permission, component communication, data flow and role model. We classified the roles based on the detection of component's communication, component's permission and sensitive path pairs of application's components, and finally the sensitive information flow paths in multi-role were detected, thereby the detection method constituting to privilege escalation attack for multi-application was ascertained. The experience result showed that we proposed method was effective for detect privilege escalation attack, and pointed out applications with potential security hazards.

1. Introduction

With the popularity of Smartphone and the expansion of the mobile Internet, it has become an indispensable part of learning, living, entertainment, and transportation. A new mobile Internet service is proposed based on applications in order to adapt to the development of mobile Internet at the background of constantly promoting Industrial 4.0 Development[1,2]. Android operating system, which accounts for 37.55% of OS market share[3], will become the top choice for enterprises and individual users. However, Android Smartphone not only faces the problem of mobile phone quality defects, but also faces the threat of malware to enterprise and personal information[4,5]. According to the threat report released by Nokia Threat Intelligence Laboratories, Android devices were responsible for 47.15% of the observed malware infections in mobile networks[6]. According to the "China mobile phone security report in the first quarter of 2020" released by 360 Internet Security Center, 392,000 new malicious applications were added to mobile platform[7].

Attack on Smartphone is no longer limited to single and isolated intrusion and attack behaviors, coordinated collusion attack have emerged, for example compound attacks, multithreading attacks, distributed intrusions, and coordinated attack[8]. The behavior of a single program will not threat to system security, but the combined effect of several concurrency programs may have serious consequences[9]. Traditional methods such as vulnerability detection and privacy detection are difficult to cope with coordinated collusion attack. The Smartphone obtain software and hardware information in the device and stealing enterprise information and user privacy data by privilege escalation and coordinated collusion attack.

Aiming at the multi-application collusion privilege escalation attack of Android, we established privilege escalation attack roles model, and detected sensitive information flow path in multi-application based on this model, thereby the application constituting the privilege escalation attack and application's role were determined.

2. Related Work

For Android security threats, such as privilege escalation attack, Felt et al.[10] proposed IPC Inspection to prevent privilege escalation attack and it must reduce the set of permissions and delete the permissions that other applications do not have, when an application receives messages from other applications. Verma et al.[11] put forward a hybrid method for Android malware by analyzing the permissions of the AndroidManifest.xml file and the Intents. Feizollah et al.[12] used the evaluation of Intents (explicit and implicit) as the way to identify malicious applications. Bedford et al.[13] used machine learning and static analysis to detect Android malicious applications. Cam et al.[14] proposed using static and dynamic analysis to detect sensitive resource accessing and leakage. Wang Cong et al.[15] proposed detection methods of privilege escalation attack based on component, application layer and stacking defect. Yu Da[16] proposed monitoring the list of parameters in the function code to indentify sensitive information, and divided the code function into permission leak function and privacy leak function. DroidAuditor[17] used the Android Security Modules access control architecture to detect application-layer privilege escalation attacks. Wu Dong-Jie et al.[18] proposed a method for detecting Android malware based on static feature mechanism. Dasgupta et al.[19] introduced a multi-user permission strategy based on user requested information sensitivity, access control charts and a set of permission grantees.

3. Role Model of Privilege Escalation Attack

The Android platform uses the permission mechanism to control access to sensitive APIs. The latest Android version adopts that apply for dangerous permission on runtime. When application wants to access sensitive resources during the running process, it will apply for permission. Once the user authorized, the user will be authorized forever. However, Android's permission mechanism has the threat of privilege escalation attack[20].

3.1.Privilege Escalation Attack Case

The privilege escalation attack model describes a problem with the privilege mechanism: for an application with less permission, it can access components of the application that have more permission than itself, thereby gaining permissions that it does not have. Privilege escalation attack can be divided into two categories[21]: the first is confusing agent attack that using an unprotected interface of safety applications, and the other is conspiracy attack that combining multi-application to gain greater privilege.

Applications A, B, and C were run independently. Applications A, B, and C were the client of the enterprise information system or the application installed into user's Android device. Application A was responsible for the collection of information, application B was responsible for the delivery of information, and application C used SEND_SMS dangerous permissions to send sensitive information. The privilege escalation process of the three applications is shown in Figure 1, which shows that:

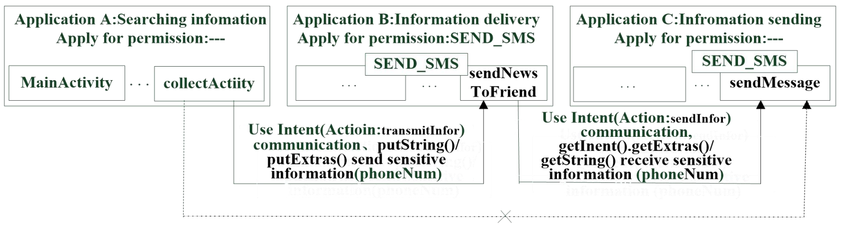

Application A:Searching infomation
Apply for permission:---
MainActivity
collectActiity
Application B:Information delivery
Apply for permission:SEND_SMS
SEND_SMS
sendNews ToFriend
Application C:Infromation sending
Apply for permission:---
SEND_SMS
sendMessage
Use Intent(Actioin:transmitInfor) communication、putString()/ putExtras() send sensitive information(phoneNum)
Use Intent(Action:sendInfor) communication, getInent().getExtras()/ getString() receive sensitive information (phoneNum)

Figure 1: Privilege escalation process.

(1) Three applications implemented the promotion of the dangerous permission "SEND_SMS".

(2) Three applications used Intent communication, then it got sensitive data through getInent().getExtras(), getString() methods, sent sensitive data through putString(), putExtras() methods, and finally realized unauthorized sending of sensitive information.

3.2. Construct Roles Model of Privilege Escalation Attack

Based on the concept and case of privilege escalation attack, the App was divided into three types of roles: searching information App, information delivery App, and information sending App. For the convenience of description, the following definitions were given.

Definition 1: Searching Information App. It refers to obtain the privacy information of enterprises or users in various ways, represent by SIApp.

Definition 2: Information Delivery App. It refers to transfer privacy information to other applications through inter-application communication, represent by IDApp.

Definition 3: Information Sending App. It refers to use dangerous APIs to send privacy information or to perform destructive operations, represent by ISApp.

Definition 4: Enter Events. It refers to App's components have events that obtains data passed by other components, represent by EET.

Definition 5: Outside Events. It refers to App's components have events that send data to other components, represent by OET.

Definition 6: Carry Data Events. It refers to the EET/OET components that carry data, represent by CDE.

Definition 7: Dangerous Events. It refers to call sensitive APIs or sensitive data flow, such as privacy leaks or dangerous operations, represent by DGE.

Definition 8: App's permission. App A has P1 permission, represent by A.P1.

Definition 9: Component's permission. Component Com of Application A has P1 permission, represent by A.Com.P1.

Definition 10: Sensitive information flow path. There are sensitive information flow path among the three roles which constitute privilege escalation attack, represent by SFP.

Based on Definition1-9, the roles of privilege escalation attack were modelled:

Model of SIApp: Component Com has OET(OET.CDE) at least. That is,

$$SIApp = \{SIApp.Com.OET \parallel SIAPP.Com.OET.CDE\} \qquad (1)$$

Model of IDApp: Component Com has EET(EET.CDE)/OET(OET.CDE) and dangerous permission P1at least. That is,

$$IDApp = \{(IDApp.Com.EET\&IDApp.Com.OET\&IDApp.P1) \parallel (IDApp.Com.EET.CDE\&IDApp.Com.OET.CDE\&IDApp.P1) \parallel (IDApp.Com.EET\&IDApp.Com.OET.CDE\&IDApp.P1) \parallel (IDApp.Com.EET.CDE\&IDApp.Com.OET\&IDApp.P1)\} \quad (2)$$

Model of ISApp: Component Com has EET(EET.CDE),DGE and dangerous permission P1. That is,

$$\text{ISApp} = \{((\text{ISApp.Com. EET\&ISApp.Com.P1\&ISApp.DGE}) \parallel (\text{ISApp.Com.EET.CDE\&ISApp.Com.P1\&ISApp.DGE})\} \quad (3)$$

4. Key Technologies and Algorithms of Detection Method

4.1.Design of Detection Method

Based on roles model, the design of the detection method is shown in Figure 2.

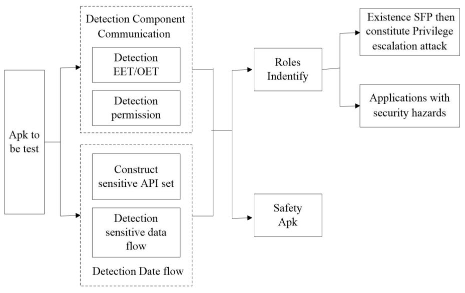

Detection Component Communication
Detection EET/OET
Detection permission
Apk to be test
Construct sensitive API set
Detection sensitive data flow
Detection Date flow
Roles Indentify
Safety Apk
Existence SFP then constitute Privilege escalation attack
Applications with security hazards

Figure 2: Design of detection method.

4.2.Detection Component Communication

The Android framework provides four types of components for application[22].

(1) Activity (user interface) focuses on user behavior.

(2) Service (background service) implements background activities.

(3) Content Provider defines storage like a database.

(4) Broadcast Receivers listen for global events.

Communication between different components in Android applications is achieved through Intent mainly. Activity, Service, Broadcast Receiver were started through canned Intent's "Call Intention". If the application just wants to start components with certain features, Intent can realize coupling without a specific component, which is benefit to high-level uncoupling. Similarly, Intent can communicate between different components in different applications. Android has a perfect communication mechanism between components. Components can restrict access to other components through permission or label protection.

1) Detection EET/OET. An unencrypted APK is decompiled and converted into Smali file for analysis and detection. Smali files are the core code of executed Dalvik VM. For any single Smali file, information (such as classes, packages, inheritance, interfaces, functions and function calls and etc.) can be obtained. EET/OET of the APK components was detected and stored it as two-dimensional tables. The detection methods were as follows:

Step 1: According to each method (such as direct method / virtual method), get function call relationship;

Step 2: Detect special function calls (such as getExtras, putExtras, etc.);

Step 3: Generate the component's "EET / OET" table.

2) Detection Permission. AndroidManifest.XML provides the Android system with the necessary information to run the application. Whether the permission mechanism uses runtime applied for permission or installation-time applied for permission, relevant information must be registered in the AndroidManifest.XML. AndroidManifest.XML contains the application's package name, registration components, registration component function, declaring what permissions must apply for to access protected content in the API.The detection methods were as follows:

Step 1: Get information of the application's permissions and registration components according to the tags;

Step 2: Get information of the component's permissions according to the tags;

Step 3: Save the permission detection into the component's "EET/OET" table;

Step 4: Generate the component's "permission + EET / OET" table.

4.3.Detection Date Flow

Sensitive API set was constructed firstly, and then sensitive data flow detection was performed using FlowDroid [23].

4.3.1. Construct Sensitive API Set

In Android permission system, permissions are divided into normal and dangerous[22]. Normal permissions do not threaten user's privacy directly, such as network connection status permission ACCESS_NETWORK_STATE. Dangerous permissions allow users to access sensitive data, such as permission READ_CONTACTS that get contact list. Google's official documents provide a list of normal and dangerous permissions. Kathy et al. developed PScout[24], it analyzed and summarized permissions of multiple versions of Android system and their corresponding API relationships. Sensitive API set was constructed base on above two points. The corresponding relationship between SEND_SMS permission and sendTextMessage () API in privilege escalation attack case is shown in Table 1.

Table 1: Examples of sensitive API sets.

Permission	corresponding APIs
SEND_SMS	void enforceReceiveAndSend(java.lang.String)
	void sendTextMessage(java.lang.Strng.java.lang.String,
	booleancopyMessageToIccEf(int,byte[],byte[])
	booleanupdateMessageOnIccEf(int,int,byte[])
	…

4.3.2. Detection Sensitive Data Flow

FlowDroid is a static stain analysis tool for Android applications, and analysis system calls through accurate Android life cycle model.

Definition 11: Sensitive Path Pairs. In order to illustrate the origination of source, sink and system calls, represent by (source, sink), source consists of two parts: originating component.calling function/originating class. calling function. For example, component A. methodA represents component A calling method A; class B. methodB represents class B calling method B. The expression for sink is the same as source. The complete expression is as follows:

(Originating Component.Calling Function/Originating Class.Calling Function, Originating Component. Calling Function / Originating Class. Calling Function), represent by SSP.

FlowDroid was used to detect the sensitive data flow of the application, so SSP (Definition 11) was got. SSP was added to the table that generated by detection permission, then the component’s "permission + EET / OET + SSP" table was formed.

4.4.Detection SFP

Step 1: Store detection component results. According to the detection results of the application, each component in the application was represented and stored in an array, such as: application’s name. component’s name (Permission, EET, EET. CDE, OET, OET. CDE, SSP). If there is no content, it is expressed by NULL.

Step 2: Construct the basic search algorithm. A string with "application name. component name" as an array object was constructed for test application. Each item in the array was feature of the application component. An extended BF algorithm was used to store the final results. The result was stored into array object string.

```
Algorithm 1: BF algorithm base on object feature
Input： array object string T， search object P
Output：array object string result
1. n=T.length
2. m=P.length
3. for s=0 to n-m
4.if P[1…m] ==T[s+1,s+m]
5. print (P,T[s]) //T[s] searched array object
6. add(P,T[s]) to array object string result
```

Step 3: Roles identification. Base on component’s array and roles model, the role of application was identified.

```
Algorithm 2: Roles identification
Input：array object string comArray, string IDAppModel,
         string ISAppModel, string SIAppModel
Output：String IDAppArray，String ISAppArray，
         String SIAppArray
1.for each arrayObject in comArrary string
2. if comArrayObj_i accord with IDAppModel Then
3. add comArrayObj_i to IDAppArray
4. else if comArrayObj_i accord with ISAppModel Then
5. add comArrayObj_i to ISApparray
6. else if comArrayObj_i accord with SIAppModel Then
7. add comArrayObj_i to SIAppArray
8. end if
9.print IDAppArray, ISAppArray, SIAppArray
```

Step 4: Detection SFP (Definition 10) algorithm. For the application of three types of roles, the ISApp and SIApp were searched from IDApp as the starting point. If there is a sensitive information flow path was constituted privilege escalation attack.

```
Algorithm 3: Detection SFP
Input： String IDAppArray， String ISAppArray，
        String SIAppArray
Output： SFP
1. arrayObj$_0$=getStartPoint(IDAppArray)
2. for each arrayObject in IDAppArray
3. call Algorithm1(ISAppArray, arrayObj$_0$)
4. print array object result1 //IDApp and ISApp have path
5. searchArrayObj$_0$=result1[0][0]
6. k= result1.length
7. for i=0 to result[0][k-1]
8. call Algorithm1(SIAppArray,searchArrayObj$_i$)
9. print array object result2 //IDApp and SIApp have path
10. Combine reult1 and result2 then add to SFP
11. print SFP
```

5. Method Experiment

5.1.Detection the Case

This method was used to detect the case of privilege escalation attack in Section III.

1. Detection Component's EET/OET. According to component communication detection method in Section IV, the component sendNewsToFriend of application B was detected, as shown in Table 2.

Table 2: Detection EET/OET of sendNewsToFriend.

Detection content	Detection result
ComponentName	sendNewsToFriend
EET	getInent().getExtras()
EET.CDE	getString()
OET	Action:sendInfor
OET.CDE	putString(),putExtras()

2. Detection Component's Permission. According to permission detection method in Section IV, the components, permission and Intent of application B were detected, as shown in Table 3.

Table 3: Detection component,permission and Intent.

MainActivity	Intent_action:android_intent_action _MAIN	android.permission.INTERNET android.permission.SEND_SMS
sendNewsToFriend	Intent_action:transmitInfor	android.permission.INTERNET android.permission.SEND_SMS

3. Generate Component's table "Permission+EET/OET". The table "Permission+EET/OET" of component sendNewsToFriend was generated by Combine Table 2 and Table 3, as shown in Table 4.

Table 4: Detection component sendNewsToFriend communicate.

Detection content	Detection result
ComponentName	sendNewsToFriend
Intent/Intent_action	Intent_action:transmitInfor
Permission	android.permission.INTERNET android.permission.SEND_SMS
EET	getInent().getExtras()
EET.CDE	getString()
OET	Action:sendInfor
OET.CDE	putString(), putExtras()

4. Detection SSP. FlowDroid was used to detect application B, which had a pair of (source, sink) sensitive data flow. That is: (Bundle.getString("phoneNum")/sendNewsToFriend.onCreate()+Bundle.getString("userName1")/sendNewsToFriend.onCreate(), Bundle.putString()/sendNewsToFriend.backBtn()). As shown in Table 5.

Table 5: Detection component sendNewsToFriend SSP.

Detection content	Detection result
ComponentName	sendNewsToFriend

Source	Bundle.getString("phoneNum")/sendNewsToFriend.onCreate()
	Bundle.getString("userName1")/sendNewsToFriend.onCreate()

Sink	Bundle.putString()/sendNewsToFriend.backBtn()
SSP	(Bundle.getString("phoneNum")/sendNewsToFriend.onCreate() +Bundle.getString("userName1")/sendNewsToFriend.onCreate() ,Bundle.putString()/sendNewsToFriend.backBtn())

5. Generate Component's table "Permission+EET/OET+SSP". The complete detection result of component sendNewsToFriend was generated by Combine Table 4 and Table 5, as shown in Table 6.

Table 6: Detection component sendNewsToFriend completely.

Detection content	Detection result
ComponentName	sendNewsToFriend
Intent/Intent_action	Intent_action: transmitInfor
Permission	android.permission.INTERNET android.permission.SEND_SMS
EET	getInent().getExtras()
EET.CDE	getString()
OET	Action:sendInfor
OET.CDE	putString(), putExtras()

Source	Bundle.getString("phoneNum")/sendNewsToFriend.onCreate()
	Bundle.getString("userName1")/sendNewsToFriend.onCreate()

Sink	Bundle.putString()/sendNewsToFriend.backBtn()
SSP	(Bundle.getString("phoneNum")/sendNewsToFriend.onCreate()+ Bundle.getString("userName1")/sendNewsToFriend.onCreate(),B undle.putString()/sendNewsToFriend.backBtn())

6. Repeat Step1-Step5 to complete the detection of components collectActivity and sendMessage in application A and application C. The complete test results of application A's component collectA ctivity and application C's component sendMessage were shown in Table 7 and Table 8.

Table 7: Detection component collectActivity completely.

Detection content	Detection result
ComponentName	collectActivity
Intent/Intent_action	Intent_action:android_intent_action_MAIN
Permission	NULL
EET	NULL
EET.CDE	NULL
OET	Action: transmitInfor
OET.CDE	putString(), putExtras()
Source	NULL
Sink	NULL
SSP	NULL

Table 8: Detection component sendMessage completely.

Detection content	Detection result
ComponentName	sendMessage
Intent/Intent_action	Intent_action:sendInfor
Permission	android.permission.SEND_SMS
EET	getInent().getExtras()
EET.CDE	getString(),getExtra(keyInfor)
OET	NULL
OET.CDE	NULL
Source	Bundle.getString()/sendMessage.onCreate()
Sink	Log/sendMessage.sendSMSMessage
SSP	(Bundle.getString()/sendMessage.onCreate()， Log/sendMessage.sendSMSMessage)

7. Indentify roles. According to the role model in Section III and algorithm 2, the roles of applica tion A, B and C were indentified.

(1) According to Equation (2) and algorithm 2, the model of Application B is shown in Equation (4):

$$\begin{aligned} B = {} & (\mathrm{B.sendNewsToFriend.EET.CDE} \\ & \&\ \mathrm{B.\ sendNewsToFriend.OET.CDE}\ \&\ \mathrm{B.INTERNET}) \\ & \&\&\ (B.\ sendNewsToFriend.EET.CDE\ \& \\ & B.\ sendNewsToFriend.OET.CDE\ \&\ B.SEND\_SMS) \end{aligned} \tag{4}$$

So application B belongs to IDApp。

(2) According to Equation (1) and algorithm 2, the model of Application A is shown in Equation (5):

$$A = A.collectActivity.OET.CDE \tag{5}$$

So application A belongs to SIApp.

(3) According to Equation (3) and algorithm 2, the model of Application C is shown in Equation (6):

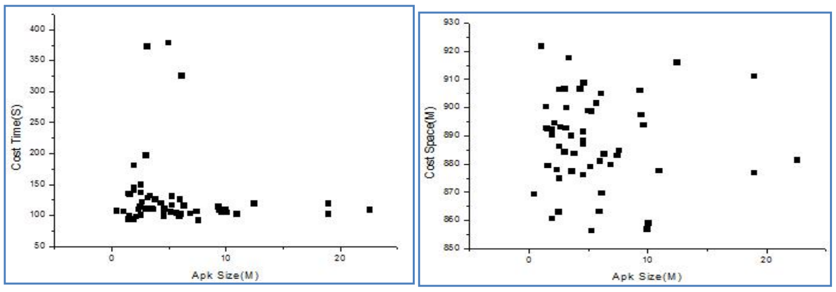
$$\begin{aligned} C = {} & C.sendMessage.EET.CDE\ \& \\ & C.sendMessage.SEND\_SMS\ \&\ C.DGEA \end{aligned} \tag{6}$$

So application C belongs to ISApp.

8. Detection SFP. (A.collectActivity)—>(B.sendNewsToFriend)—>(C.sendMessage) can be obtained by using algorithm 3. Three applications constituted privileged escalation attack.

5.2.Experimental Evaluation

The key steps of this detection method were sensitive data flow detection, role identified and dangerous path detection. The time complexity of the algorithm is O ((n-m+1) m) at worst and O (n) at best. Because the paths of sensitive data caller and callee are different, the time cost and space cost distribution was hashed when sensitive data flows were detected. 54 Android APKs of the sample were tested, and time cost and space cost were shown in Figure 3(a) and Figure 3(b).



(a) Cost time (b) Cost space

Figure 3: Cost time and Cost space.

5.3.Method Comparison

It can be seen that after APK detection, the following contents can be obtained:

(1) Multi-application that constitutes privilege escalation attack in APK sample was detected.

(2) The APK had the risk of permission and component, and its role was indentified.

(3) The APK with potential safety hazards was indentified.

According to the component-based detection method in reference[15], 54 APKs were tested, and the comparison was shown in Table 9. Thus, we proposed method that can detect the privileges escalation attack and risks effectively.

Table 9: Detection data comparison.

	component-based detection	Our method
Application with permission risk	53.7%	94.4%
Application with component risk	53.7%	87.0%
Application constituted privilege escalation attack	0%	5.5%
Safe Application	46.3%	27.8%

5.4.Experimental Validity

48 Apps were extracted from the Android application market such as Google Play, and 3Apps for enterprise information management and 3 experimental Apps were developed by our research team, those constituted the sample set together. The validity of this method was verified by testing the sample set.

In order to detect different kinds of applications and applications of the same enterprise or developer, 13 types of Apps were selected in the sample set, which 18.5% came from the same enterprise or developer. The results showed that IDAPP accounted for 31.5%, while safe application accounted for 27.8%, and three APPs were detected to constitute privilege escalation attack, as shown in Table 10.

Table 10: Role Distribution of privilege escalation attack.

Role	Number	Percentage
SIAPP	2	3.7%
IDAPP	17	31.5%
ISAPP	3	5.5%
Application with potential security hazards	17	31.5%
Safe application	15	27.8%
Application constituted privilege escalation attack	3	5.5%

6. Conclusions

We proposed a detection method of privilege escalation attack based on permissions, components, roles model, component communication and data flow. Firstly, the design of the detection method was given, and component's communication and date flow detection were carried out. Then the role of application was indentified, and SFP among multiple applications were searched, so the applications of privilege escalation attacks were detected. Finally, the correctness of this method was proved by the case. The validity of this method was verified by 54 APKs. Experiments showed that we proposed method can detect the attack and the potential security hazards effectively. It is more comprehensive than the traditional detection method. Therefore, it has great significance to prevent enterprise and user information from leaking through Android application privilege escalation attack. However, we have further work to be done:

(1) There are false alarms. If the communication between applications is normal, it will lead to false alarm. The next step will be to improve the model in order to eliminate false alarm.

(2) Improve the accuracy of the method. In order to improve the accuracy of the method, the construction of role model and sensitive API set needs to be further refined and strengthened.

Acknowledgments

National Natural Science Foundation of China(No. 61772450), Natural Science Foundation of Hebei Province(No.F2017203307), and Science and Technology Project of Hebei Province(No.17210701D).

References

[1] P. Wang, S. L. Ge, N. X. Wang, Y. H. Pan, and N. Ren, "MIS 4.0 Research for Industrie 4.0," Computer Integrated Manufacturing Systems, vol.22, no.7, pp. 1812-1820, 2016.

[2] S. Li, "Research and Implementation of Mobile Terminal Oriented Enterprise Information System," Beijin: North China Electric Power University, 2016.

[3] "Os-Market-Share in January 2019", Available athttp://gs.statcounter.com/os-market-share,Last accessed onJanuary 11, 2019.

[4] "China Mobile Phone Security Ecology Research Report in 2018", Available at http://zt.360.cn/1101061855.php?dtid=1101061451&did=610082749, Last accessed onFebruary1, 2019.

[5] S. Zhang, J. Y. Wu, W. G. Fan, and S. K. Liu, "Defect Discovery of Phones based on Social Media Analytics," Computer Integrated Manufacturing Systems, vol.22, no.9, pp. 2264-2273, 2016.

[6] "Nokia Threat IntelligenceReport-2019", Available athttps://networks.nokia.com/solutions/threat-intelligence,Last accessed onJanuary 30, 2019.

[7] "China mobile phone security report in the first quarter of 2020", Available at https://zt.360.cn/1101061855.php?dtid=1101061451&did=610519685, Last accessed on May 15, 2020.

[8] G. L. Jin, L. H. Song, W. Zhang, and S. Lu, "Automated Atomicity-Violation Fixing," Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation,pp. 389-400,2011.

[9] G. Geeraerts, A. Heubner, and J. F. Raskin, "On The Verification of Concurrent, Asynchronous Programs with Waiting Queues," ACM Transactions on Embedded Computing Systems, vol.14, no.3, pp. 1-26, 2015.

[10] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission re-delegation: attacks and defenses", SEC'11 Proceedings of the 20th USENIX conference on Security, pp. 22-22, 2011.

[11] S. Verma and S. K. Muttoo, "An Android Malware Detection Framework-based on Permissions and Intents," Defence Science Journal, vol.66, no.6, pp. 618-623, 2016.

[12] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection," Computers and Security, vol.65, pp. 121-134, 2017.

[13] A. Bedford, S. Garvin, J. Desharnais, N. Tawbi, H. Ajakan, F. Audet, and B. Lebel, "Andrana: Quick and Accurate Malware Detection for Android," Foundations and Practice of Security - 9th International Symposium, pp. 20-35,2017.

[14] N. T. Cam and N. C. H. Phuoc, "NeSeDroid-Android Malware Detection based on Network Traffic and Sensitive Resource Accessing," Proceedings of the International Conference on Data Engineering and Communication Technology, pp. 19-30, 2017.

[15] C. Wang, R. B. Zhang, and G. Li, "Technology of Detection for Privilege Escalation Attack on Android," Transducer and Microsystem Technologies, vol.36, no.1, pp. 146-148, 2017.

[16] D. Yu, "Research and Implementation of A Detection Method for Privilege Escalation Attack of Android System,"Bei Jing: Peking University, 2013.

[17] S. Heuser, M. Negro, P. K. Pendyala, and A. R. Sadeghi, "DroidAuditor: Forensic Analysis of Application-Layer Privilege Escalation Attacks on Android,"International Conference on Financial Cryptography and Data Security, pp.260-268, 2017.

[18] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu, "DroidMat: Android Malware Detection through Manifest and API Calls Tracing,"Proceedings of the 2012 Seventh Asia Joint Conference on Information Security,pp.62-69, 2012.

[19] D. Dasgupta, A. Roy, and D. Ghosh, "Multi-User Permission Strategy to Access Sensitive Information," Information Sciences,pp. 24-49, 2018.

[20] L. Davi, A. Dmitrienko, A. R. Sadeghi, and M. Winandy, "Privilege Escalation Attacks on Android," Proceedings of the 13th international conference on Information security,pp. 346-360, 2011.

[21] J. W. Zhu, L. W. Yu, Z. Guan, and Z. Chen, "A Summary of Android Security," Application Research of Computers,vol.32, no.10, pp.2881-2885, 2015.

[22] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permission Demystified," Proceedings of the 18th ACM conference on Computer and communications security, pp. 627-638,2011.

[23] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps," Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation,pp. 259-269, 2014

[24] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing The Android Permission Specification," Proceedings of the 2012 ACM conference on Computer and communications security,pp.217-228, 2012.