Research on Non-Authorized Privilege Escalation Detection of Android Applications

Yaping Yang[1]
[1]Shanghai Key Laboratory of Computer Software Testing & Evaluating
Shanghai Development Center of Computer Software Technology
Shanghai, China
yyp@ssc.stn.sh.cn

Lizhi Cai, Yanguo Zhang*
Shanghai Key Laboratory of Computer Software Testing & Evaluating
Shanghai Development Center of Computer Software Technology
Shanghai, China
{clz, zyg}@ssc.stn.sh.cn

Abstract—This paper briefly analyzes the security mechanism of android platform and describes permission statement and request of android application. According to android permission mechanism and its shortcoming, the android privilege escalation detection technology and attacking principle is mainly analyzed, and privilege escalation attacking model and architecture of application is put forward. Penetration testing tool Drozer is used to detect application permission. Application package and Manifest file is analyzed to obtain privilege elevation vulnerability. At the same time, sensitive combination permission is distinguished to avoid and exclude the use of malicious code.

Keywords—android application; permission; privilege escalation;

I. INTRODUCTION

With the rapid development of 4G, 5G network, smart city and mobile internet, security of mobile internet has become more and more severe. It has been already gotten concern from all the world and society. As we know, security awareness of mobile phone users is very weak. At the same time, malicious code appears one after another, which leads to user privacy information leakage, phone fees deliberately deducted or account money transferred and any other risks. Therefore, it has become particularly important to protect users' privacy information security that application and system permission is deeply understudied, verified and researched.

Currently, it has become more and more important to study on android security assessment domestically and internationally. W.Enck, M.Ongtang and P.McDaniel put forward a method on permission authentication during the application installation process [1]. Thomas Biasing, Leonid Batyuk put forward android application sandbox mechanism, which can detect malicious code by static and dynamic detection method [2]. Wook Shin, Shinsaku Kiyomoto et al, propose entity - relationship model which is based on android system to analyze the permission to reduce malicious software and trojan horse [3]. It is very important to apply permission for the application system. If applying permission is not adequately, it could lead function or reliability to be affected. If applying permission is excessive, security issues could be brought in. This paper is mainly based on the characteristics of android permission mechanism, use dynamic detection technology to analyze and improve permissions of android application.

II. ANDROID'S SECURITY TECHNOLOGY

The general android architecture has been described as "Java on Linux". The overall architecture consists of android applications, the android framework, the Dalvik virtual machine, user-space native code and related services, and the Linux kernel multiple layers. Android applications have multiple components, including android manifest.xml,

978-1-5090-2239-7/16/$31.00 copyright 2016 IEEE
SNPD 2016, May 30-June 1, 2016, Shanghai, China

intent, activity, broadcast receiver, service, content provider. Android is equipped with multiple primary system security mechanisms, including sandbox, permission, signature, etc. Permission mechanism is the one of cores of the android mechanisms.

A. Sandbox and Signature Mechanism

Android system inherits Unix-like process isolation mechanism and the least privilege principle from Linux system and also follows the UID / GID permission model of Linux. Each application in android system runs in a separate virtual process in order to ensure each application not to be affected by other applications. The application process is regarded as a sandbox, and every process is separated in its own space to run, so as to ensure the process security. Data resource sharing between different applications need to share the same UID, make clear the attribute of SharedUserID in AndroidManifest.xml file and have the same digital signature. Digital signature is mainly used to identify the application's developer and establish a trust relationship between the application and the developer. However, digital signature is not certified by authority. Android system uses a self-certification program to implement digital signature, which is generated by using standard JAVA tools and is signed to application package.

B. Permission Mechanism

Permission information to access is declared in the corresponding file in the software development process. When the application software is installed on the android device, the system will automatically detect the digital signature of the program and permission of the declaration. After user's confirmation, the software package will be installed in the device and set appropriate permissions. For android permission security mechanism, developers must indicate the scope of their access permissions in AndroidManifest.xml file at the software development process. When software is installed to the Android platform, system will detect the program's digital signature and permission declaration. The access permission can be authorized only after obtaining user' confirmation, otherwise it is impossible to access the protection and API resource on the platform.

Permission declaration is specified by the label <permission> in AndroidManifest.xml. The permission that other components are allowed to access to the application is defined. Permission request is specified by the label <users-permission> in AndroidManifest.xml. Permission that may be needed to use at runtime is defined. The following table is permission statement instance on sending text message.

TABLE I. PERMISSION STATEMENT INSTANCE

```
<?xml version="1.0" encoding="uif-8"?>
<manifest
xmlns:android=http://schemas.android.com/apk/res/android
      package="cn.com.fetion.android"
      android: versionCode=" 1 "
      android: versionName="1.0.0">
  <application                    android:icon="@drawable/icon"
android:label="@string/app_name">
            <activity android:name=".welcomActivity"
                  android:label="@string/app_name">
          <intent-filter>
            <action android: name="android.intent.action.MAIN"/>
            <category                                    android:
name="android.intent.category.LAUNCHER/"
            </intent-filter>
      </activity>
   </application>
<uses-permission          android:          name="android.
permission.SEND_SMS">
</uses-permission>
</manifest>
```

Protectionlevel consists of four different kinds of levels as Normal, Dangerous, Signature and SignatureOrsystem. Authentication mode of application, in the process of using the permission, is limited by different protectionlevel. The following table is the four permission protectionlevel description.

TABLE II. PERMISSION PROTECTIONLEVEL DESCRIPTION

Protection Level	**Description**
Normal	A permission of the application is with minimal risk to other applications, the system, or the user.
Dangerous	A higher-risk permission that requested by an application may be displayed to the user and required confirmation before proceeding.
Signature	A permission is granted to the requesting application with the same signed certificate as the application that declared the permission.
SignatureOrSystem	The protectionLevel is the highest. The "SignatureOrSystem" permission is used for certain special situation where is android system image application.

III. Based On The Android Application Unauthorized Privileges Escalation Detection Technology

Android malware detection technology refers to computer malware detection technology, which can be divided into static and dynamic detection technology. This section explains privilege escalation model of application and static and dynamic detection technology of unauthorized privilege escalation. Android system framework and privilege escalation framework is mainly given and analyzed based on dynamic detection technology.

A. Privileges Eescalation Attack Model

Permission mechanism only considers the access permission of single application, but android system also provides inter-process communication mechanism (IPC). It does not consider the permission transfer and composition mechanism between applications. Privilege escalation attack can use permission combination to enhance the application permission. Attack model is shown as Fig 1. Suppose three applications run independently in their own Dalvik virtual machine. Applications have a plurality of Comps for collaboration between applications. Comp1 and Comp2 in APP1 do not have been granted any privileges. Two Comps in APP2 are not protected by any privileges, but have permission P1. Comp1 in APP3 is protected by permission P1, and Comp2 is protected by permission P2. Therefore, each Comp in APP1 can access the Comps in APP2, each comp in APP2 can access to Comp1 in APP3. Thus, the comp in APP1 can eventually access to comp1 in APP3 by the comp1 in APP2. This resulted in android privilege escalation attack. In view of this privilege escalation attack, access control protection of Comp1 in APP2 must be set to prevent occurrence of similar privilege escalation vulnerability attack at last. But it is also not realistic to require every software developer to consider whether application he developed will be accessed by other malicious code to obtain leapfrog permission.

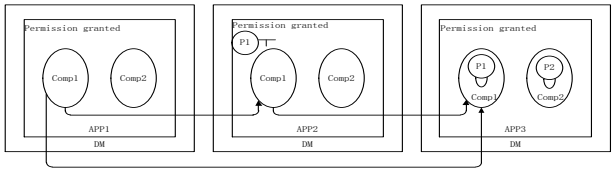

Fig. 1. Multiple applications privilege escalation attack model

B. Static Detection Technology

Static analysis techniques generally do not need to run application code directly, which is a new technology by using pattern matching technology or reverse engineering to analyze application and their supporting components code and data. Static analysis technology is generally divided into code semantic analysis and feature value analysis. Code semantic analysis focuses primarily on analysis and returns to the original intent and process. For example, the AndroidManifest.xml file information about the application request permission and internal components as Activity, Broadcast and Provider declaration is analyzed. Class.dex files can be decompiled by dex2jar tool, transformed into jar packets. The decompiled source code is analyzed on lookup sensitive function, tracking static parameters and malware behavior analysis, ultimately the malicious code is determined. Another feature value analysis method is to extract binary sequence, call function sequence and operate code sequence and any other characteristic information from the install files. Then classify to detect these information and determine malware. Bartel et al. analyze unnecessary privilege and the risk from extra privilege by using static analysis method [4] [5]. Currently static analysis tool as FindBugs, PMD, JLint is very popular.

C. Dynamic Detection Technology

The difference between dynamic analysis and static analysis technique is that behavioral characteristics of application software are obtained by executing program and simulating application tested. Monitoring tools are deployed to defense and analyze program's dynamic behavior, such as file permission changing, the system calling, network access condition, process and thread run condition and of system state changing, then malicious behavior is determined. In this paper, behavioral characteristics of application software are achieved by executing program using dynamic detection tool drozer.

D. Non-authorized privilege escalation detection analysis

Drozer, by MWR Labs, is an security testing framework for Android. It uses an agent application running on the target device, and a Python-based remote console from

which the tester can issue commands. It features numerous modules for operations like retrieving app information, discovering unprotected IPC interfaces, and exploiting the device. By default, it will run as a standard app user with only the INTERNET permission.

Before installing drozer tool, JDK, Android SDK, ADB and Java environment has to be installed and configured, and the client agent is installed on the tested device.

I. Application information is to be obtained using drozer tool, including application package and Manifest.xml file;
II. Through analyzing the package and Manifest.xml of APP, permission information and components (Activity, Service, Broadcast Receiver, Content Provider) permission information of APP has been obtained.
III. Privilege escalation is achieved by modifying the Manifest file and rebuilding the agent.

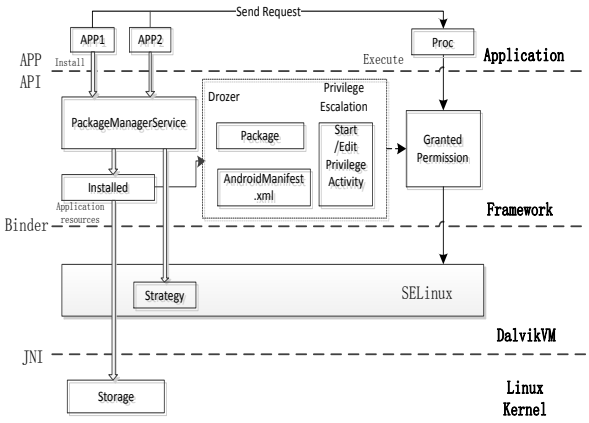

Fig. 2. Privileges escalation framework

APPs of the pre-installed and downloaded from app store are stored respectively to /system/app and /data/app directory. PackageManagerSerivice is responsible to analyze the package in the process of installation, and the relevant information, such as UID, GID and permission is generated. The service is to establish communication with installer by socket, and calls installer to install. After unzip the package, META-INF, res, class.dex, AndroidManifest.xml and resource.arsc are extracted. Analyzing the installing principle of android program, testing tool drozer is chosen to obtain package and AndroidManifest.xml file information, including application permissions and Activity, Service, Broadcast Receiver, Content Provider component information. Further component calling permissions and protection level declaration has been adopted. Simultaneously intent and permission applied of inter-application communication can be learned by analyzing call request of Binder cross-process communication. According to permissions and protection level declaration of APP, the same level of authority is obtained by the agent of drozer through modifying the Manifest file and rebuilding the agent APK.

IV. Testing Of Permissions Escalation Based On Mobile APP

Permission of application installed in Android devices MEIZU 3 (OS: Android 4.4.4, Memory: 32G) will be extracted and analyzed by drozer tool. A total of 130 applications are installed in the device, including 43 pre-installed applications and user- installed 83 applications. There are applications about chatting, video, reading, payment and game and so on. Application detail information, such as version, APK path, UID, GID, and permission information, is obtained by drozer tool. The following Fig 3 is permission information of a APP package.

```
Process Name: com.letv.android.client
Version: 6.1
Data Directory: /data/data/com.letv.android.client
APK Path: /data/app/com.letv.android.client-1.apk
UID: 10112
GID: [1028, 1015, 3003]
Shared Libraries: null
Shared User ID: null
Uses Permissions:
- android.permission.RECEIVE_USER_PRESENT
- android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
- android.permission.ACCESS_WIFI_STATE
- android.permission.ACCESS_NETWORK_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.MOUNT_UNMOUNT_FILESYSTEMS
- android.permission.FLASHLIGHT
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.BROADCAST_STICKY
- android.permission.CHANGE_NETWORK_STATE
- android.permission.WAKE_LOCK
- com.android.launcher.permission.INSTALL_SHORTCUT
- com.android.launcher.permission.UNINSTALL_SHORTCUT
- com.android.launcher.permission.READ_SETTINGS
- android.permission.ACCESS_FINE_LOCATION
- android.permission.INTERNET
- android.permission.READ_LOGS
- android.permission.VIBRATE
- android.permission.WRITE_SETTINGS
- android.permission.SYSTEM_OVERLAY_WINDOW
- android.permission.REORDER_TASKS
- android.permission.DISABLE_KEYGUARD
- android.permission.GET_TASKS
- android.permission.MODIFY_AUDIO_SETTINGS
- android.permission.EXPAND_STATUS_BAR
- com.letv.android.client.permission.JPUSH_MESSAGE
- android.permission.READ_EXTERNAL_STORAGE
- lepay.permission.ACCESS_SERVICE
- android.permission.KILL_BACKGROUND_PROCESSES
- android.permission.READ_SETTINGS
- android.permission.USE_CREDENTIALS
- android.permission.GET_ACCOUNTS
- android.permission.MANAGE_ACCOUNTS
Defines Permissions:
- com.letv.android.client.permission.JPUSH_MESSAGE
```

Fig. 3. Permission information of a APP package

Application's permission is escalated by permission escalation tool. The permission escalated information is as shown in Figure 4. Android application defaults to open a lot of user permissions during installation process, such as READ_CONTACTS, READ_SMS, and WRITE_SMS permissions. A lot of permissions are actually unnecessary, but the application also applies for the permission in system. The abuse of permission could help application in the background to achieve some unknown functions. For example, READ_CONTACTS, INTERNET, and any other permission are opened at the same time. These permissions together will be configured to dangerous permissions set. Then it may lead to the contact information disclosure through network. The more permission applied, the more likely to be utilized. Before carrying out malicious attack, a large number of permissions will be applied to system in order to invoke and use successfully.

```
Process Name: com.letv.android.client
Version: 6.1
Data Directory: /data/data/com.letv.android.client
APK Path: /data/app/com.letv.android.client-1.apk
UID: 10113
GID: [1028, 1015, 3003]
Shared Libraries: null
Shared User ID: null
Uses Permissions:
- android.permission.RECEIVE_USER_PRESENT
- android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
- android.permission.ACCESS_WIFI_STATE
- android.permission.ACCESS_NETWORK_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.READ_PHONE_STATE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.MOUNT_UNMOUNT_FILESYSTEMS
- android.permission.CAMERA
- android.permission.FLASHLIGHT
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.READ_SMS
- android.permission.WRITE_SMS
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.BROADCAST_STICKY
- android.permission.RECORD_AUDIO
- android.permission.CHANGE_NETWORK_STATE
- android.permission.WAKE_LOCK
- com.android.launcher.permission.INSTALL_SHORTCUT
- com.android.launcher.permission.UNINSTALL_SHORTCUT
- com.android.launcher.permission.READ_SETTINGS
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.INTERNET
- android.permission.READ_LOGS
- android.permission.VIBRATE
- android.permission.WRITE_SETTINGS
- android.permission.CALL_PHONE
- android.permission.SYSTEM_OVERLAY_WINDOW
- android.permission.REORDER_TASKS
- android.permission.DISABLE_KEYGUARD
- android.permission.GET_TASKS
- android.permission.READ_CONTACTS
- android.permission.MODIFY_AUDIO_SETTINGS
- android.permission.EXPAND_STATUS_BAR
- com.letv.android.client.permission.JPUSH_MESSAGE
- android.permission.READ_EXTERNAL_STORAGE
- lepay.permission.ACCESS_SERVICE
- android.permission.KILL_BACKGROUND_PROCESSES
- android.permission.READ_SETTINGS
- android.permission.USE_CREDENTIALS
- android.permission.GET_ACCOUNTS
- android.permission.MANAGE_ACCOUNTS
Defines Permissions:
- com.letv.android.client.permission.JPUSH_MESSAGE
```

Fig. 4. The permission information is escalated by tool

In the experimental process, 130 applications permissions have been analyzed. Sensitive permissions such as read user's contact, location information, sending and receiving messages, network and so on, are counted, the statistical result is as shown in Fig. 5.

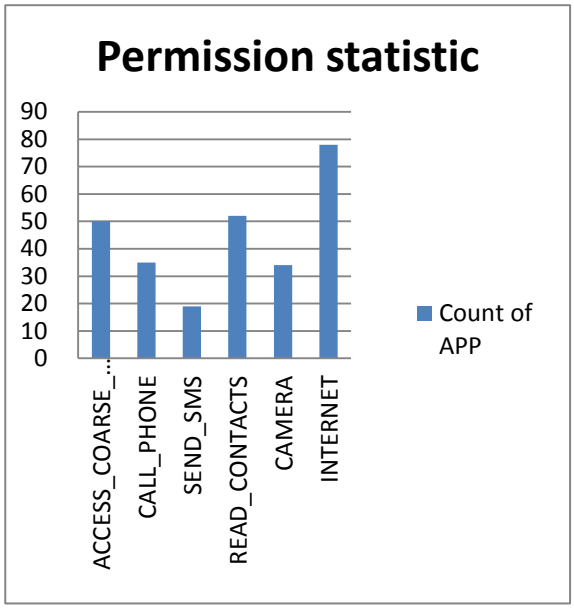

Fig. 5. Sensitive permission statistic of 130 apps

Based on permission analysis model and the application, components, and user's permission, risk component is to be extracted. There are three kinds of risk components. a) Components don't set permission protectionlevel and set low level permission. b) Component has <exported> label whose value is set as true. c) Components don't have <exported> label but with intent-filter label. Risk component recognition process is shown as Fig 6. 130 applications are installed in the device covering 43 user-installed APPs. 10 APPs are selected randomly, covering player, game, chat, news APPs and so on. For the 10 APPs, individual permissions of 7 APPs packages are set permission protectionlevel. <exported> label of some components were set as true, The other three APPs are not set permission protectionlevel and <exported> label, but with <intent-filter> label. The experimental results show that risk components are contained in the testing application. If it is utilized by the attacker, user's messages, contacts, geographical position, and so on private information can easily be gotten.

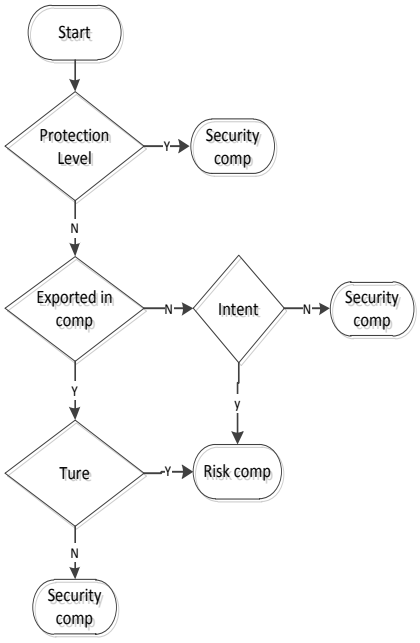

Fig. 6. Identify risk components process

Manifest files of 10 APPs are analyzed by drozer tool to obtained user-permission and components of APPs. The following table III is package 's manifest information and number of component.

TABLE III. PACKAGE'S MANIFEST INFROMANCE AND NUMBER OF COMPONENT

NO.	*Package*	*Permission*		*Activity*	*Service*	*Broadcast Receiver*	*Content Provider*
		User permission	*Defines Permission*				
1.	cn.kuwo.player	36	3	6	8	14	0
2.	com.alibaba.android.rimet	41	1	6	3	4	0
3.	com.amphibius.prison_break	0	0	1	0	0	0
4.	com.happyelements.AndroidAnimal	33	0	6	3	6	0
5.	com.icbc	48	1	7	2	9	0
6.	com.shjt.map	14	0	1	0	0	0
7.	com.sina.news	26	2	8	8	8	0
8.	com.sina.weibo	73	7	120	14	15	1
9.	com.tencent.mm	52	7	13	3	9	8
10.	com.tuniu.app.ui	33	1	7	1	2	0

V. SUMMARY

This paper based on android system structure and sandbox, permission, authentication, digital signature mechanism, analyzes privilege escalation detection technology method, which includes static and dynamic analysis technologies. Permissions of mobile applications installed in MEIZU 3X are detected, analyzed, escalated using penetration testing tool drozer and APK privilege escalation tool. AndroidManifest file, IPC mechanism and dangerous combination of components is analyzed and associated to discover the permission and combination permission that may cause user's privacy information to reveal. The next step is to build feature library for privilege escalation vulnerability to recognize and prevent malicious privilege escalation radically.

REFERENCES

[1]. W.Enck, M.Ongtang, P.McDaniel. On lightweight mobile phone application certification[C].//CCS '09 Proceedings of the 16th ACM conference on Computer and communications security. NEW YORK: ACM press,2009: 235-245.

[2]. Thomas Biasing, Leonid Batyuk, Aubrey-Derrick Schmidt. An Android Application Sandbox system for Suspicious Software Detection[C]// Malicious and Unwanted Software. Nancy : IEEE 2010 5th International Conference on,2010: 55-62.

[3]. Wook Shin, Sanghoon Kwak, Shinsaku Kiyomoto, et. A small but non-negligible aw in the Android permission scheme[C]. In Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks,2010:107-110.

[4]. A Bartel, J Klein,M Monperrus,et al. Static analysis for extracting permission checks of a large scale framework: the challenges and solutions for analyzing Android[J].IEEE Trans on Software Engineering, 2014.40(6):617-632.

[5]. A Bartel, J Klein,Y Le Traon etc. Automatically securing permission-based software by reducing the attack surface: An application to android[C]//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. NEW YORK: ACM press,2012: 274-277.