ISSN: 2278-0181
NCRTS-2015 Conference Proceedings

A Survey of user Privacy for Android Applications Based on Permission Analysis and Permission Removal

Pushpa S
M.Tech 2nd semester
Department of CSE
City Engineering College

Mukesh Kamath
Associate professor
Department of CSE
City Engineering College

***Abstract*— Android mobile devices are becoming a popular alternative to computers. The rise in the number of tasks performed on mobile devices means sensitive information is stored on the devices. Since the openness of android platform leads to a mess of privacy leaks and property damages of users and Android devices are a potential vector for criminal exploitation. This research proposes the use of permission analysis and permission removal for android application. The existing research on user privacy on android devices can be classified as android modifications; these solutions often require operating systems modifications which significantly reduce their potential. The proposed research for permission analysis is based on the detection system which is incorporated with computer terminals as well as mobile terminals and can detect the permission information of Apps and check the sensitive permission. In addition, the detect system can provide a secondary judgment of APPs to guarantee the information and property security of the users. The proposed research for the permission removal is based on the reverse engineering process. This process is used to remove an app's permission to a resource. The repackaged app will run on all devices the original app supported. Our findings that are based on a study of seven popular social networking apps for Android mobile devices indicate that the difficulty of permissions removal may vary between types of permissions and how well-integrated a permission is within an app.**

Keywords: Android platform; Permission analysis; Permission removal

I. INTRODUCTION

Android is an open source mobile operating system developed based on Linux system. Introduced by Google and its open handset alliance [10]. Android has been widely used in mobile phones, tablet PCs, laptops and other smart mobile devices. Paying bills, banking, ordering items online and others can now be done entirely on a smartphone .With the increase in the amount of sensitive information stored on a mobile device, user privacy becomes an important. As mobile device usage increases in ubiquity and capability, so will the need for increased security and privacy. Android dominates the mobile market. Because of the open source characteristic and market openness property of Android, Android is convenient for individual to release APPs development freely. Everything has two sides, on one hand, Android provides developers with the convenience of APPs development. On the other hand, it also means a convenience to criminals due to the lack of effective supervision mechanism on the publisher. The ordinary user'slack of safety knowledge is easy to download and install this malicious software, which may lead to the leak of the user's privacy information. The Google Play Store uses a blacklist style of accepting Android applications ("apps") that is all apps are accepted unless they are reported by users. Android relies on its permissions system in order to reduce the risk of a malicious app on a device. A user can manually check the list of permissions required by the app upon installation as a method to determine if it is a legitimate app.

II. RELATED WORKS

2.1 Android structure

Android apps are stored and distributed within an Android Application Package File (APK), a ZIP format file. Apps are commonly installed via the Google Play Store platform, which contains hundreds of thousands of apps created by third-party programmers and companies. Apps are generally unmoderated, and Google uses Google Bouncer [1], an in-house developed anti-malware application, to scan all submitted apps. The use of the Google Play Store allows automatic selection of appropriate app installation packages based on the device that is installing the app.

2.2 APK File Structure

An APK contains at a minimum, the directories and files shown in Figure 1. This AndroidManifest.xml file is most important. This is stored in a binary XML format and must be converted to a plain text format before becoming human-readable. This file contains information such as the minimum Android version the app was designed for, the main activity (which is launched upon opening the app) and other details important to the basic functionality of an Android app.

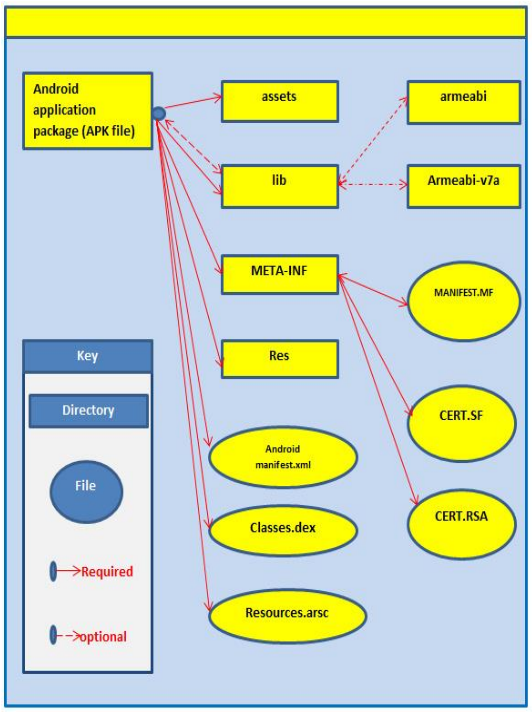
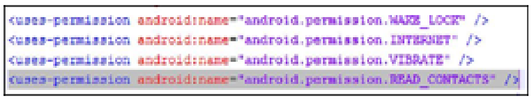


Figure 1: Overview of an APK files structure.

Most importantly for our purposes, it contains declarations of the Android permissions the app requires. Another file that will be used within this research is the classes.dex file, which contains the binary code of the app compiled to Dalvik byte code [2].Programmers are free to add as many directories and files as needed to fulfill their requirements. Due to the inclusion of the manifest filedetailing every file contained within an app, the structure is quite flexible.
Android apps are required to go through the application signing process before they can be installed onto a device. By default an Android system will not install an application if it is unsigned. This includes both physical and emulated Android systems. For an organisation that releases Android apps, there is a single private key used to sign all their applications. By signing different applications with the same private key, they are able to share code and data as Android considers them to be within the same process [3].

2.3 Android Permissions System

Android uses a permissions-based approach to user privacy and security. Each app runs in its own virtual machine process, separate from all other apps currently running. Each Android app has a unique “Linux” User ID (POSIX). Two apps with different IDs cannot run in the same process [4]. This sandbox approach ensures that app data cannot leak to other apps.Before installation of an app; a user is presented with a list of permissions the app requires. A user can only accept all permissions the app requires and install the app or cancel the installation completely. These permissions are defined by the AndroidManifest.xml file noted above, contained within the APK file in the root directory. An Android app’s list of permissions is a reflection of the functionality of that particular app. A heavily over-privileged app [an app with too many permission requests] can act as a deterrent to users due to the long, potentially suspicious list of permissions requested. As of Android version 4.2.2, the Android OS has over 120 permissions [5]. Many of these permissions, though, have little effect on the privacy concerns of an Android smartphone user and are called normal permissions.“Dangerous” permissions, on the other hand, are requested upon installation and explicitly defined in the AndroidManifest.xml file [6]. Figure 2 gives an example of a dangerous permission; the highlighted row shows that the app requests access to the user’s contacts.



```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Figure 2. Example of AndroidManifest.xml.

2.4 Android specific security mechanisms

Android system supports multi-platform operation, which uses the version of the kernel of Linux 2.6, and uses Dalvik virtual machine as an APP runtime environment. Android system has a layered architecture [11]. The bottom to the top, there are five layers, which are the Linux kernel, the local library, the Android runtime environment, the APP framework and the APP. During designing and developing the Android operating system, Google not only inherits the designing idea of Linux, but also sets a corresponding security mechanism in each layer.Google also sets two kinds of Android specific security mechanisms: signature and APP permission control.

2.5 Signature mechanism

All Android APPs must have a digital certificate, due to that the system will not install an APP that doesn’t have a digital certificate. Unlike other platforms, Android APP signature not only indicates the publisher of the APK, but also provides validation of the integrity and reliability of the program. For those who attempt to tamper with the APK file, the system will force them to re-sign the APK. Under the condition that the author's signature private key does not leak, the fake signature is almost impossible exactly the same as that of the original signature which has uniqueness. Signature mechanism plays a protective role in the APP update. Only under the circumstance that the two signatures are exactly the same, system allows the update operation. Otherwise the system will prohibit this update to further protect the security of the system.

2.6 APP permission control mechanism

Permission control is the key of Android APPs security mechanism. Android deals with security problems by means of implementation of security policies based on permission control, i.e., using permission control to restrict the APP installation, so that the APP can only access API and resources within the permission. Androiddefines 135 kinds of thesystem permissions which are divided into four protection levels [5], which are normal, dangerous, signature and signatureOrSystem, respectively. All the permissions and related functions can be seen in the development document of the Android system [12].

By default, Android APPs don't have any permission. Permissions involved in the APP runtime need to be declared in the label of uses-permission in the AndroidManifest.xml of APK file. At the time of installation, Android APP package manager will prompt the user of the application of the APP permissions only with the authorization of the user, the installation can begin, and otherwise, installation will be cancelled. After successful installation, the system will answer the requests for program to access resources according to the solidification permission information when APP runs. If there are corresponding permissions, access successes; otherwise the APP will be forced to shut down by system. The specific process is shown in figure 3.

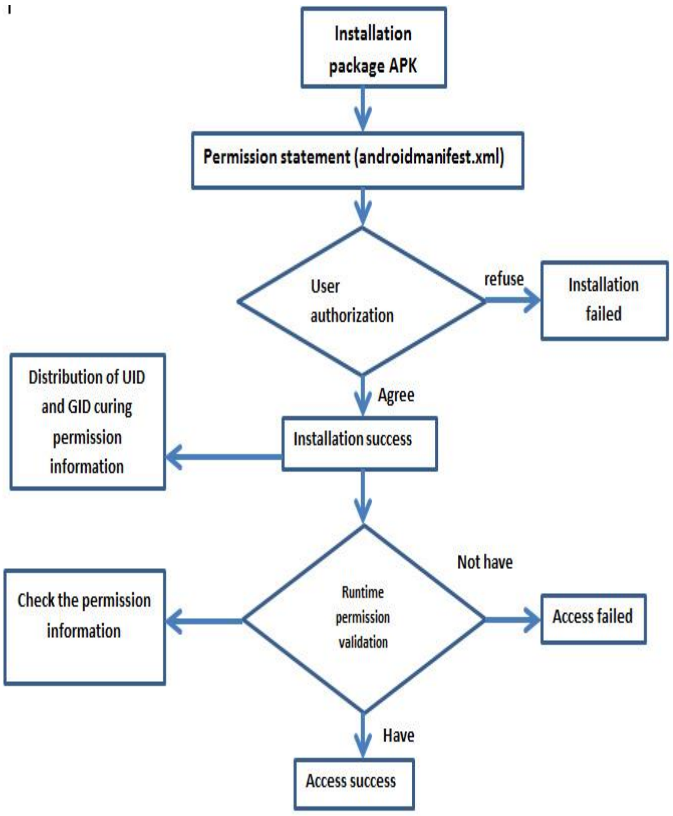
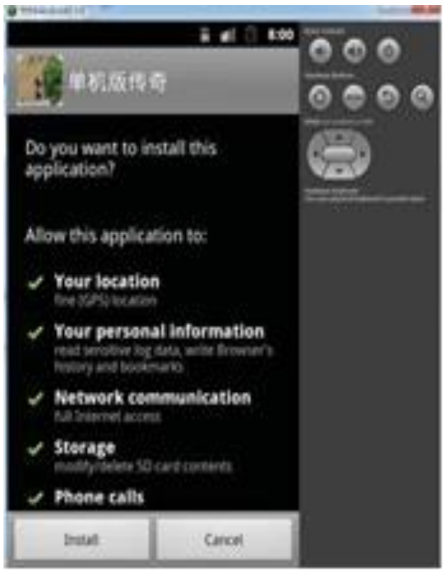

Installation package APK
Permission statement (androidmanifest.xml)
User authorization
refuse
Installation failed
Agree
Distribution of UID and GID curing permission information
Installation success
Not have
Check the permission information
Runtime permission validation
Access failed
Have
Access success

Figure 3. Permission statement and validation process when APP installs and runs.

Android permission mechanism exists obvious safety defects, i.e., when permissions are authorized to the APP by the user, the authorization will followed the APP though its whole life and cannot be removed even though the source program is deleted[13], which will lead to potential security flaws. To reduce this risk, it is required that the user has the ability to distinguish permissions information to decide whether authorize or not when APP is installed. There is no doubt that it is rather difficult for ordinary users. For the hidden safe troubles that may be caused by the permissions mechanism, the Android system gives only sketchy permissions prompt interface during APP installation, as shown in figure 4. Permission entry in this interface is incomplete, which makes the ordinary users confused and headache. But with the purpose to use the APP, users usually grant permissions to the APP, which results in the wide spread of malicious software's. Though some of the mobile phone housekeeper software's, such as king soft mobile guards, ten cent security housekeeper, can provide query of permissions information of the APP , it is too rough to show accurate information. In addition, if the APP is connected to the Internet to upgrade, the new version may apply for the new permissions in the update. Therefore, helping users understand the various permissions information of APPs and helping them judge selection and constantly monitoring permissions information and upgrade situation become very important.



Do you want to install this application?

Allow this application to:

✓ **Your location**
fine (GPS) location

✓ **Your personal information**
read sensitive log data, write Browser's history and bookmarks

✓ **Network communication**
full Internet access

✓ **Storage**
modify/delete SD card contents

✓ **Phone calls**

Figure 4. Permissions prompt interface when APP installed in the simulator Android2.3.3

III. ARCHITECTURE

3.1Architecture for permissions detection system

A permission detection system which is combined with PC and mobile phone side is proposed. The system can detect permission information of installed APPs and the uninstalled APK file in advance. The system architecture shown in figure 5 consists of two modules: permissions on the basis of decompilation module and permissions on the basis of the PackageManager module, among which, the second one can be deeply divided into two parts: extracting the corresponding permissions for each APP according to the list of APPs, and listing the permissions; monitoring all the APPs that using a specific sensitive permission according to a number of sensitive permissions, and listing APPs. By combining the two monitoring functions, the system can help users understand the permissions deeply and constantly monitor the APP permission information so that it can help guarantee the safetyof the system, and protect users' privacy information security.

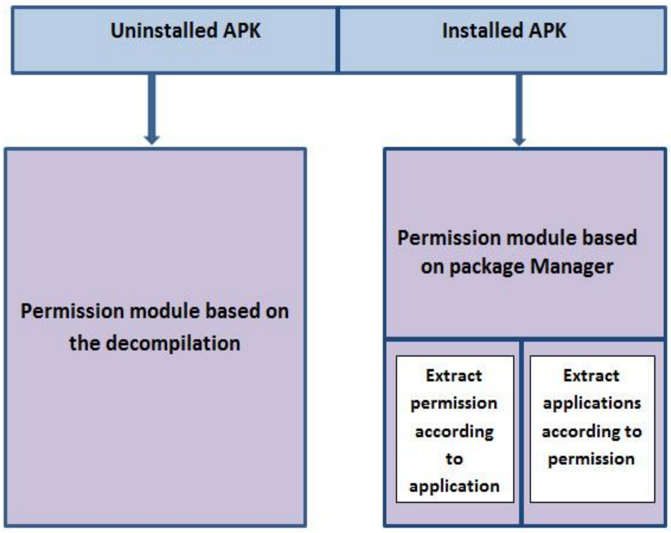

Uninstalled APK
Installed APK
Permission module based on the decompilation
Permission module based on package Manager
Extract permission according to application
Extract applications according to permission

Figure 5. Architecture of detection system

IV. METHODOLOGY

4.1 Permissions Selection

Before a permission request is to be removed, it must first be selected to be removed. When selecting a permission to remove or block, it must not affect the major functions of an app. For example, social networking apps require Internet access in order to function; as such the “INTERNET” permission is required. Testing an app without Internet access can be done simply by disabling all Internet connections. The aim, therefore, is to remove dangerous permissions from an app that should not be required. As such, the permissions that are most commonly requested by apps but also not necessarily required are considered for removal.

The AndroidManifest.xml file obtained can then be read with any plain-text editor. Figure 6 outlines our proposed app permissions selection process. The first step is for the user to determine whether the app requires this permission. The second step determines whether the app actually requires this permission in order to function. For example, a mapping app will require location resources such as the GPS system in order to function. A note keeping app, on the other hand, has no obvious need for such information. The next two steps will determine whether the permission is harmless and feasible to be removed from the app. For example, many app permissions allow an app to access sensitive information such as contact information, phone logs, IMEI numbers, and SMS. A user may choose to expressly disallow a particular permission even when the app has well defined justifications. The feasibility of removing an app’s permission is considered. Some apps may be so heavily integrated with a certain resource that it may not run without it.

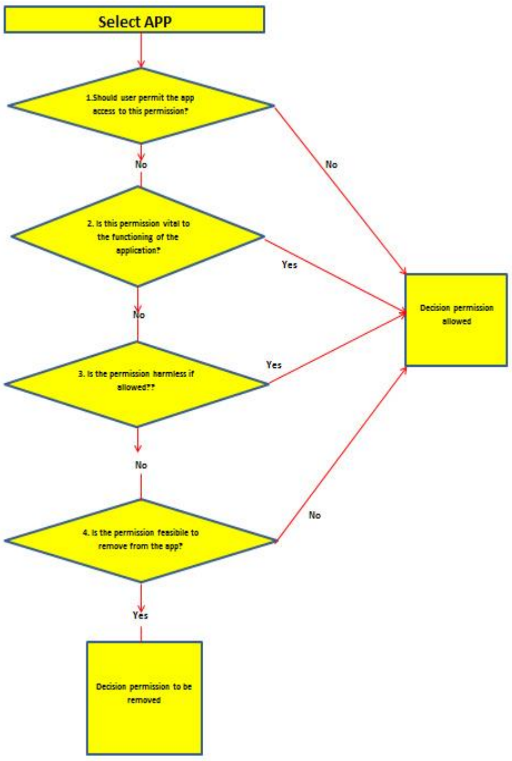

Select APP
1.Should user permit the app access to this permission?
No
No
2. Is this permission vital to the functioning of the application?
Yes
No
3. Is the permission harmless if allowed??
Yes
No
4. Is the permission feasibile to remove from the app?
No
Yes
Decision permission allowed
Decision permission to be removed

Figure 6. Permissions selection process

4.2 Permissions Removal

Permissions removal is used in order to improve user privacy on Android devices. Permissions removal is the process wherein an app’s package installer is reverse engineered to removeunnecessary or privacy-intruding permissions. The benefit of this method is that the app can be installed on any version of Android that supports the unmodified app. This means no additional third party software or rooted/custom Android OS is required which may have been an additional privacy/security risk. A major downside to this method is the time required to properly remove one or more permissions and address dependencies within the app. It may not be possible to fully remove an Android permission’s dependencies as the app’s coded functionality may be too tightly integrated. For example, removing both coarse and fine locations from a turn-by-turn navigation app would not be useful or even viable due to the nature of the app. Another challenge with this method is that due to the digital signature verification in Android - the modified app is not signed with the original key and hence cannot be updated over the official version of the app installed on the device.
This means a completely new installation of the app is required in order for this app to be updated on the (one) device.

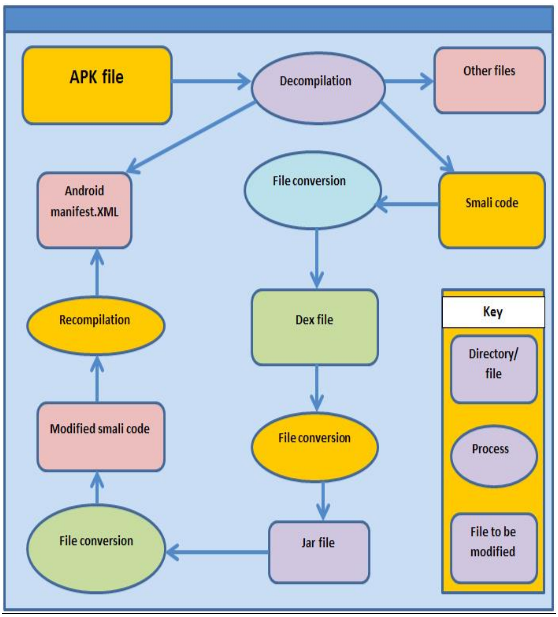


Figure 7. Ideal permissions removal process

Figure 7 shows the ideal method of manual permissions removal to be performed on an Android app. The reason this method is considered ideal is that this process results in the entire app's source code being readable and modifiable in Java. An app is first decompiled using a decompilation tool – in our case, APK Multi Tool is used [7]. Decompilation results in several files, as shown, with importance placed on the "smali" code files and AndroidManifest.xml file.

The smali code files are the source code of the particular Android app in a human readable format. The problem with this format is that it is difficult to read and debug apps; the language is complicated and hard to understand. As this is the case, the smali code files are then converted to a single .dex or Dalvik Executable file using a tool called smali/baksmali [8]. This results in a .jar file, simply a Java archive file containing Java classes which can be read and extracted to .java files using JD-GUI [9]. At this point, changes to the app can be easily made by modifying its Java files. The plain text AndroidManifest.xml file can now be read and modified using any plain texteditor. Removing the highlighted row in Figure 2 would effectively render the app unable to read contacts data from the Android device, but may render the app unusable due to instability issues.

Due to this, sourcecode changes must be made in order to result in a usable app that cannot access contacts data. After the source code changes are made, the app must be converted back into smali code in order for the recompilation process to be successful. The smali/baksmali software package is used once again to convert the Java code to smali code. APK Multi Tool is then used to recompile and sign the repackaged app. The result should be a working app installation package with some resource access removed, thus improving user privacy.

V. CONCLUSION

A detection system combined with PC and cell phone side is proposed and demonstrated based on the analysis of Android security mechanisms and the study of the potential safety problems caused by the inherent defects on the Android platform, such as the system rough permissions prompt interface and the uneven ability of publisher of APP., The detection system can thoroughly detect permissions information's of the installed APPs and uninstalled APKs in advance. The monitoring system can not only give understandable explanation in detail, but also provide users with the function of screening on APPs with certain sensitive permission. In addition, the system can also provide users secondary warning and secondary judgement opportunities, and help consumers improve their safety consciousness and protect their privacy from unknown infringement. In the next, we are going to take the overriding of Kernel of Linux into consideration to restrain the APPS from applying for more sensitive permissions, such as preventing certain APPs connecting to the Internet or sending short messages, so that we can eradicate the malicious software completely and guarantee the users' property and privacy.This leads onto a further future work or research that could be undertaken where this automated system would be implemented onto an Android device. This would result in a self-functioning system that could enhance the privacy of apps on the device.

REFERENCES

1) O. Hou, 2012, "A Look at Google Bouncer", http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/, accessed 14 April 2013
2) Y. Zhou, X. Zhang, X. Jiang & V. Freeh, "Taming information-stealing smartphone applications (on Android)", TRUST 2011, pp. 93-107.
3) S. Bugiel, S. Heuser & AR. Sadeghi, myTunes: Semantically Linked and User-Centric Fine-Grained Privacy Control on Android, Technical Report TUD-CS-2012-0226, Center for Advanced Security Research Darmstadt, 2012.
4) A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev&CGlezer, "GoogleAndroid: A Comprehensive Security Assessment", IEEE Security & Privacy Magazine, vol. 8, no. 2, 2012, pp. 35-44.
5) http://developer.android.com/reference/android/ Manifest.permission.html, accessed 14 June 2013
6) http://developer.android.com/guide/topics/manifest /permission-element.html, accessed 13 June 2013
7) http://apkmultitool.com/, accessed 11 April 2013
8) http://code.google.com/p/smali/, accessed 11 April 2013
9) http://java.decompiler.free.fr/?q=jdgui, accessed 11 April 2013
10) Android.Developers.http://developer.android.corn/guide/basics/what-is android.html[EB/OL].March 2011.
11) Sinascience and technology http://tech.sina.com.cn/it/ 2013-05-10/07478325514.shtml[EB/OL].May 2013.
12) Google. Android Reference: Security and Permissions.http://developer.android.com/guide/topics/security/securit y. Html [EB/OL].
13) Minghua Liao, Liming Zheng.The Security Mechanism Analysis and Probe into the Solution of Androids[M]. Science Technology and Engineering Vol.26, September 2011, pp.6350-6355.