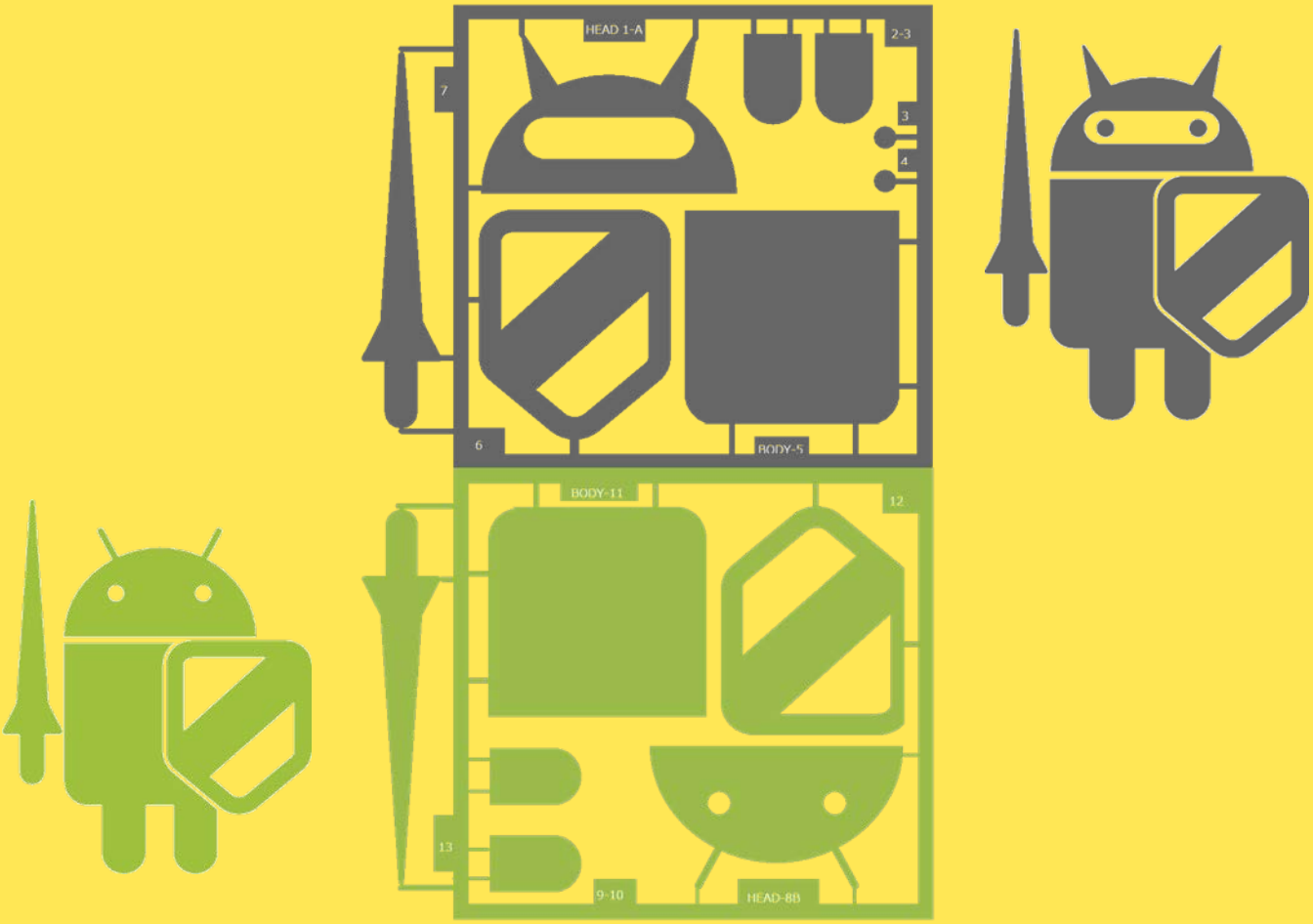
JAPAN SMARTPHONE SECURITY ASSOCIATION

Android Application Secure Design/Secure Coding Guidebook

- Secure communication! -



April 1st, 2014 version

Japan Smartphone Security Association (JSSEC)

Secure Coding Group

Document control number: JSSEC-TECA-SC-GD20140401BE

- The content of this guide is up to date as of the time of publication, but standards and environments are constantly evolving. When using sample code, make sure you are adhering to the latest coding standards and best practices.
- JSSEC and the writers of this guide are not responsible for how you use this document. Full responsibility lies with you, the user of the information provided.
- Android™ is a trademark or a registered trademark of Google Inc.

  The company names, product names and service names appearing in this document are generally the registered trademarks or trademarks of their respective companies.

  Further, the registered trademark ®, trademark (TM) and copyright © symbols are not used throughout this document.
- Parts of this document are copied from or based on content created and provided by Google, Inc. They are used here in accordance with the provisions of the Creative Commons Attribution 3.0 License

Android Application Secure Design/Secure Coding Guidebook

- Beta version -

April 1st, 2014

Japan Smartphone Security Association

Secure Coding Group

Index

Revision history

Date	Revised contents
2014-4-01	• Initial English version
• New versions of the guidebook updated based on public opinions and comments.	

- Published by -
Japan Smartphone Security Association
Secure Coding Group, Application Working Group, Smartphone Technology Committee

Leader	Masaru Matsunami	Sony Digital Network Applications, Inc.
Member	Tomoyuki Hasegawa	Android Security Japan
	Mayumi Nishiyama	BJIT Inc.
	Tohru Ohzono	Cisco Systems, Inc.
	Masaki Kubo	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
	Daniel Burrowes	Kobe Digital Labo Inc.
	Zachary Mathis	Kobe Digital Labo Inc.
	Renta Futamura	NextGen, Inc.
	Naonobu Yatsukawa	Nihon Unisys, Ltd.
	Shigenori Takei	NTT Software Corporation
	Ikuya Fukumoto	Software Research Associates, Inc.
	Tsutomu Kumazawa	Software Research Associates, Inc.
	Akira Ando	Sony Digital Network Applications, Inc.
	Hiroko Nakajima	Sony Digital Network Applications, Inc.
	Ken Okuyama	Sony Digital Network Applications, Inc.
	Satoshi Fujimura	Sony Digital Network Applications, Inc.
	Setsuko Kaji	Sony Digital Network Applications, Inc.
	Taeko Ito	Sony Digital Network Applications, Inc.
	Yoshinori Kataoka	Sony Digital Network Applications, Inc.
	Hidenori Yamaji	Sony Mobile Communications Inc.
	Takuya Nishibayashi	Sony Mobile Communications Inc.
	Koji Isoda	Symantec Japan, Inc.
	Gaku Taniguchi	Tao Software, Inc.
	Michiyoshi Sato	Tokyo System House Co., Ltd.

(In no particular order)

- Authors of April 1, 2013 Japanese Edition -

Leader	Masaru Matsunami	Sony Digital Network Applications, Inc.
Member	Masaomi Adachi	Android Security Japan
	Tomoyuki Hasegawa	Android Security Japan
	Yuki Abe	Software Research Associates, Inc.
	Tomomi Oouchi	Software Research Associates, Inc.
	Tsutomu Kumazawa	Software Research Associates, Inc.
	Toshimi Sawada	Software Research Associates, Inc.
	Kiyoshi Hata	Software Research Associates, Inc.
	Youichi Higa	Software Research Associates, Inc.
	Yuu Fukui	Software Research Associates, Inc.
	Ikuya Fukumoto	Software Research Associates, Inc.
	Eiji Hoshimoto	Software Research Associates, Inc.
	Shun Yokoi	Software Research Associates, Inc.
	Takakazu Yoshizawa	Software Research Associates, Inc.
	Takeshi Fujiwara	NRI SecureTechnologies, Ltd.
	Shigenori Takei	NTT Software Corporation
	Masaki Kubo	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
	Hiroshi Kumagai	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
	Yozo Toda	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
	Tohru Ohzono	Cisco Systems, Inc.
	Shigeru Yatabe	Cisco Systems, Inc.
	Mikiya Arai	Secure Sky Technology, Inc.
	Masahiko Sakamoto	Secure Sky Technology, Inc.
	Toru Asano	Sony Digital Network Applications, Inc.
	Akira Ando	Sony Digital Network Applications, Inc.
	Ryohji Ikebe	Sony Digital Network Applications, Inc.
	Jun Ogiso	Sony Digital Network Applications, Inc.
	Ken Okuyama	Sony Digital Network Applications, Inc.
	Yoshinori Kataoka	Sony Digital Network Applications, Inc.
	Muneaki Nishimura	Sony Digital Network Applications, Inc.
	Koji Furusawa	Sony Digital Network Applications, Inc.
	Kenji Yamaoka	Sony Digital Network Applications, Inc.
	Gaku Taniguchi	Tao Software, Inc.
	Naonobu Yatsukawa	Nihon Unisys, Ltd.

(In no particular order)

- Authors of November 1, 2012 Japanese Edition -

Leader

Masaru Matsunami	Sony Digital Network Applications, Inc.

Member

Katsuhiko Sato, Nakaguchi Akihiko	Android Security Japan
Tomomi Oouchi, Naoyuki Ohira, Tsutomu Kumazawa, Miki Sekikawa, Seigo Nakano, Youichi Higa, Ikuya Fukumoto, Eiji Hoshimoto, Shoichi Yasuda, Tadayuki Yahiro, Takakazu Yoshizawa	Software Research Associates, Inc.
Shigenori Takei	NTT Software Corporation
Keisuke Takemori	KDDI CORPORATION
Masaki Kubo, Hiroshi Kumagai, Yozo Toda	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
Tohru Ohzono, Shigeru Yatabe	Cisco Systems, Inc.
Toru Asano, Akira Ando, Ryohji Ikebe, Shigeru Ichikawa, Mitake Ohtani, Jun Ogiso, Ken Okuyama, Yoshinori Kataoka, Ikue Sato, Muneaki Nishimura, Kazuo Yamaoka, Takeru Kikkawa	Sony Digital Network Applications, Inc.
Gaku Taniguchi, Eiji Shimano, Hisao Kitamura	Tao Software, Inc.
Takao Yamakawa	Japan Online Game Association
Masaki Ishihara, Yasuaki Mori	Nippon System Kaihatsu Co., Ltd.
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigeki Fujii	UNIADEX, Ltd.

(In no particular order)

- Authors of June 1, 2012 Japanese Edition-

Leader

Masaru Matsunami	Sony Digital Network Applications, Inc.

Member

Katsuhiko Sato	Android Security Japan
Tomomi Oouchi, Youichi Higa, Eiji Hoshimoto	Software Research Associates, Inc.
Shigenori Takei	NTT Software Corporation
Masaki Kubo, Hiroshi Kumagai, Yozo Toda	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
Tohru Ohzono, Shigeru Yatabe	Cisco Systems, Inc.
Yoichi Taguchi	System House. ING Co., Ltd.
Masahiko Sakamoto	Secure Sky Technology, Inc.
Akira Ando, Shigeru Ichikawa, Ken Okuyama, Ikue Sato, Muneaki Nishimura, Kazuo Yamaoka	Sony Digital Network Applications, Inc.
Hidehira Kuranaga	Daiwa Institute of Research Holdings Ltd.
Gaku Taniguchi, Eiji Shimano, Hisao Kitamura	Tao Software, Inc.
Michiyoshi Sato	Tokyo System House Co., Ltd.
Masakazu Hattori	Trend Micro Incorporated.
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Masaaki Chida	NetAgent Inc.
Shigeki Fujii	UNIADEX, Ltd.

(In no particular order)

1. Introduction

1.1. Building a Secure Smartphone Society

This guidebook is a collection of tips concerning the know-how of secure designs and secure coding for Android application developers. Our intent is to have as many Android application developers as possible take advantage of this, and for that reason we are making it public.

In recent years, the smartphone market has witnessed a rapid expansion, and its momentum seems unstoppable. Its accelerated growth is brought on due to the diverse range of applications. An unspecified large number of key functions of mobile phones that were once not accessible due to security restrictions on conventional mobile phones have been made open to smartphone applications. Subsequently, the availability of varied applications that were once closed to conventional mobile phones is what makes smartphones more attractive.

With great power that comes from smartphone applications comes great responsibility from their developers. The default security restrictions on conventional mobile phones had made it possible to maintain a relative level of security even for applications that were developed without security awareness. As it has been aforementioned with regard to smartphones, since the key advantage of a smartphone is that they are open to application developers, if the developers design or code their applications without the knowledge of security issues then this could lead to risks of users' personal information leakage or exploitation by malware causing financial damage such as from illicit calls to premium-rate numbers.

Due to Android being a very open model allowing access to many functions on the smartphone, it is believed that Android application developers need to take more care about security issues than iOS application developers. In addition, responsibility for application security is almost solely left to the application developers. For example, applications can be released to the public without any screening from a marketplace such as Google Play (former Android Market), though this is not possible for iOS applications.

In conjunction with the rapid growth of the smartphone market, there has been a sudden influx of software engineers from different areas in the smartphone application development market. As a result, there is an urgent call for the sharing knowledge of secure design and consolidation of secure coding know-how for specific security issues related to mobile applications.

Due to these circumstances, Japan's Smartphone Security Association (JSSEC) has launched the Secure Coding Group, and by collecting the know-how of secure design as well as secure coding of Android applications, it has decided to make all of the information public with this guidebook. It is our intention to raise the security level of many of the Android applications that are released in the market by having many Android application developers become acquainted with the know-how of secure design and coding. As a result, we believe we will be contributing to the creation of a more reliable and safe smartphone society.

1.2. Timely Feedback on a Regular Basis Through the Beta Version

We, the JSSEC Secure Coding Group, will do our best to keep the content contained in the Guidebook as accurate as possible, but we cannot make any guarantees. We believe it is our priority to publicize and share the know-how in a timely fashion. Equally, we will upload and publicize what we consider to be the latest and most accurate correct information at that particular juncture, and will update it with more accurate information once we receive any feedback or corrections. In other words, we are taking the beta version approach on a regular basis. We think this approach would be meaningful for many of the Android application developers who are planning on using the Guidebook.

The latest version of the Guidebook and sample codes can be obtained from the URL below.

- http://www.jssec.org/dl/android_securecoding_en.pdf Guidebook (English)
- http://www.jssec.org/dl/android_securecoding_en.zip Sample Codes (English)

The latest Japanese version can be obtained from the URL below.

- http://www.jssec.org/dl/android_securecoding.pdf Guidebook (Japanese)
- http://www.jssec.org/dl/android_securecoding.zip Sample Codes (Japanese)

1.3. Usage Agreement of the Guidebook

We need your consent for the following two precautionary statements when using the Guidebook.

1. The information contained in the Guidebook may be inaccurate. Please use the information written here by your own discretion.
2. In case of finding any mistakes contained in the Guidebook, please send us an e-mail to the address listed below. However, we cannot guarantee a reply or any revisions thereof.

Japan Smartphone Security Association
E-mail: sec@jssec.org
Subject: [Comment] Android Secure Coding Guidebook 20130401EN
Content: Name (optional), Affiliation (optional), E-mail (optional), Comment (required) and Other matters (optional)

2. Composition of the Guidebook

2.1. Developer's Context

Many guidebooks that have been written on secure coding include warnings about harmful coding practices and their suggested revisions. Although this approach can be useful at the time of reviewing the source code that has already been coded, it can be confusing for developers that are about to start coding, as they do not know which article to refer to.

The Guidebook has focused on the developer's context of "What is a developer trying to do at this moment?" Equally, we have taken steps to prepare articles that are aligned with the developer's context. For example, we have divided articles into project units by presuming that a developer will be involved in operations such as [Creating/Using Activities], [Using SQLite], etc.

We believe that by publishing articles that support the developer's context, developers will be able to easily locate necessary articles that will be instantly useful in their projects.

2.2. Sample Code, Rule Book, Advanced Topics

Each article is comprised of three sections: Sample Code, Rule Book, and Advanced Topics. If you are in a hurry, please look up the Sample Code and Rule Book sections. The content is provided in a way where it can be reused to a certain degree. For those who have issues that go beyond these, please refer the Advanced Topics section. We have given descriptions that will be helpful in finding solutions for individual cases.

Unless it is specifically noted, our focus of development will be targeted to platforms concerning Android 2.2 (API Level 8) and later. Since we have not verified the operational capability of any versions pertaining to Android 2.1 (API Level 7) or prior, the measures described may prove ineffective on these older systems. In addition, even for versions that are covered under the scope of focus, it is important to verify their operational capability by testing them on your own environment before releasing them publically.

2.2.1. Sample Code

Sample code that serves as the basic model within the developer's context and functions as the theme of an article is published in the Sample Code section. If there are multiple patterns, we have provided source code for the different patterns and classified them accordingly. We have strived to make our commentaries as simple as possible. For example, when we want to direct the reader's attention to a security issue that requires attention, a bullet-point number will appear next to "**Point**" in the article. We will also comment on the sample code that corresponds to the bullet-point number by writing "**`*** Point (Number) ***`**." Please note that a single point may correspond to multiple pieces of sample code. There are sections throughout the entire source code, albeit very little compared to the entire code, that requires our attention for security. In order to be able to survey the sections that call for scrutiny, we try to post the entire class unit of sample code.

Please note that only a portion of sample code is posted in the Guidebook. A compressed file, which contains the entire sample code, is made public in the URL listed below. It is made public by the Apache License, Version 2.0; therefore, please feel free to copy and paste it. Please note that we have minimized the code for error processing in the sample code to prevent it from becoming too long.

- http://www.jssec.org/dl/android_securecoding_en.zip Sample Codes Archive

The projects/keystore file that is attached in the sample code is the keystore file that contains the developer key for the signature of the APK. The password is "android." Please use it when singing the APK in the In-house sample code.

We have provided the keystore file, debug.keystore, for debugging purposes. When using Eclipse for development, it is convenient for verifying the operational capability of the In-house sample code if a file path is set in the In-house defined debug keystore of [Android]-[Build] of [Preferences] in advance. In addition, for sample code that is comprised of multiple APKs, it is necessary to match the android:debuggable setting contained inside each AndroidManifest.xml in order to verify the

cooperation between each APK. If the android:debuggable setting is not explicit set when installing the APK from Eclipse, it will automatically become android:debuggable= "true."

For embedding the sample code as well as keystore file into Eclipse, please refer to "2.5 Steps to Install Sample Codes into Eclipse."

2.2.2. Rule Book

Rules and matters that need to be considered regarding security within the developer's context will be published in the Rule Book section. Rules to be handled in that section will be listed in a table format at the beginning and will be divided into two levels: "Required" and "Recommended." The rules will consist of two types of affirmative and negative statements. For example, an affirmative statement that expresses that a rule is required will say "Required." An affirmative statement that expresses a recommendation will say "Recommended." For a negative statement that expresses the requisite nature of the rule would say, "Definitely not do." For a negative sentence that expresses a recommendation would say, "Not recommended." Since these differentiations of levels are based on the subjective viewpoint of the author, it should only be used as a point of reference.

Sample code that is posted in the Sample Code section reflect these rules and matters that need to be considered, and a detailed explanation on them is available in the Rule Book section. Furthermore, rules and matters that need to be considered that are not dealt with in the Sample Code section are handled in the Rule Book section.

2.2.3. Advanced Topics

Items that require our attention, but that could not be covered in the Sample Code and Rule Book sections within the developer's context will be published in the Advanced Topics section. The Advanced Topics section can be utilized to explore ways to solve separate issues that could not be solved in the Sample Code or Rule Book sections. For example, subject matters that contain personal opinions as well as topics on the limitations of Android OS in relation the developer's context will be covered in the Advanced Topics section.

Developers are always busy. Many developers are asked to have basic knowledge of security and produce as many Android applications as quickly as possible in a somewhat safe manner than to really understand the deep security matters. However, there are certain applications out there that require a high level of security design and implementation from the beginning. For developers of such applications, it is necessary for them to have a deep understanding concerning the security of Android OS.

In order to benefit both developers who emphasize development speed and also those who emphasize security, all articles of the Guidebook are divided into the three sections of Sample Code, Rule Book, and Advanced Topics. The aim of the Sample Code and Rule Book sections is to provide generalizations about security that anyone can benefit from and source code that will work with a minimal amount of customization and hopefully by just copying and pasting. In the Advanced Topics section, we offer materials that will help developers think in a certain way when they are facing

specific problems. It is the aim of the Advanced Topics section to help developers examine optimal secure design and coding when they are involved in building individual applications.

2.3. The Scope of the Guidebook

The purpose of the Guidebook is to collect security best practices that are necessary for general Android application developers. Consequently, our scope is focused mainly on security tips (The "Application Security" section in figure below) for the development of Android applications that are distributed primarily in a public market.

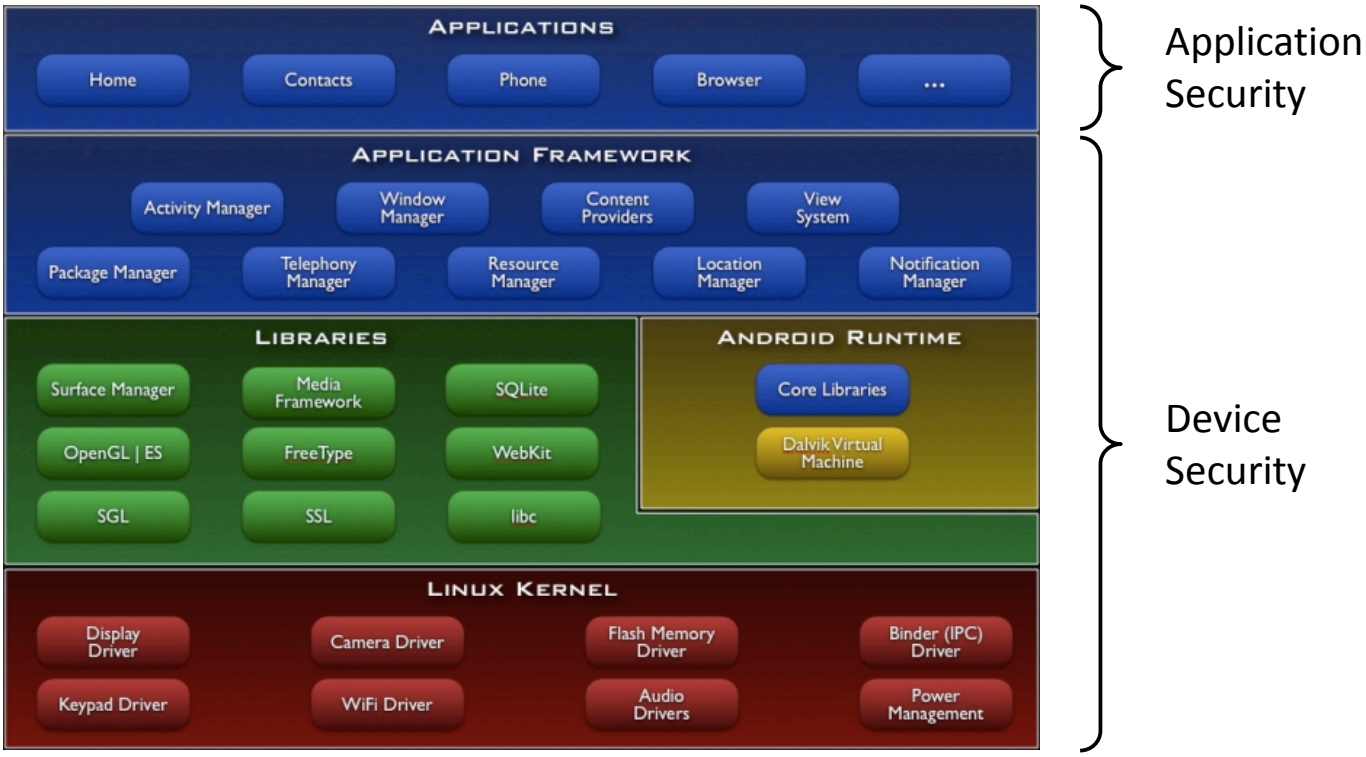


Figure 2.3-1

Security regarding the implementation of components in the Device Security of the above figure is outside the scope of this guidebook. There are differences in the viewpoint of security between general applications that are installed by users and pre-installed applications by device manufacturers. The Guidebook only handles the former and does not deal with the latter. In the current version, tips only on the implementation by Java are posted, but in future versions, we plan on posting tips on JNI implementations as well.

Also as of now we do not handle threats that results from an attacker obtaining root privileges. We will assume the premise of a secure Android device in which it is not possible to obtain root privileges and base our security advice on utilizing the Android OS security model. For handling of assets and threats, we have provided a detailed description on "3.1.3 Asset Classification and Protective Countermeasures."

2.4. Literature on Android Secure Coding

Since we are not able to discuss all of Android's secure coding in the Guidebook, we recommend that you read the literature mentioned below in conjunction with the Guidebook.

- *Android Security: Anzenna Application Wo Sakusei Surutameni (Secured Programming in Android)*
  Author: Taosoftware Co., Ltd. ISBN: 978-4-8443-3134-6
  http://www.amazon.co.jp/dp/4844331345/
- *The CERT Oracle Secure Coding Standard for Java*
  Authors: Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
  http://www.amazon.com/dp/0321803957

2.5. Steps to Install Sample Codes into Eclipse

This section explains how to install sample code into Eclipse. Sample code is divided into multiple projects depending on the purpose. Installing all of the sample code at once is described in, "2.5.1 Installing the Sample Project" and how to selectively install the sample code is described in, "2.5.2 Install by Selecting Individual Projects of the Sample." After the installation is completed, please refer to "2.5.3 Setup the Sample Code Validator debug.keystore" and install the debug.keystore file into Eclipse. We have verified the following steps in the following environment:

- OS
  - Windows 7 Ultimate SP1
- Android Developer Tool
  - Build: v22.3.0-887826
- Android SDK
  - Android 2.2(API 8)
    - For any sample projects that do not require any attention can be built through Android 2.2 (API 8). However, depending on the sample, some may require SDK Platform that comes later than Android 2.3.3 (API 10).

2.5.1. Installing the Sample Project

1. Download the sample code.
   Acquire the sample code from the URL shown in "2.2.1 Sample Code"

2. Extract the sample code.
   Right click on the sample code that has been compressed into zip file, and click on "Extract All" as shown below.

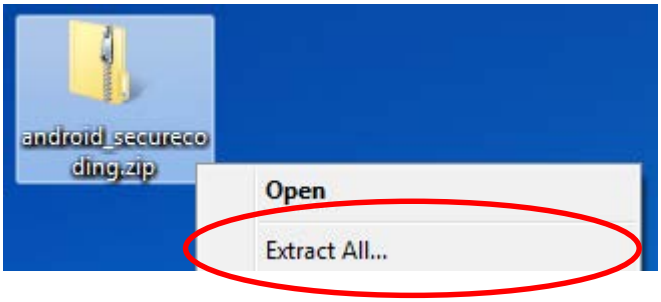


Figure 2.5-1

3. Designate where to deploy.
   Create a workspace under the name "C:¥android_securecoding" by designating "C:¥" and clicking on the "Extract" button.

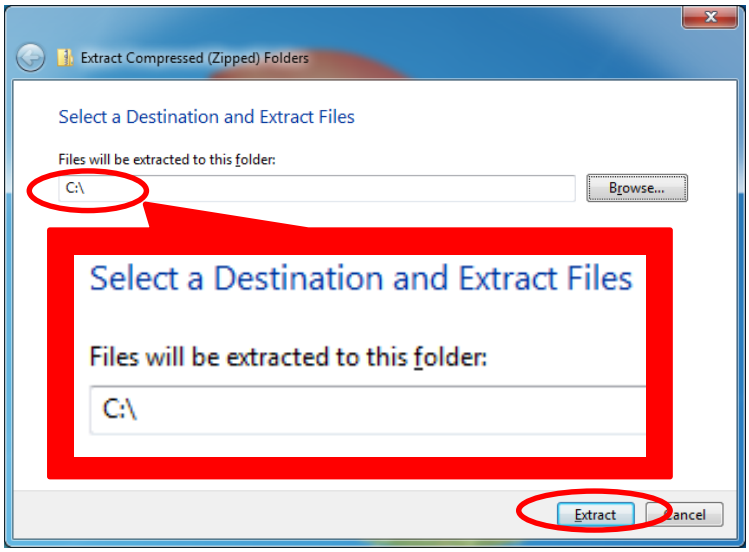


Figure 2.5-2

After clicking on the "Extract" button, right underneath "C:¥" a folder called "android_securecoding" will be created.

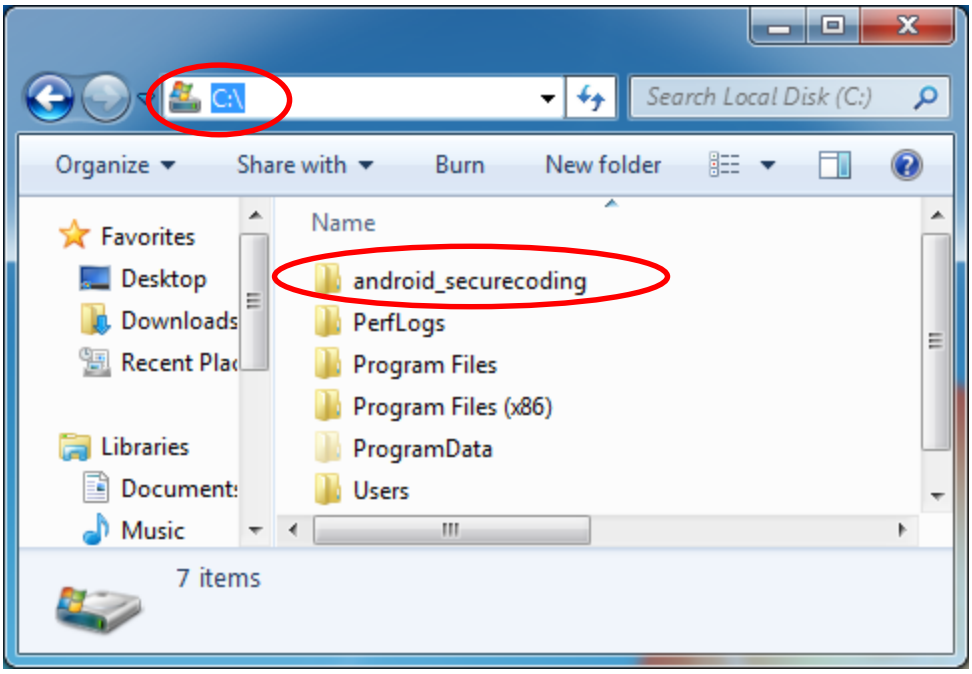


Figure 2.5-3

The sample code is contained in the"android_securecoding" folder. For example, when you want to refer to the sample code within "4.1.1.3 Creating/Using Partner Activities" of "4.1 Creating/Using Activities" please look below.

android_securecoding
└Create Use Activity
└Activity PartnerActivity

In this way, the sample code project will be located under the chapter title in the "android_securecoding" folder.

4. Designate workspace by starting up Eclipse
   Launch Eclipse from the start menu or from a desktop icon. From the displayed selected dialogue, designate "C:¥android_securecoding" the workspace that was deployed in the previous step. If the selected dialogue is not displayed, click on "File->Switch Workspace->Other..." from the menu.

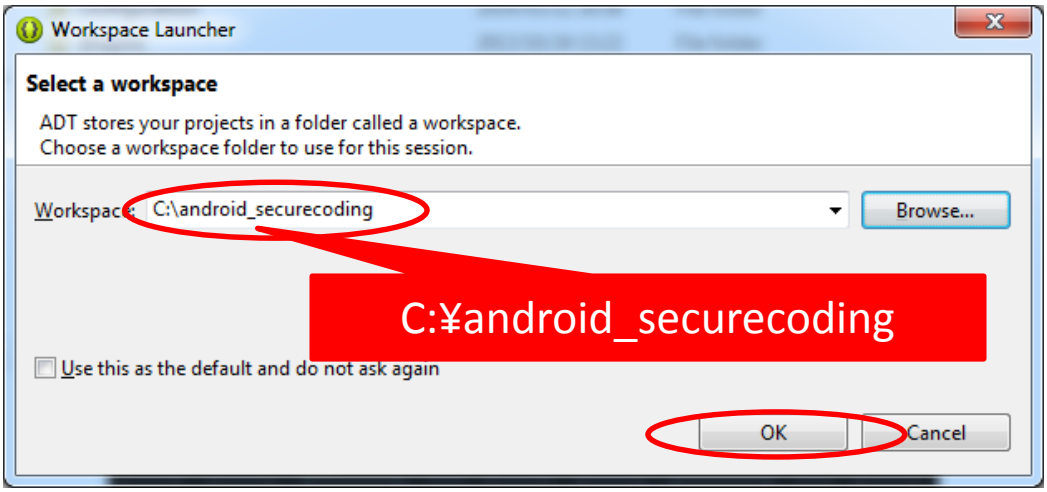


Figure 2.5-4

5. Display the workbench
   Close the Android IDE pane after Eclipse has been launched.

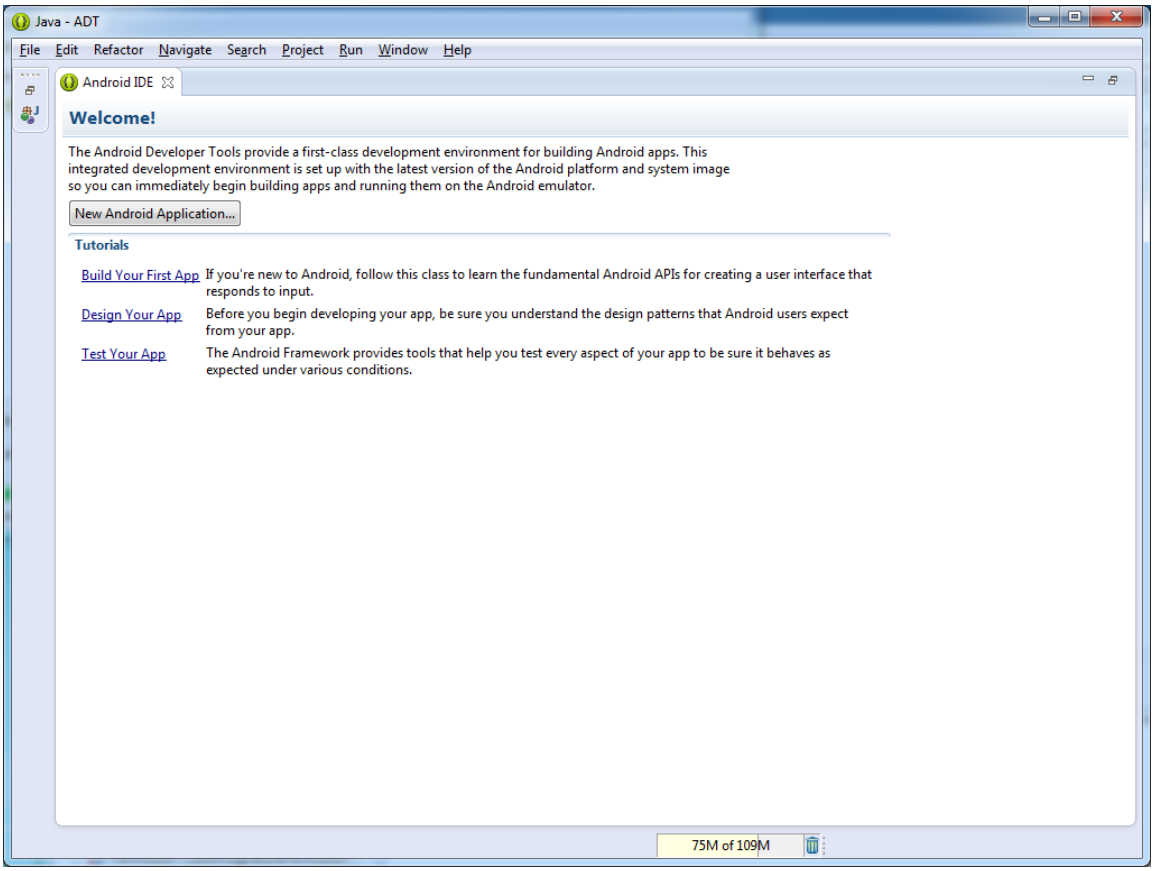


Figure 2.5-5

6. Start importing
   Right click on the package explorer and click "Import".

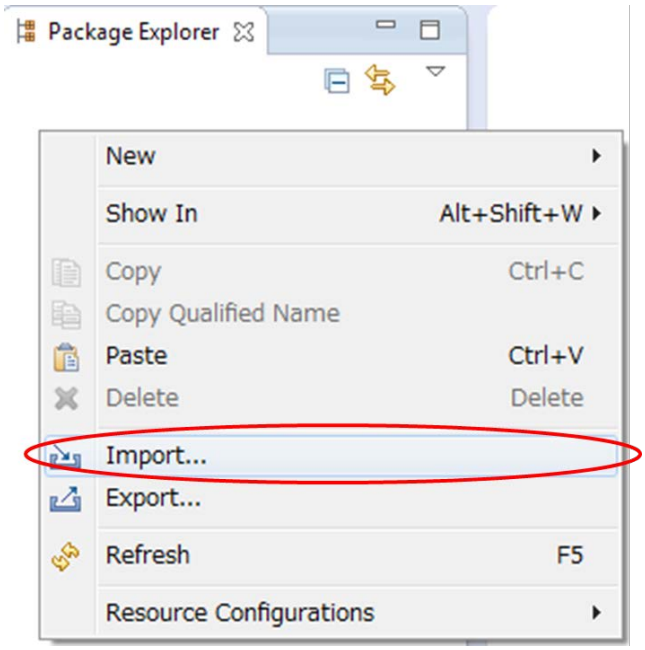


Figure 2.5-6

7. Import existing projects into workspace
   Click on "Existing Projects into Workspace" and then click "Next"

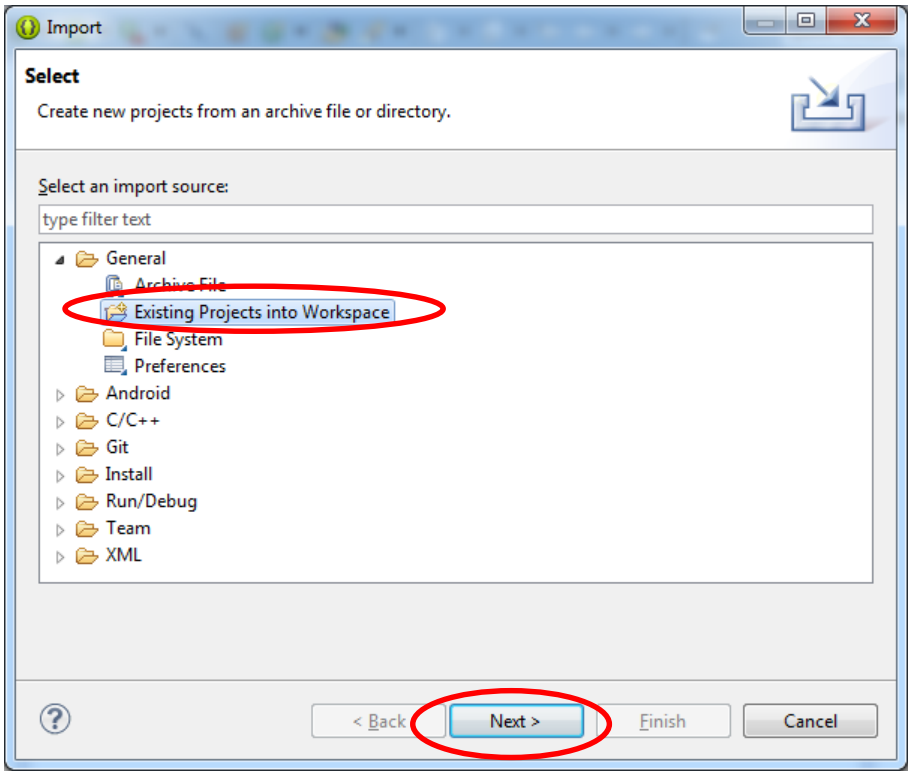


Figure 2.5-7

8. Select the project
   Click on "Browse" next to the "Select root directory"

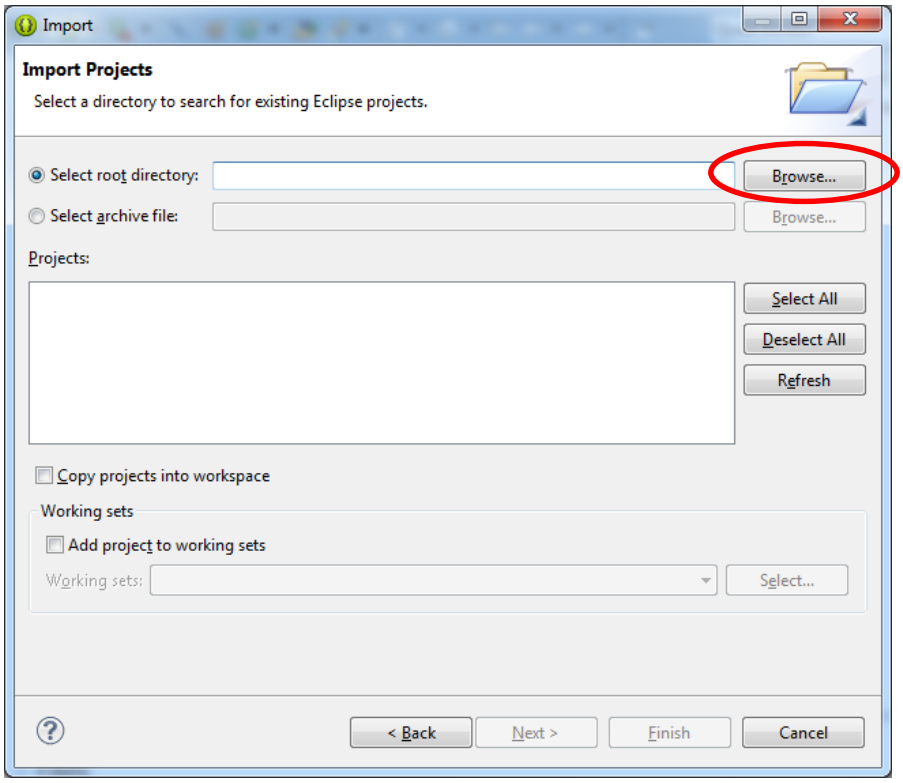


Figure 2.5-8

9. Select the extracted folder "android_securecoding"

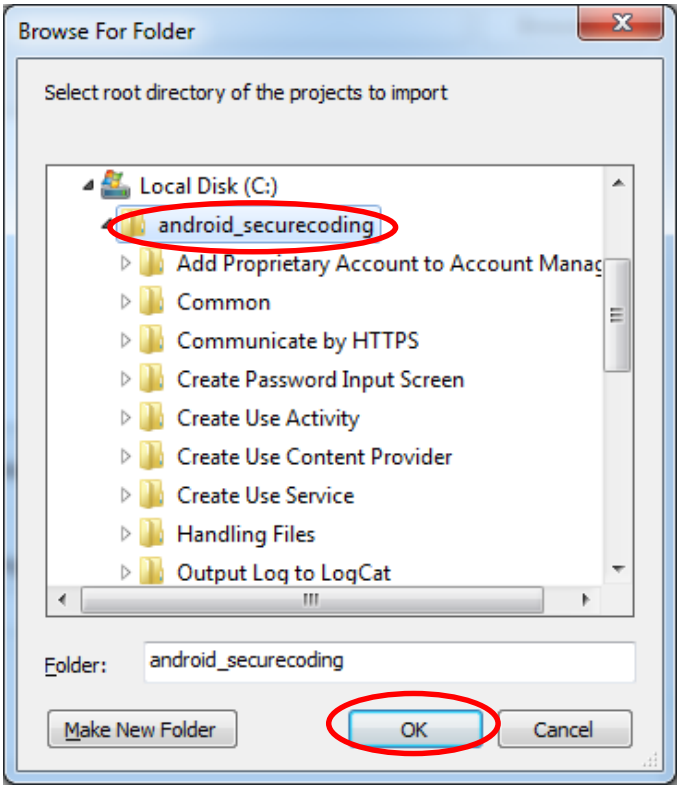


Figure 2.5-9

10. Finish importing

    Click "Finish" to import all of the projects. If "Copy project into workspace" is checked, the folder hierarchy of each project will become flat. Please be aware that some projects may refer to the "Shared/JSSEC Shared" folder which can lead to a compilation error if the folder hierarchy is changed. In the case of installing only a portion of a project, please refer to, "2.5.2 Install by Selecting Individual Projects of the Sample."

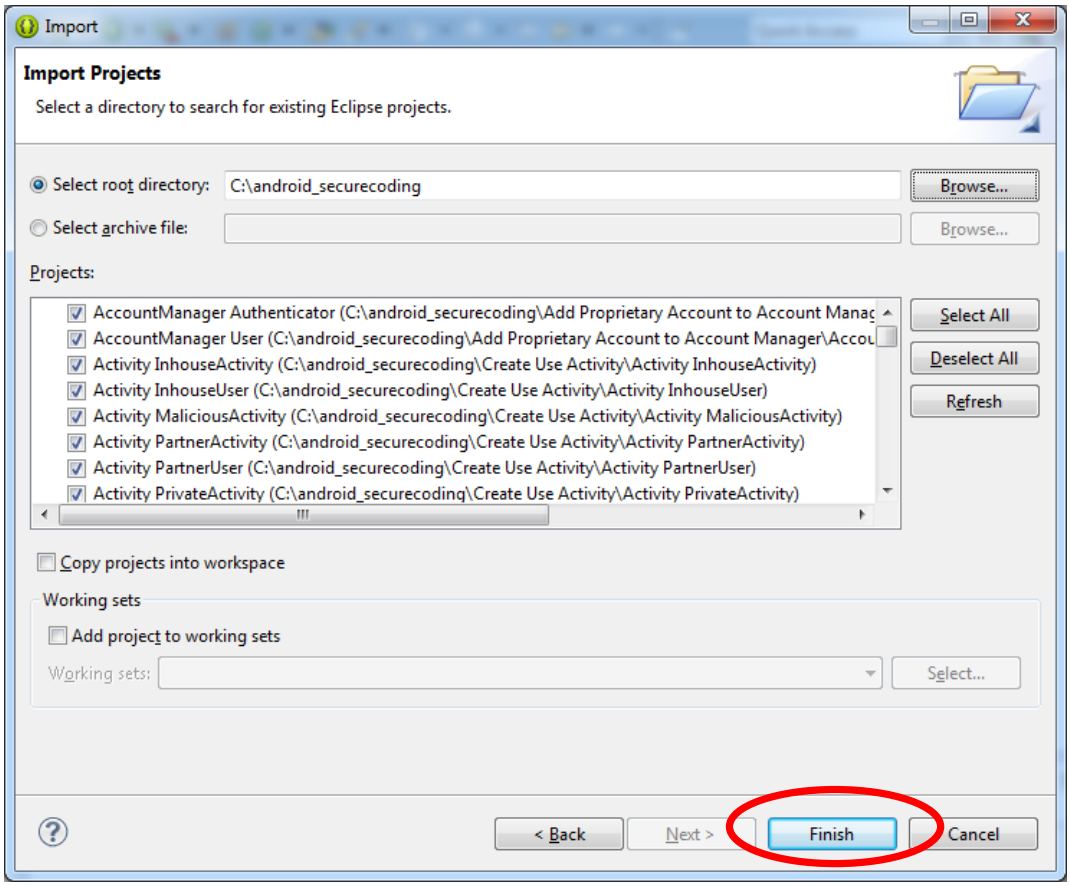


Figure 2.5-10

11. All projects are imported

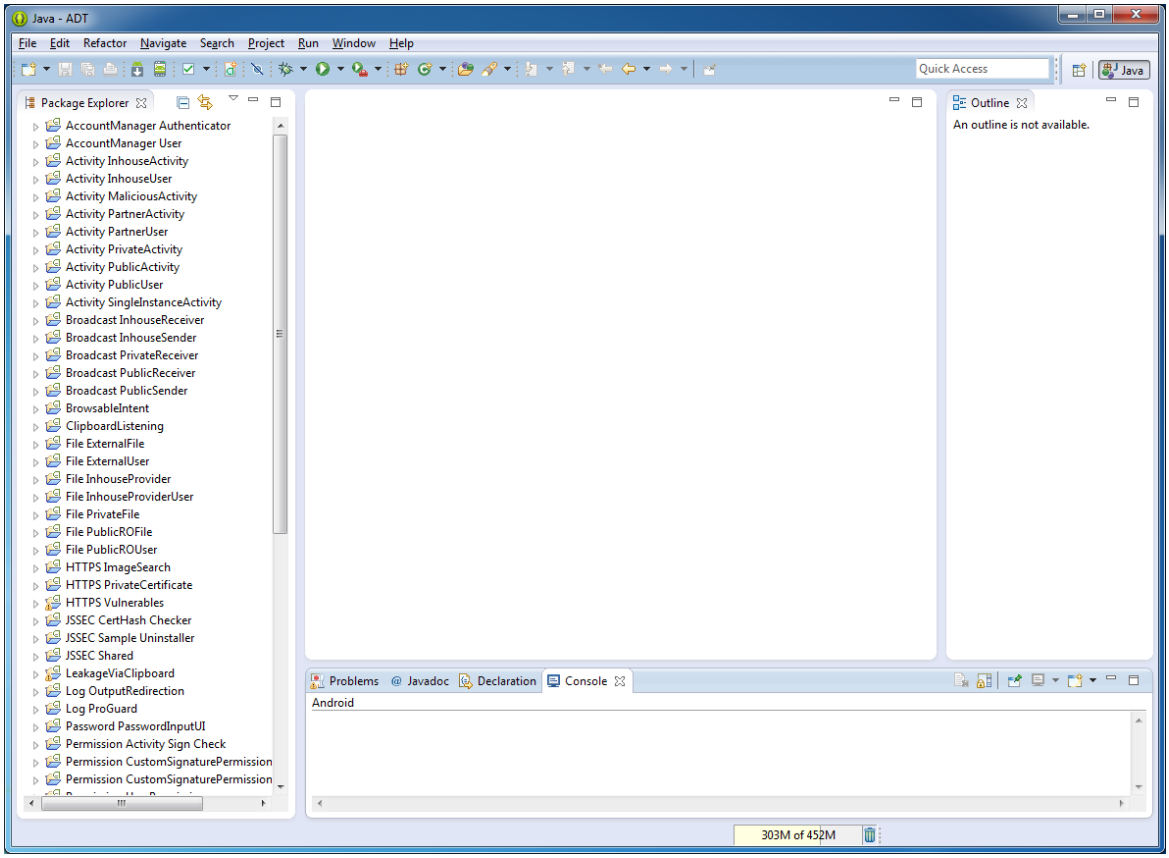


Figure 2.5-11

2.5.2. Install by Selecting Individual Projects of the Sample Project

Please refer to "2.5.1 Installing the Sample Project" and follow the steps until "10. Finishing importing".

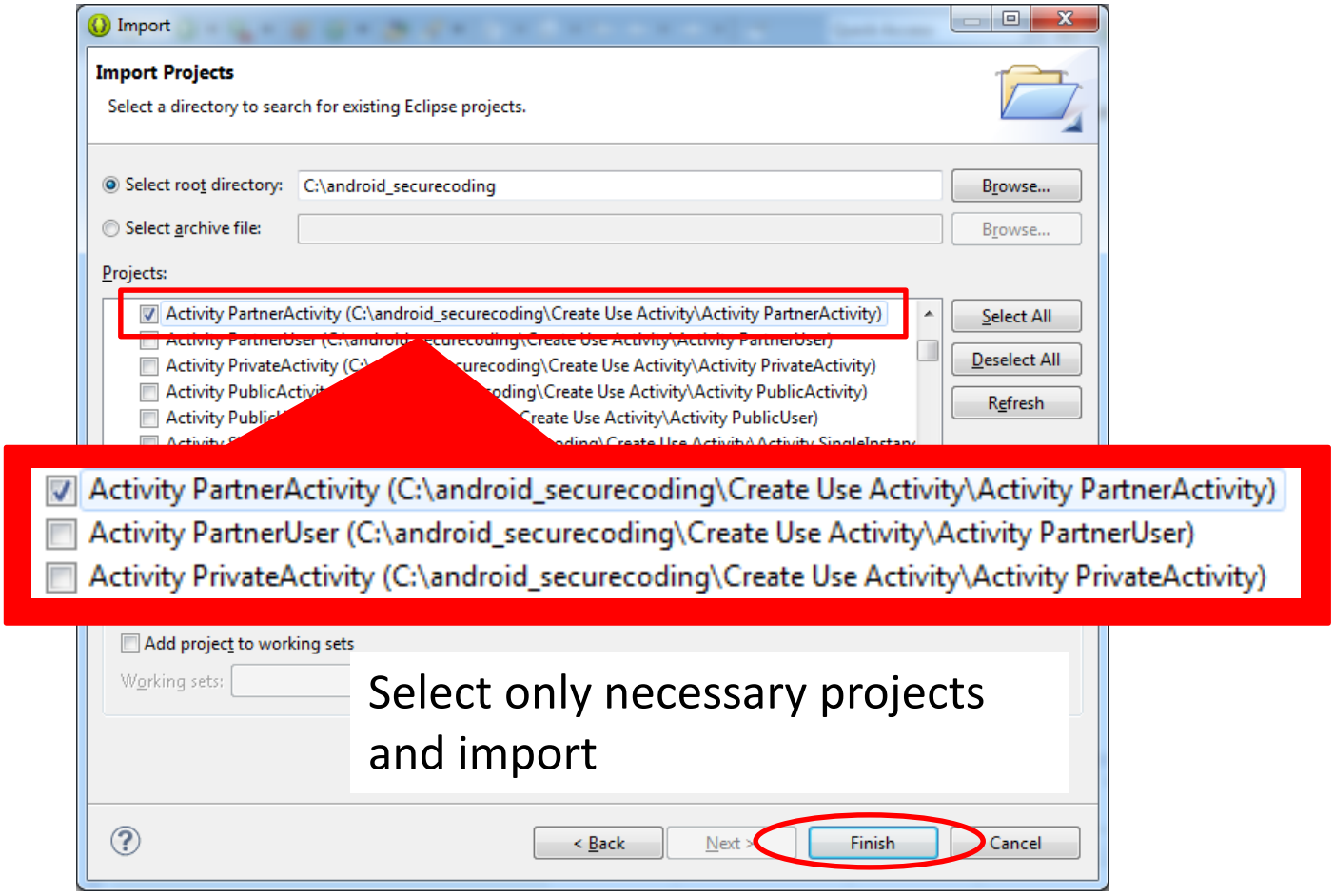


Figure 2.5-12

Some projects may refer to the "JSSEC Shared" folder. In that case, if the "JSSEC Shared" is not imported, it will look like the figure below. Please import "JSSEC Shared" if you need it

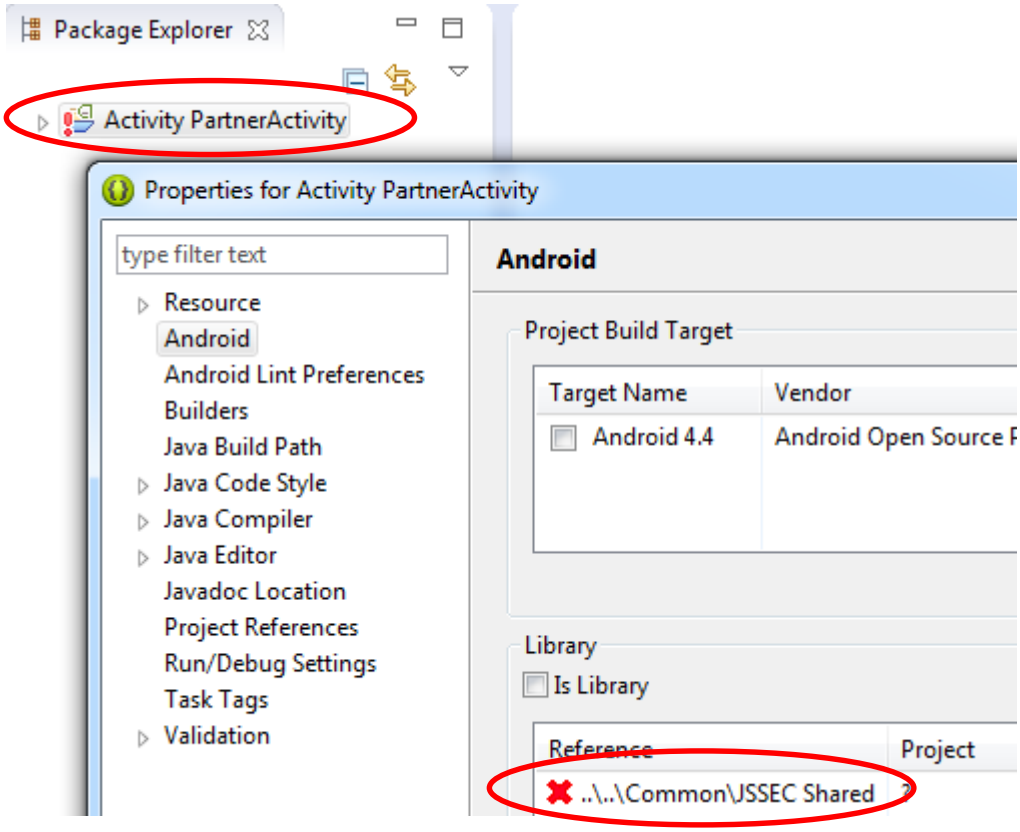


Figure 2.5-13

Select "JSSEC Shared" and import it as shown below.

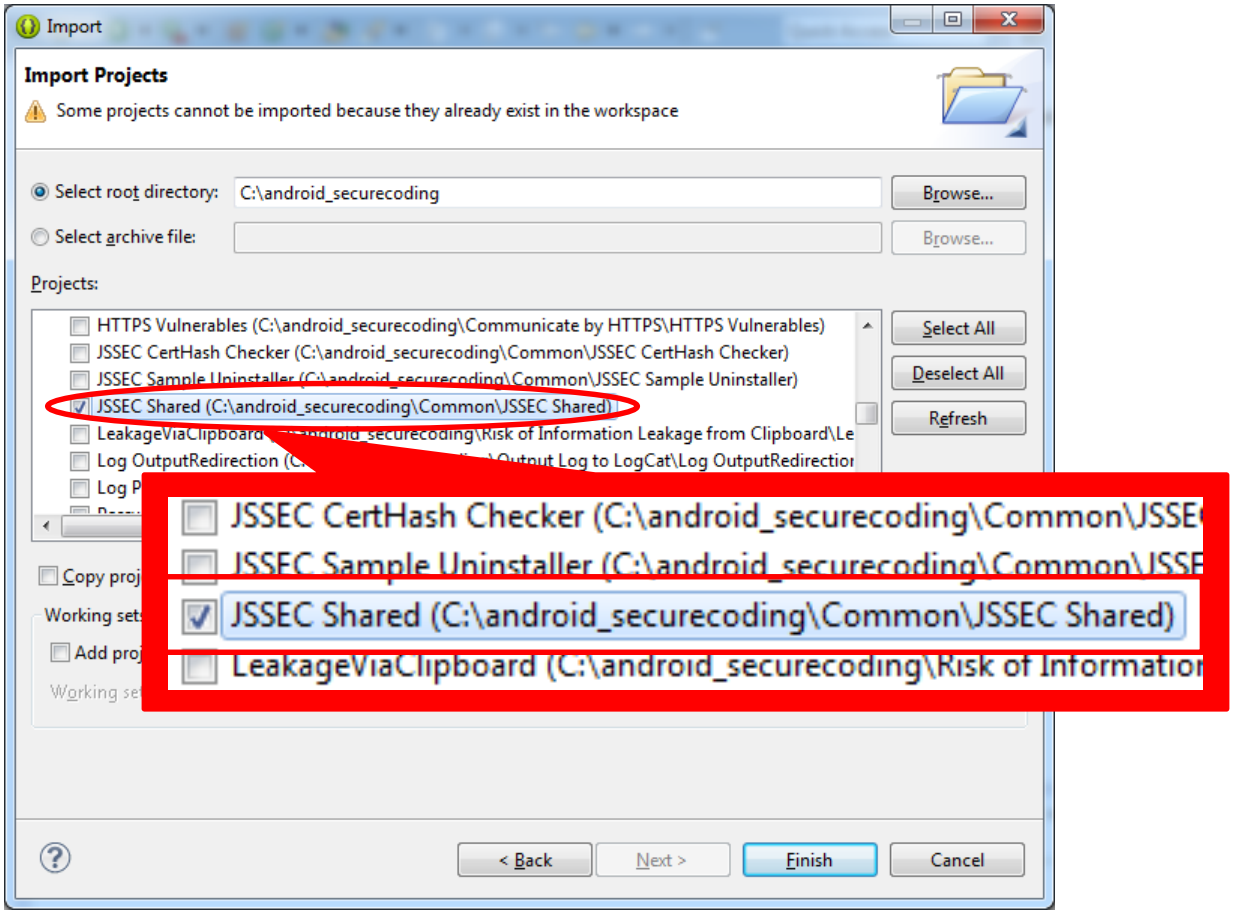


Figure 2.5-14

2.5.3. Setup the Sample Code Validator debug.keystore

A signature is needed in order to activate a sample-code-generated application onto an Android device or emulator. Install the debugging key file "debug.keystore" that will be used for the signature into Eclipse.

1. Click on Window->Preferences

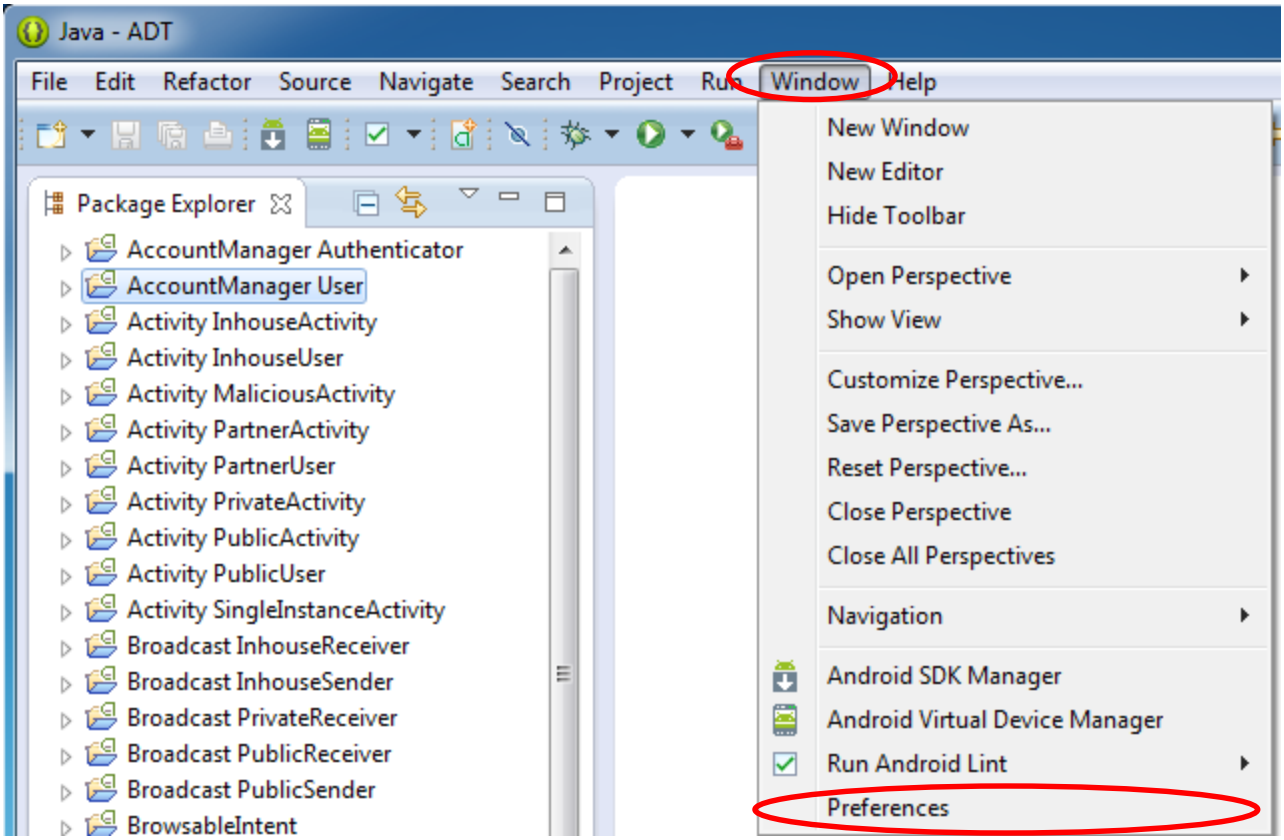


Figure 2.5-15

2. Click "Browse" after selecting Android->Build

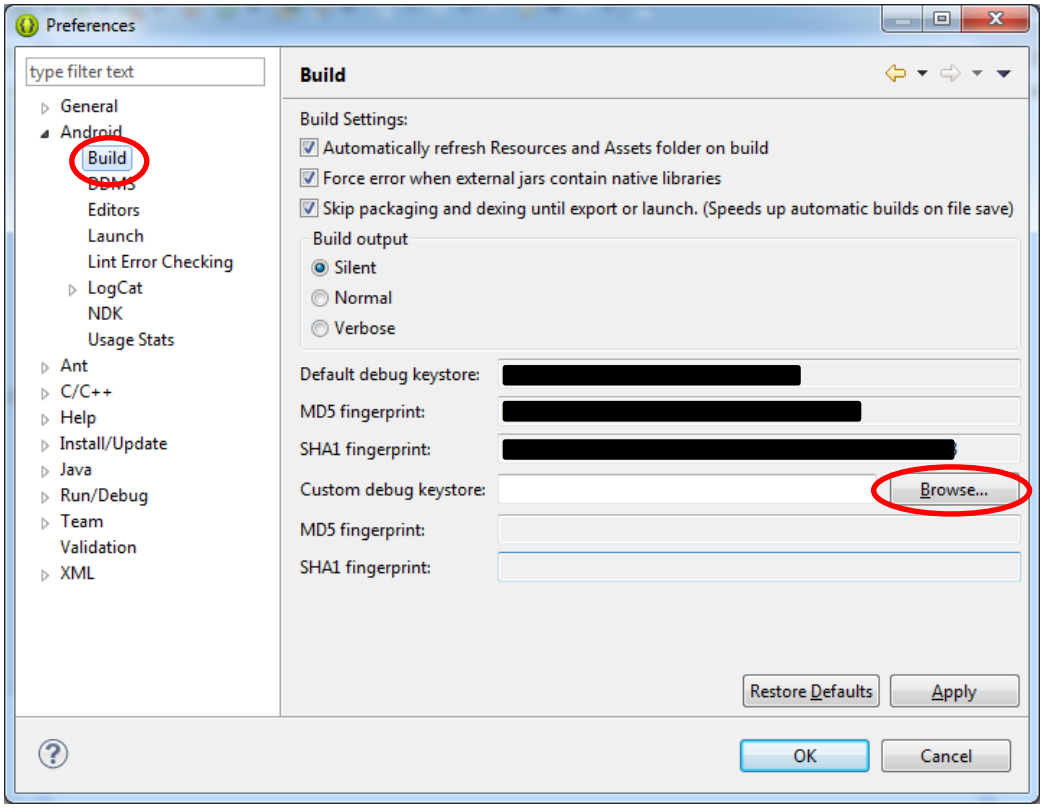


Figure 2.5-16

3. Click on and open "debug.keystore"
   Debug.keystore is contained in the sample code (underneath the android_securecoding folder)

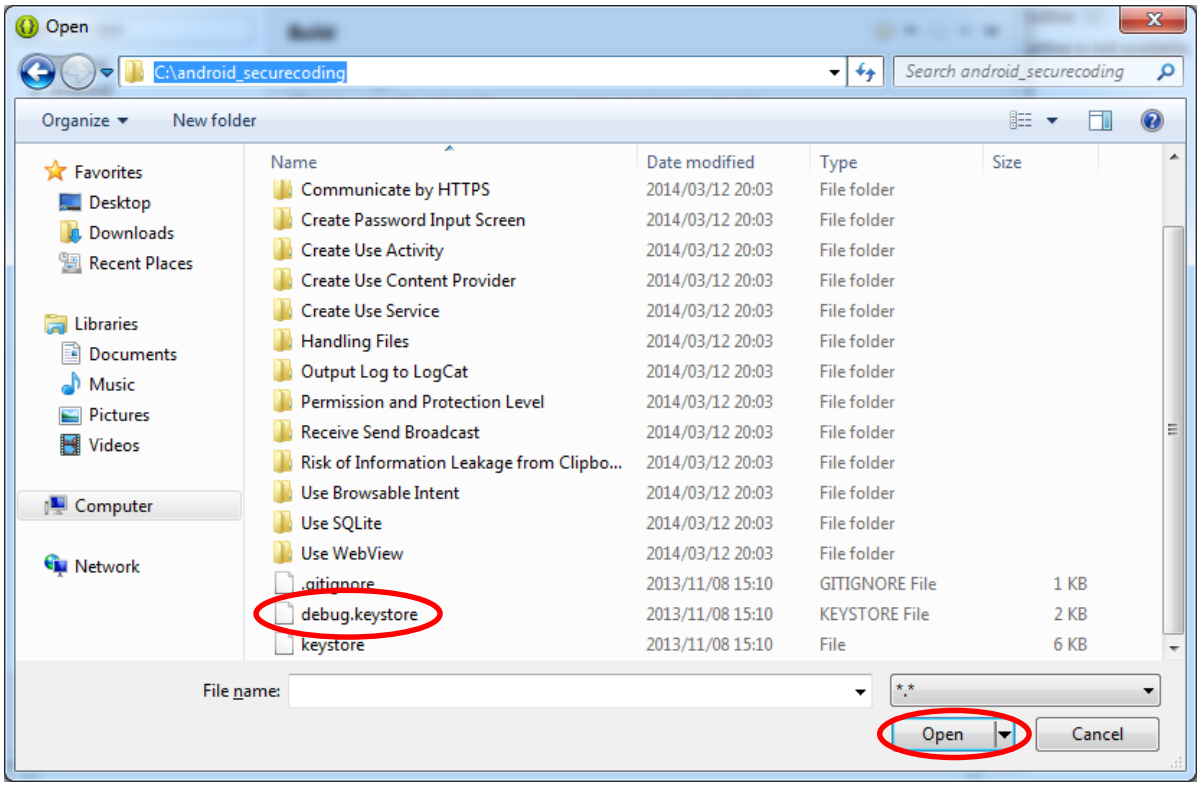


Figure 2.5-17

4. Click "OK" and apply the setting
   Finally, after verifying that the path of debug.keystore, click on "OK" to finish as shown below.

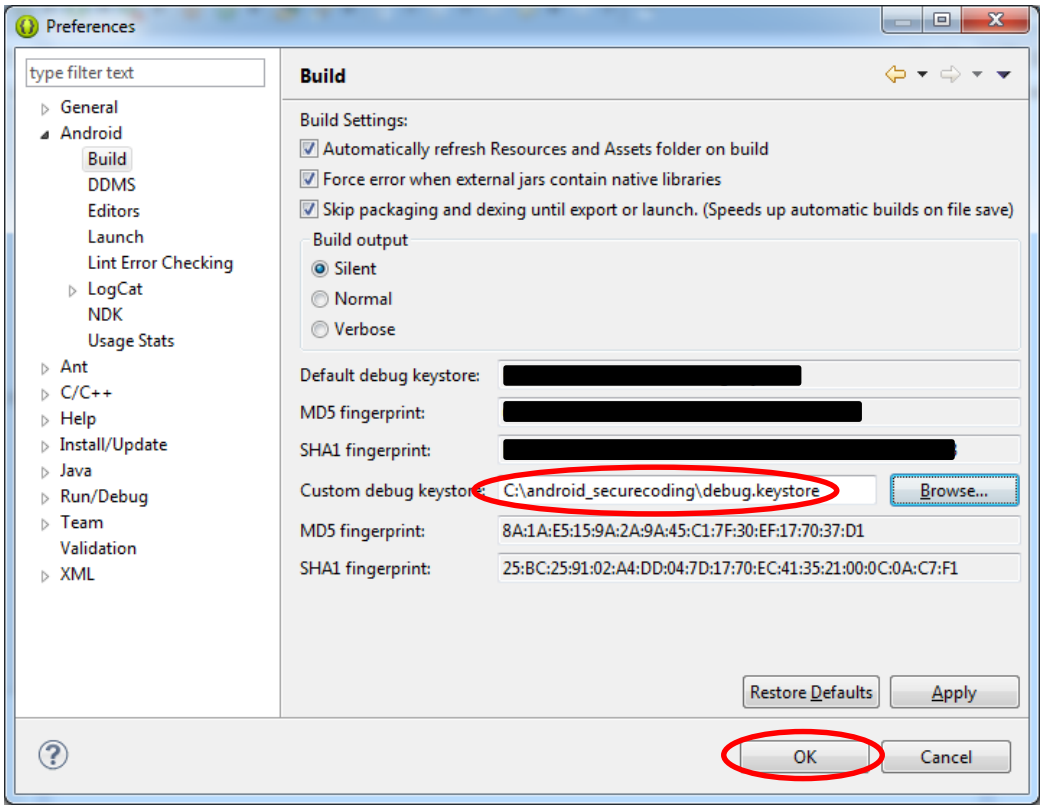


Figure 2.5-18

3. Basic Knowledge of Secure Design and Secure Coding

Although the Guidebook is a collection of security advice concerning Android application development, this chapter will deal with the basic knowledge on general secure design and secure coding of Android smartphones and tablets. Since we will be referring to secure design and coding concepts in the later chapters we recommend that you familiarize yourself with the content contained in this chapter first.

3.1. Android Application Security

There is a commonly accepted way of thinking when examining security issues concerning systems or applications. First, we need to have a grasp over the objects we want to protect. We will call these assets. Next, we want to gain an understanding over the possible attacks that can take place on an asset. We will call these threats. Finally, we will examine and implement measures to protect assets from the various threats. We will call these countermeasures.

What we mean by countermeasures here is secure design and secure coding, and will deal with these subjects after Chapter 4. In this section, we will focus on explaining assets and threats.

3.1.1. Asset: Object of Protection

There are two types of objects of protection within a system or an application: information and functions. We will call these information assets and function assets. An information asset refers to the type of information that can be referred to or changed only by people who have permission. It is a type of information that cannot be referred to or changed by anyone who does not have the permission. A function asset refers to a function that can be used only by people who have permission and no one else.

Below, we will introduce types of information assets and functional assets that exist in Android smartphones and tablets. We would like you to use the following as a point of reference to deliberate on matters with regard to assets when developing a system that utilizes Android applications or Android smartphones/tablets. For the sake of simplicity, we will collectively call Android smartphones/tablets as Android smartphones.

3.1.1.1. Information Asset of an Android Smartphone

Table 3.1-1 and Table 3.1-2 represent examples of information contained on an Android smartphone. Appropriate protection is necessary since this information is equivalent to personal information, confidential information or information that belongs to both.

Table 3.1-1 Examples of Information Managed by an Android Smartphone

Information	Remarks
Phone number	Telephone number of the smartphone itself
Call history	Time and date of incoming and outgoing calls as well as phone numbers
IMEI	Device ID of the smartphone
IMSI	Subscriber ID
Sensor information	GPS, geomagnetic, rate of acceleration, etc.
Various setup information	Wi-Fi setting value, etc...
Account information	Various account information, authentication information, etc.
Media data	Pictures, videos, music, recording, etc...
...	

Table 3.1-2 Examples of Information Managed by an Application

Information	Remarks
Contacts	Contacts of acquaintances
E-mail address	User's e-mail address
E-mail mail box	Content of incoming and outgoing e-mail, attachments, etc.
Web bookmarks	Bookmarks
Web browsing history	Browsing history
Calendar	Plans, to-do list, events, etc.
Facebook	SNS content, etc.
Twitter	SNS content, etc.
...	

The type of information seen in Table 3.1-1 is mainly the type of information that is stored on the Android smartphone itself or on an SD card. Similarly, the type of information seen in Table 3.1-2 is primarily managed by an application. In particular, the type of information seen in Table 3.1-2 grows in proportion to the number of applications installed on the device.

Table 3.1-3 is the amount of information contained in one entry case of contacts. The information here is not of the smartphone user's, but of the smartphone user's friends. In other words, we must be aware that a smartphone not only contains information on the user, but of other people too.

Table 3.1-3 Examples of Information Contained in One Contact Entry

Information	Content
Phone number	Home phone number, mobile phone number, FAX, MMS, etc.
E-mail address	Home e-mail, work e-mail, mobile phone e-mail, etc.
Photo	Thumbnail image, large image, etc.
IM address	AIM, MSN, Yahoo, Skype, QQ, Google Talk, ICQ, Jabber, Net meeting, etc.
Nicknames	Acronyms, initials, maiden names, nicknames, etc.
Address	Country, postal code, region, area, town, street name, etc.
Group membership	Favorites, family, friends, coworkers, etc.
Website	Blogs, profile site, homepage, FTP server, home, office, etc.
Events	Birthdays, anniversaries, others, etc.

Relation	Spouse, children, father, mother, manager, assistants, domestic partner, partners, etc.
SIP address	Home, work, other, etc.
…	…

Until now, we have primarily focused on information about smartphone users, however, application possesses other important information as well. Figure 3.1-1 displays a typical view of the information inside an application divided into the program portion and data portion. The program portion mainly consists of information about the application developer, and the data portion mostly pertains to user information. Since there could be information that an application developer may not want a user to have access to, it is important to provide protective countermeasures to prohibit a user from referring to or making changes to such information.

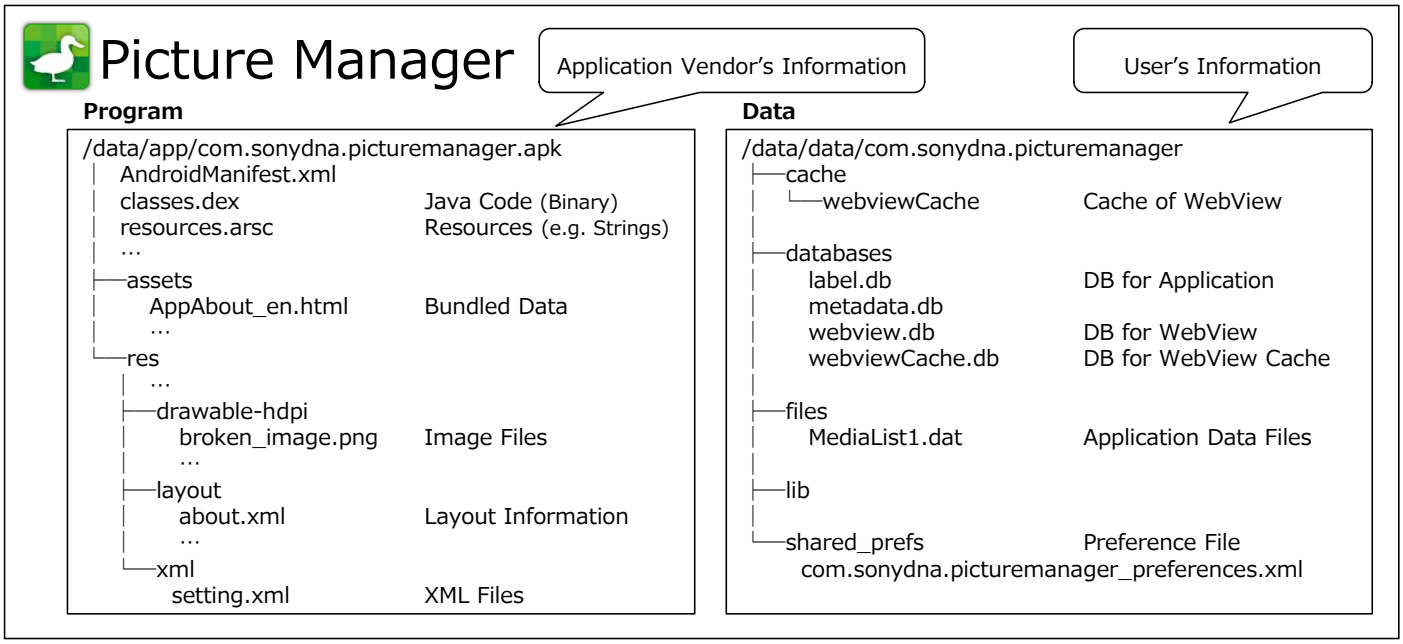


Figure 3.1-1 Information Contained in an Application

When creating an Android application, it is important to employ appropriate protective countermeasures for information that an application manages itself, such as shown in Figure 3.1-1. However, it is equally important to have robust security measure in place for information contained in the Android smartphone itself as well as for information that has been gained from other applications such as shown in Table 3.1-1, Table 3.1-2, and Table 3.1-3.

3.1.1.2. Function Assets of an Android Smartphone

Table 3.1-4 shows examples of features that an Android OS provides to an application. When these features are exploited by a malware, etc., damages in the form of unexpected charges or loss of privacy may be incurred by a user. Therefore, appropriate protective counter-measures that are equal the one extended to information asset should be set in place.

Table 3.1-4 Examples of Features an Android OS Provides to an Application

Function	Function
Sending and receiving SMS messages	Camera
Calling	Volume
Network communication	Reading the Contract List and Status of the Mobile Phone
GPS	SD card
Bluetooth communication	Change system setup
NFC communication	Reading Log Data
Internet communication (SIP)	Obtaining Information of a Running Application
...	...

In addition to the functions that the Android OS provides to an application, the inter-application communication components of Android applications are included as part of the function assets as well. Android applications can allow other applications to utilize features by accessing their internal components. We call this inter-application communication. This is a convenient feature, however, there have been instances where access to functions that should only be used inside a particular application are mistakenly given to other applications due the lack of knowledge regarding secure coding on the part of the developer. There are functions provided by the application that could be exploited by malware that resides locally on the device. Therefore, it is necessary to have appropriate protective countermeasures to only allow legitimate applications to access these functions.

3.1.2. Threats: Attacks that Threaten Assets

In the previous section, we talked about the assets of an Android smartphone. In this section, we will explain about attacks that can threaten an asset. Put simply, a threat to an asset is when a third party who should not have permission, accesses, changes, deletes or creates an information asset or illicitly uses a function asset. The act of directly or indirectly attacking such assets is called a "threat." Furthermore, the malicious person or applications that commit these acts are referred to as the source of the threats. Malicious attackers and malware are the sources of threats but are not the threats themselves. The relationship between our definitions of assets, threats, threat sources, vulnerabilities, and damage are shown below in Figure 3.1-2.

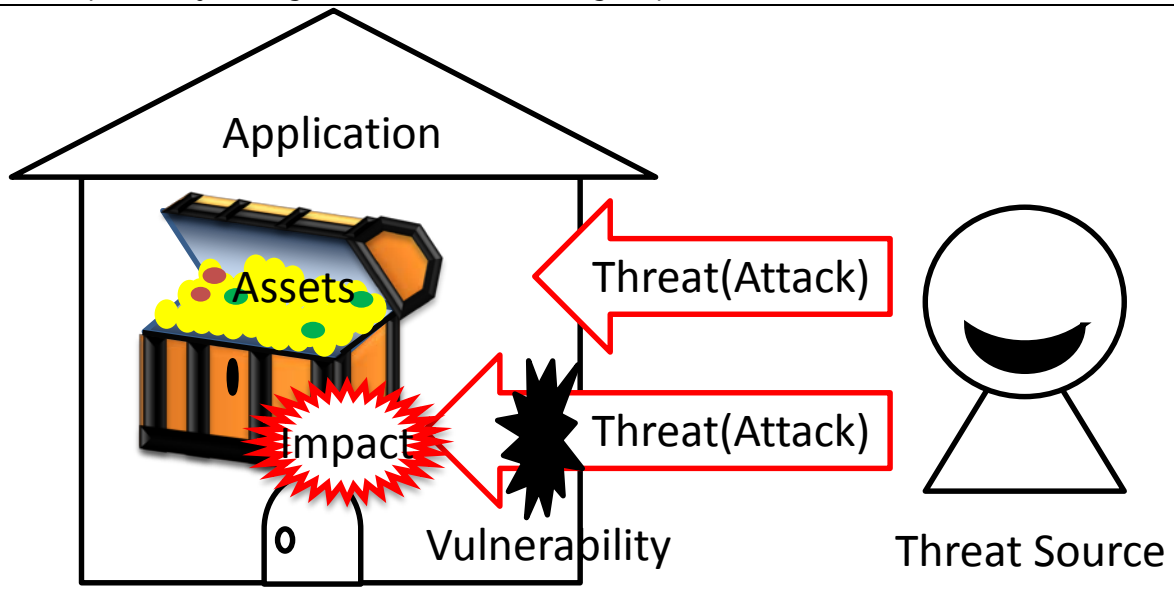


Figure 3.1-2 Relation between Asset, Threat, Threat Source, Vulnerability, and Damage

Figure 3.1-3 shows a typical environment that an Android application behaves in. From now on, in order to expand on the explanation concerning the type of threats an Android application faces by using this figure as a base, we will first learn how to view this figure.

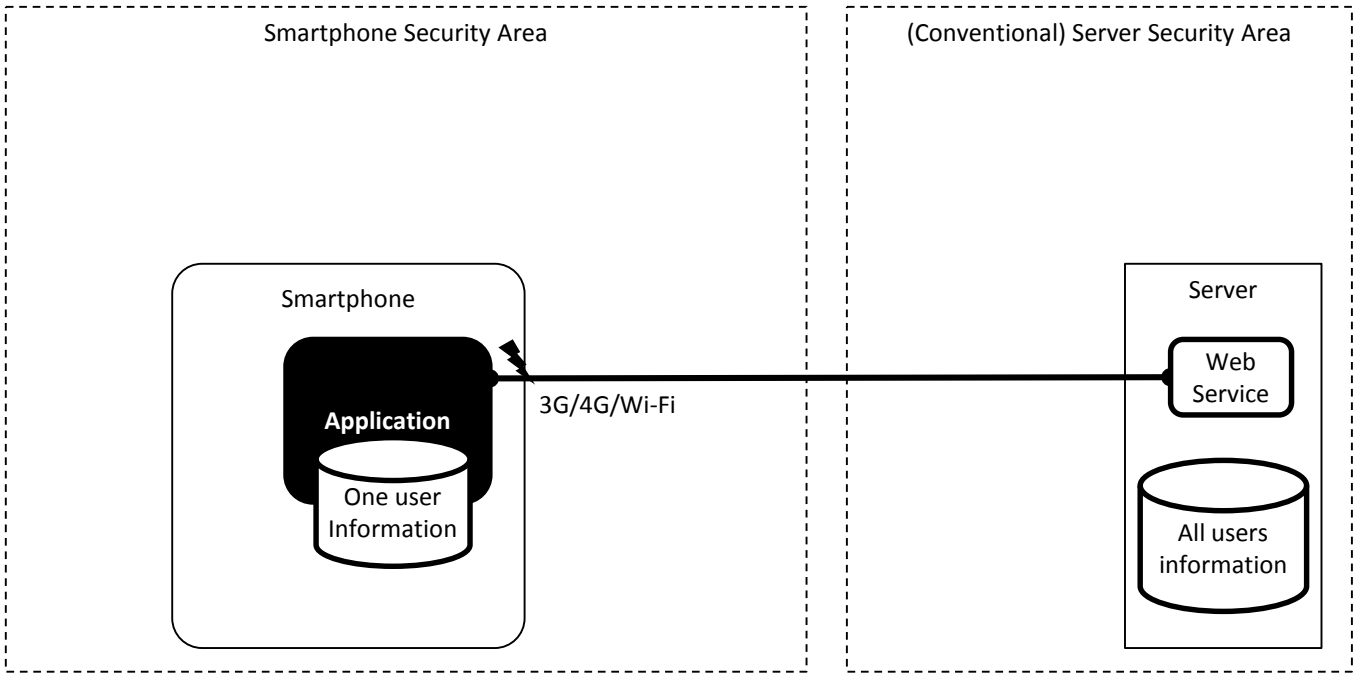


Figure 3.1-3 Typical Environment an Android Application Behaves in

The figure above depicts the smartphone on the left and server on the right. The smartphone and server communicate through the Internet over 3G/4G/Wi-Fi. Although multiple applications exist within a smartphone, we are only showing a single application in the figure in order to explain the threats clearly. Smartphone-based applications mainly handle user information, but the server-based web services collectively manage information of all of its users. Consequently, there is no change the importance of server security as usual. We will not touch upon issues relating to server security as it falls outside of the scope of the Guidebook.

We will use the following figure to describe the type of threats that exist towards Android applications.

3.1.2.1. Network-based Third-Party

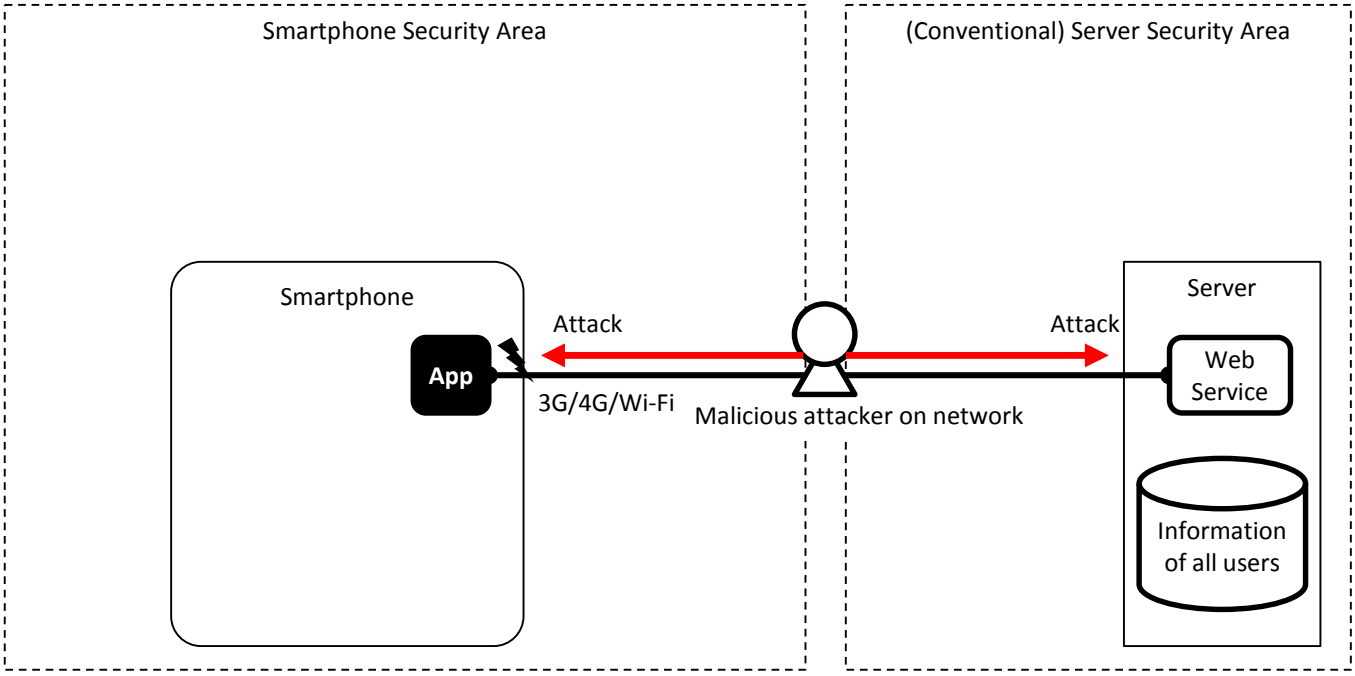


Figure 3.1-4 Network-Based Malicious Third Party Attacking an Application

Generally, a smartphone application manages user information on a server so the information assets will move between the networks connecting them. As indicated in Figure 3.1-4, a network-based malicious third party may access (sniff) any information during this communication or try to change information (data manipulation). The malicious attacker in the middle (also referred to as "Man in The Middle") can also pretend to be the real server tricking the application. Without saying, network-based malicious third parties will usually try to attack the server as well.

3.1.2.2. Threat Due to User-Installed Malware

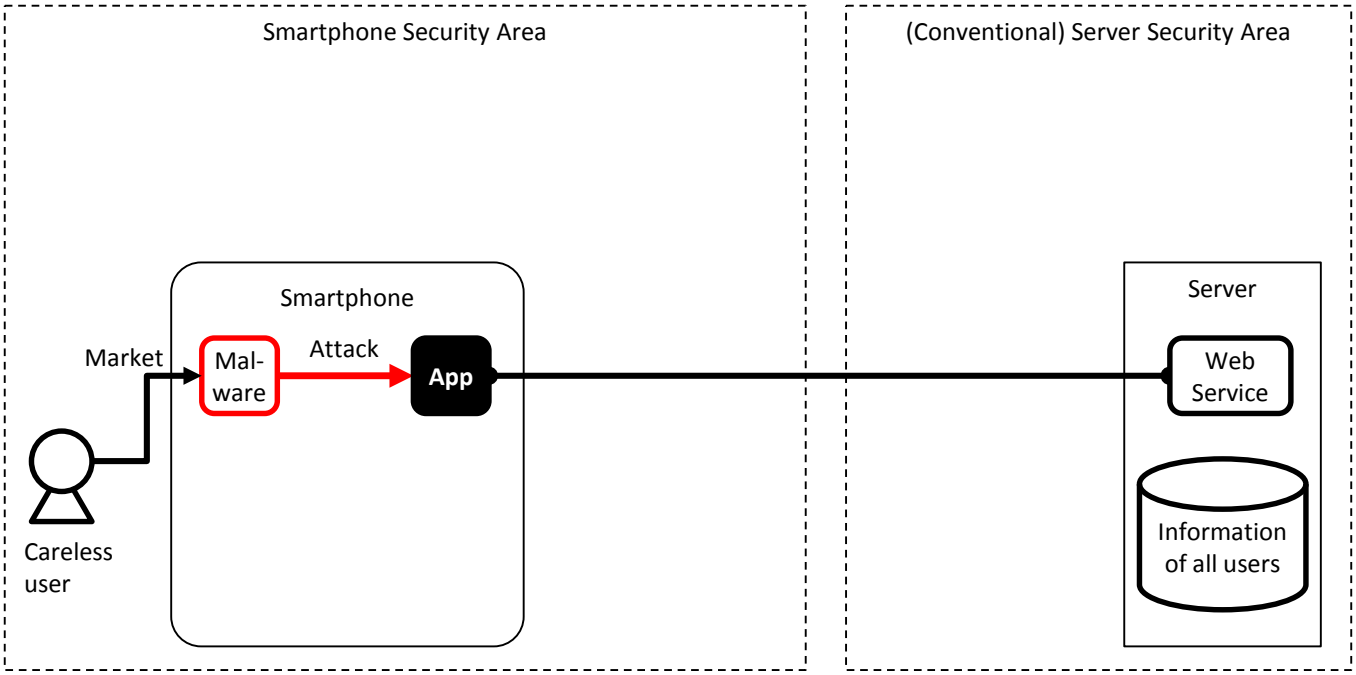


Figure 3.1-5 Malware Installed by a User Attacks an Application

The biggest selling point of a smartphone is in its ability to acquire numerous applications from the market in order to expand on its features. The downside to users being able to freely install many applications is that they will sometimes mistakenly install malware. As shown in Figure 3.1-5, malware may exploit the inter-application communication functions or a vulnerability in the application in order to gain access to information or function assets.

3.1.2.3. Threat of an Malicious File that Exploits a Vulnerability in an Application

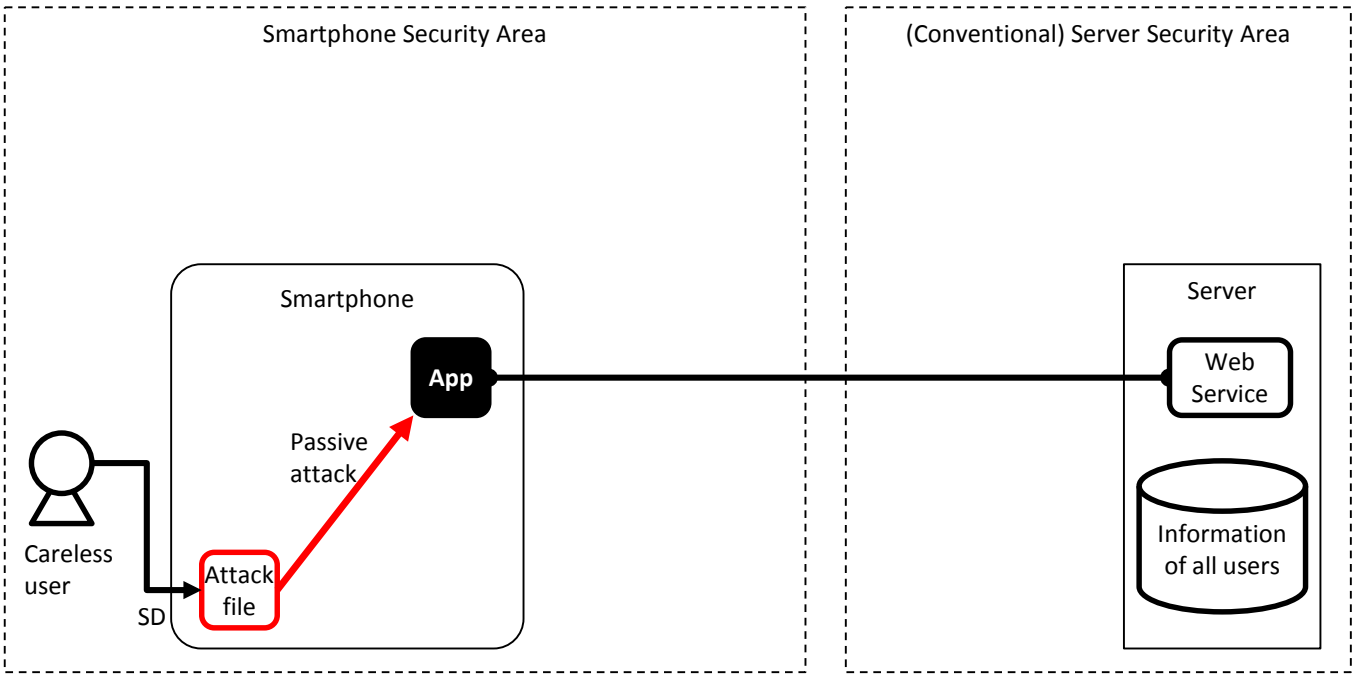


Figure 3.1-6 Attack from Malicious Files that Exploit a Vulnerability in an Application

Various types of files such as music, images, videos and documents are widely available on the

Internet and typically users will download many files to their SD card in order to use them on their smartphone. Furthermore, it is also common to download attached files sent in an e-mail. These files are later opened by a viewing or editing application.

If there is any vulnerability in the function of an application that processes these files, an attacker can use a malicious file to exploit it and gain access to information or function assets of the application. In particular, vulnerabilities are often present in processing a file format with a complex data structure. The attacker can fulfill many different goals when exploiting an application in this way.

As shown in Figure 3.1-6, an attack file stays dormant until it is opened by a vulnerable application. Once it is opened, it will start causing havoc by taking advantage of an application's vulnerability. In comparison to an active attack, we call this attack method a "Passive Attack."

3.1.2.4. Threats from a Malicious Smartphone User

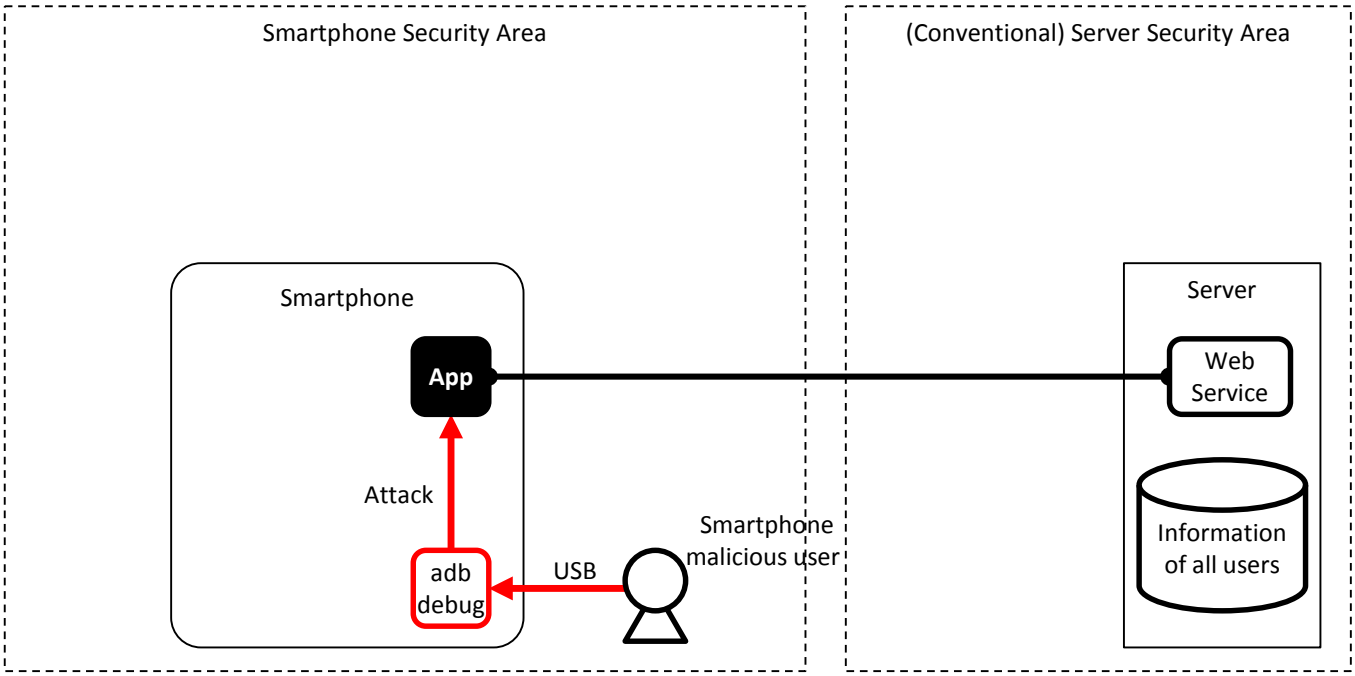


Figure 3.1-7 Attacks from a Malicious Smartphone User

With regard to application development for an Android smartphone, the environment as well as features that help to develop and analyze an application are openly provided to the general user. Among the features that are provided, the useful ADB debugging feature can be accessed by anyone without registration or screening. This feature allows an Android smartphone user to easily perform OS or application analysis.

As it is shown in Figure 3.1-7, a smartphone user with malicious intent can analyze an application by taking advantage of the debugging feature of ADB and try to gain access to information or function assets of an application. If the actual asset contained in the application belongs to the user, it poses no problem, but if the asset belongs to someone other than the user, such as the application developer, then it will become a concern. Accordingly, we need to be aware that the legitimate smartphone user can maliciously target the assets within an application.

3.1.2.5. Threats from Third Party in the Proximity of a Smartphone

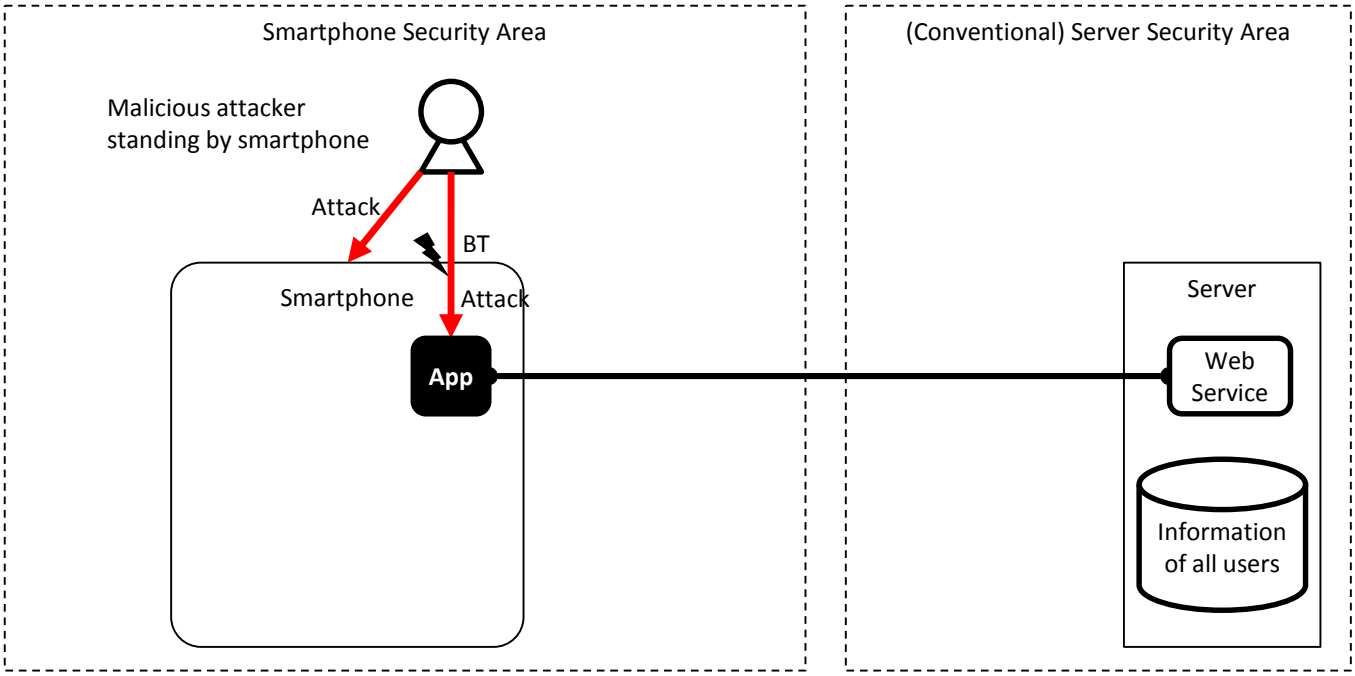


Figure 3.1-8 Attacks from a Malicious Third Party in the Proximity of a Smartphone

Due to face that most smartphones possess a variety of near-field communication mechanisms, such as NFC, Bluetooth and Wifi, we must not forget that attacks can occur from a malicious attacker who is in physical proximity of a smartphone. An attacker can shoulder surf a password while peeping over a user who is inputting it in. Or, as indicated in Figure 3.1-8, an attacker can be more sophisticated and attack the Bluetooth functionality of an application from a remote distance. There is also the threat that a malicious person could steal the smartphone creating a risk of data leakage or even destroy the smartphone causing a loss of critical information. Developers need to take these risks into consideration as well as early as the design stage.

3.1.2.6. Summary of Threats

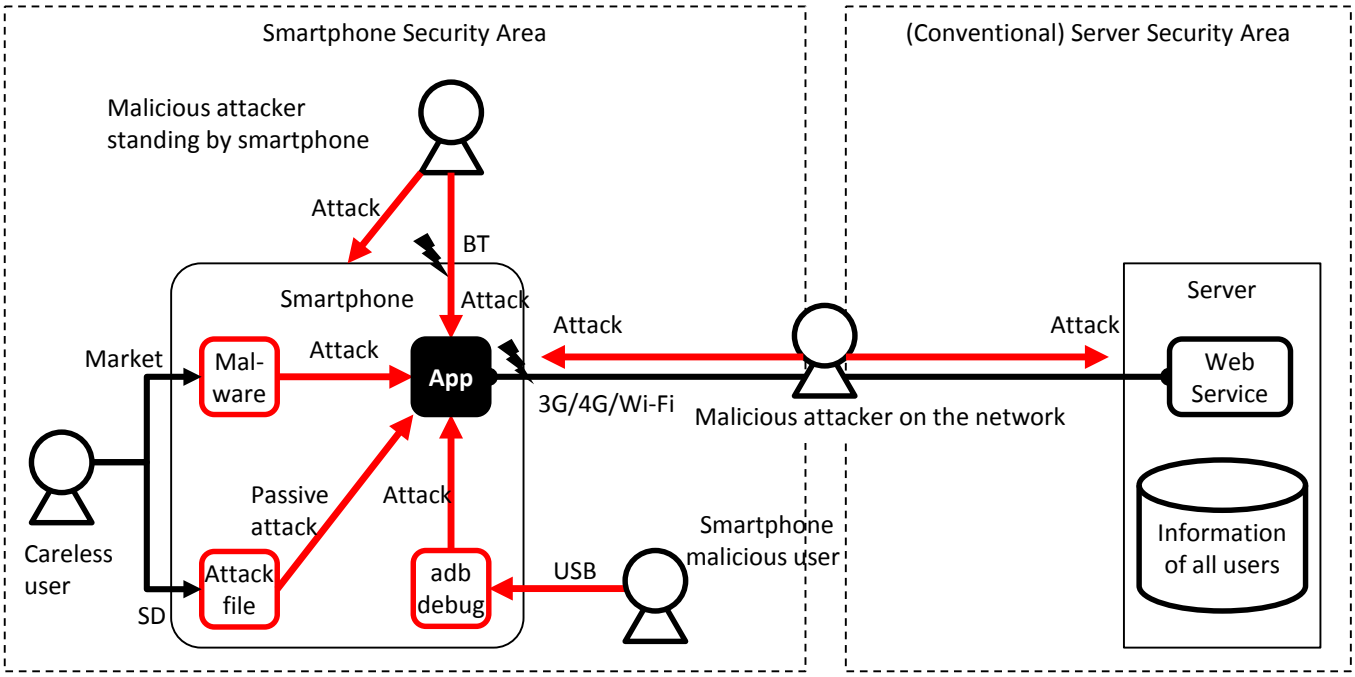


Figure 3.1-9 Summary of the Various Attacks on Smartphone Applications

Figure 3.1-9 summarizes the main types of threats explained in the previous sections. Smartphones are surrounded by a wide variety of threats and the figure above does not include all of them. Through our daily information gathering, we need to spread the awareness concerning the various threats that surround an Android application and be aware of them during the application's secure design and coding. The following literature that was created by Japan's Smartphone Security Association (JSSEC) contains other valuable information on the threats to smartphone security.

- Security Guidebook for Using Smartphones and Tablets [Version 1]
  http://www.jssec.org/dl/Guidebook2011_v1.1.pdf (Japanese)
  http://www.jssec.org/dl/Guidebook2012Enew_v1.0.pdf (English)

- Implementation Guidebook for Smartphone Network Security[Version 1]
  http://www.jssec.org/dl/NetworkSecurityGuide1.pdf (Japanese)

- Cloud Usage Guidebook for Business Purposes of Smartphones [Beta Version]
  http://www.jssec.org/dl/cloudguide2012_beta.pdf (Japanese)

- Guidebook for Reviewing the Implementation/Operation of MDM [Version1]
  http://www.jssec.org/dl/MDMGuideV1.pdf (Japanese)

3.1.3. Asset Classification and Protective Countermeasures

As was discussed in the previous sections, Android smartphones are surrounded by a variety of threats. Protecting every asset in an application from such threats could prove to be very difficult given the time it takes for development and due to technical limitations. Consequently, Android application developers should examine feasible countermeasures for their assets. This should be done according to priority level based on the developer's judgement criteria. This is a subjective matter that is based on how the importance of an asset is viewed and what the accepted level of damage is.

In order to help decide on the protective countermeasures for each asset, we will classify them and stipulate the level of protective countermeasures for each group. This will be achieved by examining the legal basis, pertaining to the level of importance regarding the impact of any damages that can occur and the social responsibility of the developer (or organization). These will prove to be the judgement criteria when deciding on how to handle each asset and the implementation of the type of countermeasures. Since this will become a standard for application developers and organizations on determining how to handle an asset and provide protective countermeasures, it is necessary to specify the classification methods and pertaining countermeasures in accordance the application developer's (or organization's) circumstances.

Asset classification and protective countermeasure levels that are adopted in the Guidebook are shown below for reference:

Table 3.1-5 Asset Classification and Protective Countermeasure Levels

Asset Classification	Asset Level	Level of Protective Counter-Measures
High	The amount of damage the asset causes is fatal and catastrophic to the organization or an individual's activity. i.e.) When an asset at this level is damaged, the organization will not be able to continue its business.	• Provide protection against sophisticated attacks that break through the Android OS security model and prevent root privilege compromises and attacks that alter the dex portion of an APK. • Ensure security takes priority over other elements such as user experience, etc.
Medium	The amount of damage the asset causes has a substantial impact the organization or an individual's activity. i.e.) When an asset at this level is damaged, the organization's profit level deteriorates, adversely affecting its business.	• Utilize the Android OS security model. It will provide protection covered under its scope. • Ensure security takes priority over other elements such as user experience, etc.
Low	The amount of damage the asset causes has a limited impact on the organization or an individual's activity. I.e.) When an asset at this level is damaged, the organization's profit level will be affected but is able to compensate its losses from other resources.	• Utilize the Android OS security model. It will provide protection covered under its scope. • Compare security countermeasures with other elements such as user experience, etc. At this level, it is possible for non-security issues to take precedence over security issues.

This Guidebook's Scope of Focus

Asset classification and protective countermeasures described in the Guidebook are proposed under the premise of a secure Android device where root privilege has not been compromised. Furthermore, it is based on the security measures that utilize the security model of Android OS. Specifically, we are hypothetically devising protective countermeasures by utilizing the Android OS security model on the premise of a functioning Android OS security model against assets that are classified lower than or equal to the medium level asset. On the other hand, we also believe in the necessity of protecting high level assets from attacks that are caused due the breaching of the Android OS security model. Such attacks include the compromise of root privileges and attacks that analyze or alter the APK binary. To protect these types of assets, we need to design sophisticated defensive countermeasures against such threats through the combination of multiple methods such as encryption, obfuscation, hardware support and server support. As the collection of know-how regarding these defenses cannot be easily written in this guidebook, and since appropriate defensive design differ in accordance to individual circumstances, we have deemed them to be outside of the Guidebook's scope. We recommend that you consult with a security specialist who is well versed in tamper resistant designs of Android if your device requires protection from sophisticated attacks that include attacks resulting from the compromise of root privileges or attacks caused by the analysis or alteration of an APK file.

3.1.4. Sensitive Information

The term "sensitive information", instead of information asset, will be used from now on in the Guidebook. As it has been aforementioned in the previous section, we have to determine the asset level and the level of protective countermeasures for each information asset that an application handles.

3.2. Handling Input Data Carefully and Securely

Validating input data is the easiest and yet most effective secure coding method. All data that is inputted into the application either directly or indirectly by an outside source needs to be properly validated. To illustrate best practices of input data validation, the following is an example of an Activity as used in a program that receives data from Intent.

It is possible that an Activity can receive data from an Intent that was tampered by an attacker. By sending data with a format or a value that a programmer is not expecting, the attacker can induce a malfunction in the application that leads to some sort of security incident. We must not forget that a user can become an attacker as well.

Intents are configured by action, data and extras, and we must be careful when accepting all forms of data that can be controlled by an attacker. We always need to validate the following items in any code that handles data from an untrusted source.

(a) Does the received data match the format that was expected by the programmer and does the value fall in the expected scope?
(b) Even if you have received the expected format and value, can you guarantee that the code which handles that data will not behave unexpectedly?

The next example is a simple sample where HTML is acquired from a remote web page in a designated URL and the code is displayed in TextView. However, there is a bug.

Sample Code that Displays HTML of a Remote Web page in TextView

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { ...
```

From the viewpoint of (a), "urlstr is the correct URL", verified through the non-occurrence of a MalformedURLException by a new URL(). However, this is not sufficient. Furthermore, when a "file://..." formatted URL is designated by urlstr, the file of the internal file system is opened and is displayed in TextView rather than the remote web page. This does not fulfill the viewpoint of (b), since it does not guarantee the behavior which was expected by the programmer.

The next example shows a revision to fix the security bugs. Through the viewpoint of (a), the input data is validated by checking that "urlstr is a legitimate URL and the protocol is limited to http or https." As a result, even by the viewpoint of (b), the acquisition of an Internet-routed InputStream is guaranteed through url.openConnection().getInputStream().

Revised sample code that displays HTML of Internet-based Web page in TextView

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    String prot = url.getProtocol();
    if (!"http".equals(prot) && !"https".equals(prot)) {
        throw new MalformedURLException("invalid protocol");
    }
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { ...
```

Validating the safety of input data is called "Input Validation" and it is a fundamental secure coding method. Surmising from the sense of the word of Input Validation, it is quite often the case where the viewpoint of (a) is heeded but the viewpoint of (b) is forgotten. It is important to remember that damage does not take place when data enters the program but when the program uses that data in an incorrect way. We hope that you will refer the URLs listed below.

- The CERT Oracle Secure Coding Standard for Java
  https://www.securecoding.cert.org/confluence/x/Ux (English)

- Application of CERT Oracle Secure Coding Standard for Android Application Development
  https://www.securecoding.cert.org/confluence/x/C4AiBw (English)

- Rules Applicable Only to the Android Platform (DRD)
  https://www.securecoding.cert.org/confluence/x/H4ClBg (English)

- IPA "Secure Programming Course"
  http://www.ipa.go.jp/security/awareness/vendor/programmingv2/clanguage.html (Japanese)

4. Using Technology in a Safe Way

In Android, there are many specific security related issues that pertain only to certain technologies such as Activities or SQLite. If a developer does not have enough knowledge about each of the different security issues regarding each technology when designing and coding, then unexpected vulnerabilities may arise. This chapter will explain about the different scenarios that developers will need to know when using their application components.

4.1. Creating/Using Activities

4.1.1. Sample Code

The risks and countermeasures of using Activities differ depending on how that Activity is being used. In this section, we have classified 4 types of Activities based on how the Activity is being used. You can find out which type of activity you are supposed to create through the following chart shown below. Since the secure coding best practice varies according to how the activity is used, we will also explain about the implementation of the Activity as well.

Table 4.1-1 Definition of Activity Types

Type	Definition
Private Activity	An activity that cannot be launched by another application, and therefore is the safest activity
Public Activity	An activity that is supposed to be used by an unspecified large number of applications.
Partner Activity	An activity that can only be used by specific applications made by a trusted partner company.
In-house Activity	An activity that can only be used by other in-house applications.

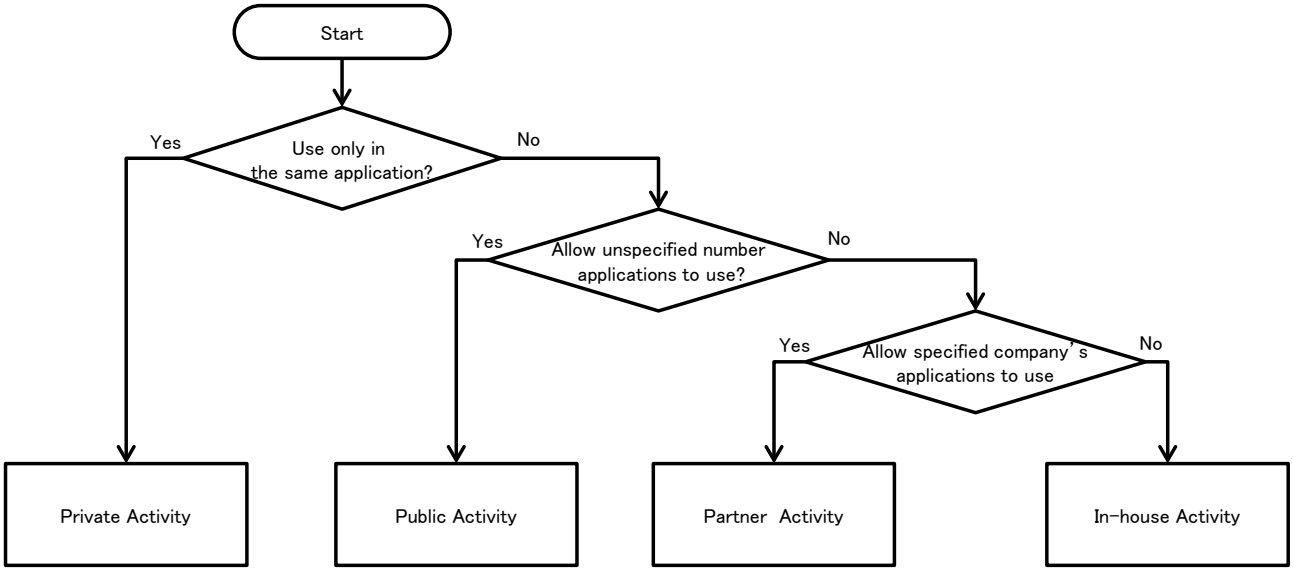


Figure 4.1-1

4.1.1.1. Creating/Using Private Activities

Private Activities are Activities which cannot be launched by the other applications and therefore it is the safest Activity.

When using Activities that are only used within the application (Private Activity), as long as you use explicit Intents to the class then you do not have to worry about accidently sending it to any other application. However, there is a risk that a third party application can read an Intent that is used to start the Activity. Therefore it is necessary to make sure that if you are putting sensitive information inside an Intent used to start an Activity that you take countermeasures to make sure that it cannot be read by a malicious third party.

Sample code of how to create a Private Activity is shown below.

Points (Creating an Activity):

1. Do not specify taskAffinity.
2. Do not specify launchMode.
3. Explicitly set the exported attribute to false.
4. Handle the received intent carefully and securely, even though the intent was sent from the same application.
5. Sensitive information can be sent since it is sending and receiving all within the same application.

To make the Activity private, set the "exported" attribute of the Activity element in the AndroidManifest.xml to false.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.privateactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Private activity -->
        <!-- *** POINT 1 *** Do not specify taskAffinity -->
        <!-- *** POINT 2 *** Do not specify launchMode -->
        <!-- *** POINT 3 *** Set false for the exported attribute explicitly. -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />

        <!-- Public activity launched by launcher -->
        <activity
            android:name=".PrivateUserActivity"
            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>
</manifest>
```

PrivateActivity.java

```
package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.private_activity);

        // *** POINT 4 *** Handle the received Intent carefully and securely, even though the Intent was sent from the
 same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("Received param: ¥"%s¥"", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 5 *** Sensitive information can be sent since it is sending and receiving all within the same app
lication.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

Next, we show the sample code for how to use the Private Activity.

Point (Using an Activity):

6. Do not set the FLAG_ACTIVITY_NEW_TASK flag for intents to start an activity.
7. Use the explicit Intents with the class specified to call an activity in the same application.
8. Sensitive information can be sent since the destination activity is in the same application.[1]
9. Handle the received result data carefully and securely, even though the data comes from an activity within the same application.

PrivateUserActivity.java

```
package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 6 *** Do not set the FLAG_ACTIVITY_NEW_TASK flag for intents to start an activity.
        // *** POINT 7 *** Use the explicit Intents with the class specified to call an activity in the same applicati
on.
        Intent intent = new Intent(this, PrivateActivity.class);

        // *** POINT 8 *** Sensitive information can be sent since the destination activity is in the same application
.
        intent.putExtra("PARAM", "Sensitive Info");

        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");
```

[1] Caution: Unless points 1, 2 and 6 are abided by, there is a risk that Intents may be read by a third party. Please refer to sections 4.1.2.2 and 4.1.2.3 for more details.


```
            // *** POINT 9 *** Handle the received data carefully and securely,
            // even though the data comes from an activity within the same application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this, String.format("Received result: ¥"%s¥"", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

4.1.1.2. Creating/Using Public Activities

Public Activities are Activities which are supposed to be used by an unspecified large number of applications. It is necessary to be aware that Public Activities may receive Intents sent from malware. In addition, when using Public Activities, it is necessary to be aware of the fact that malware can also receive or read the Intents sent to them.

The sample code to create a Public Activity is shown below.

Points (Creating an Activity):

1. Handle the received intent carefully and securely.
2. When returning a result, do not include sensitive information.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.publicactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Public Activity -->
        <activity
            android:name=".PublicActivity"
            android:label="@string/app_name"
            android:enabled="true">

            <!-- Define intent filter to receive an implicit intent for a specified action -->
            <intent-filter>
                <action android:name="org.jssec.android.activity.MY_ACTION" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

PublicActivity.java

```
package org.jssec.android.activity.publicactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PublicActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
```



```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 1 *** Handle the received intent carefully and securely.
        // Since this is a public activity, it is possible that the sending application may be malware.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("Received param: ¥"%s¥"", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 2 *** When returning a result, do not include sensitive information.
        // Since this is a public activity, it is possible that the receiving application may be malware.
        // If there is no problem if the data gets received by malware, then it can be returned as a result.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Not Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

Next, Herein after sample code of Public Activity user side.

Points (Using an Activity):

3. Do not send sensitive information.
4. When receiving a result, handle the data carefully and securely.

PublicUserActivity.java

```
package org.jssec.android.activity.publicuser;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PublicUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        try {
            // *** POINT 3 *** Do not send sensitive information.
            Intent intent = new Intent("org.jssec.android.activity.MY_ACTION");
            intent.putExtra("PARAM", "Not Sensitive Info");
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            Toast.makeText(this, "Target activity not found.", Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // *** POINT 4 *** When receiving a result, handle the data carefully and securely.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        if (resultCode != RESULT_OK) return;
        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");
            Toast.makeText(this, String.format("Received result: ¥"%s¥"", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```


4.1.1.3. Creating/Using Partner Activities

Partner activities are Activities that can only be used by specific applications. They are used between cooperating partner companies that want to securely share information and functionality.

There is a risk that a third party application can read an Intent that is used to start the Activity. Therefore it is necessary to make sure that if you are putting sensitive information inside an Intent used to start an Activity that you take countermeasures to make sure that it cannot be read by a malicious third party

Sample code for creating a Partner Activity is shown below.

Points (Creating an Activity):

1. Do not specify taskAffinity.
2. Do not specify launchMode.
3. Do not define the intent filter.
4. Verify the requesting application's certificate through a predefined whitelist.
5. Handle the received intent carefully and securely, even though the intent was sent from a partner application.
6. Only return Information that is granted to be disclosed to a partner application.

Plesese refer to "4.1.3.2 Validating the Requesting Application" for how to validate an application by a white list. Also, please refer to "5.2.1.3 How to verify the hash value of an application's certificate" for how to verify the certificate hash value of a destination application which is specified in the whitelist.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.partneractivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Partner activity -->
        <!-- *** POINT 1 *** Do not specify taskAffinity -->
        <!-- *** POINT 2 *** Do not specify launchMode -->
        <!-- *** POINT 3 *** Do not define the intent filter -->
        <activity
            android:name=".PartnerActivity"
            android:exported="true" />

    </application>
</manifest>
```

PartnerActivity.java

```
package org.jssec.android.activity.partneractivity;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PartnerActivity extends Activity {

    // *** POINT 4 *** Verify the requesting application's certificate through a predefined whitelist.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application org.jssec.android.activity.partneruser.
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of "partner key" in the keystore.
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 4 *** Verify the requesting application's certificate through a predefined whitelist.
        if (!checkPartner(this, getCallingPackage())) {
            Toast.makeText(this,
                    "Requesting application is not a partner application.",
                    Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // *** POINT 5 *** Handle the received intent carefully and securely, even though the intent was sent from a p
artner application.
        // Omitted, since this is a sample. Refer to "3.2 Handling Input Data Carefully and Securely."
        Toast.makeText(this, "Accessed by Partner App", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 6 *** Only return Information that is granted to be disclosed to a partner application.
```



```
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Information for partner applications");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
```

```
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


Sample code for using a Partner Activity is described below.

Points (Using an Activity):

7. Verify if the certificate of the target application has been registered in a whitelist.
8. Do not set the FLAG_ACTIVITY_NEW_TASK flag for the intent that start an activity.
9. Only send information that is granted to be disclosed to a Partner Activity.
10. Use explicit intent to call a Partner Activity.
11. Use startActivityForResult() to call a Partner Activity.
12. Handle the received result data carefully and securely, even though the data comes from a partner application.

Refer to "4.1.3.2 Validating the Requesting Application" for how to validate applications by white list. Also please refer to "5.2.1.3 How to verify the hash value of an application's certificate" for how to verify the certificate hash value of a destination application which is to be specified in a white list.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.partneruser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:name="org.jssec.android.activity.partneruser.PartnerUserActivity"
            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>
</manifest>
```

PartnerUserActivity.java

```
package org.jssec.android.activity.partneruser;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
```

```
public class PartnerUserActivity extends Activity {

    // *** POINT 7 *** Verify if the certificate of a target application has been registered in a white list.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register the certificate hash value of partner application org.jssec.android.activity.partneractivity.
        sWhitelists.add("org.jssec.android.activity.partneractivity", isdebug ?
                // The certificate hash value of "androiddebugkey" is in debug.keystore.
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // The certificate hash value of "my company key" is in the keystore.
                "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    private static final int REQUEST_CODE = 1;

    // Information related the target partner activity
    private static final String TARGET_PACKAGE =  "org.jssec.android.activity.partneractivity";
    private static final String TARGET_ACTIVITY = "org.jssec.android.activity.partneractivity.PartnerActivity";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 7 *** Verify if the certificate of the target application has been registered in the own white li
st.
        if (!checkPartner(this, TARGET_PACKAGE)) {
            Toast.makeText(this, "Target application is not a partner application.", Toast.LENGTH_LONG).show();
            return;
        }

        try {
            // *** POINT 8 *** Do not set the FLAG_ACTIVITY_NEW_TASK flag for the intent that start an activity.
            Intent intent = new Intent();

            // *** POINT 9 *** Only send information that is granted to be disclosed to a Partner Activity.
            intent.putExtra("PARAM", "Info for Partner Apps");

            // *** POINT 10 *** Use explicit intent to call a Partner Activity.
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);

            // *** POINT 11 *** Use startActivityForResult() to call a Partner Activity.
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this, "Target activity not found.", Toast.LENGTH_LONG).show();
        }
    }
```



```
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // *** POINT 12 *** Handle the received data carefully and securely,
            // even though the data comes from a partner application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this,
                    String.format("Received result: ¥"%s¥"", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;
```

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


4.1.1.4. Creating/Using In-house Activities

In-house activities are the Activities which are prohibited to be used by applications other than other in-house applications. They are used in applications developed internally that want to securely share information and functionality.

There is a risk that a third party application can read an Intent that is used to start the Activity. Therefore it is necessary to make sure that if you are putting sensitive information inside an Intent used to start an Activity that you take countermeasures to make sure that it cannot be read by a malicious third party.

Sample code for creating an In-house Activity is shown below.

Points (Creating an Activity):

1. Define an in-house signature permission.
2. Do not specify taskAffinity.
3. Do not specify launchMode.
4. Require the in-house signature permission.
5. Do not define an intent filter.
6. Verify that the in-house signature permission is defined by an in-house application.
7. Handle the received intent carefully and securely, even though the intent was sent from an in-house application.
8. Sensitive information can be returned since the requesting application is in-house.
9. When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.inhouseactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 1 *** Define an in-house signature permission -->
    <permission
        android:name="org.jssec.android.activity.inhouseactivity.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- In-house Activity -->
        <!-- *** POINT 2 *** Do not specify taskAffinity -->
        <!-- *** POINT 3 *** Do not specify launchMode -->
        <!-- *** POINT 4 *** Require the in-house signature permission  -->
        <!-- *** POINT 5 *** Do not define an intent filter -->
        <activity
            android:name="org.jssec.android.activity.inhouseactivity.InhouseActivity"
```

```
            android:exported="true"
            android:permission="org.jssec.android.activity.inhouseactivity.MY_PERMISSION" />
    </application>
</manifest>
```

InhouseActivity.java

```
package org.jssec.android.activity.inhouseactivity;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class InhouseActivity extends Activity {

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.activity.inhouseactivity.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 6 *** Verify that the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "The in-house signature permission is not declared by in-house application.",
                    Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // *** POINT 7 *** Handle the received intent carefully and securely, even though the intent was sent from an
in-house application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("Received param: ¥"%s¥"", param), Toast.LENGTH_LONG).show();
    }
```



```
    public void onReturnResultClick(View view) {

        // *** POINT 8 *** Sensitive information can be returned since the requesting application is in-house.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
```

```
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


*** Point9 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

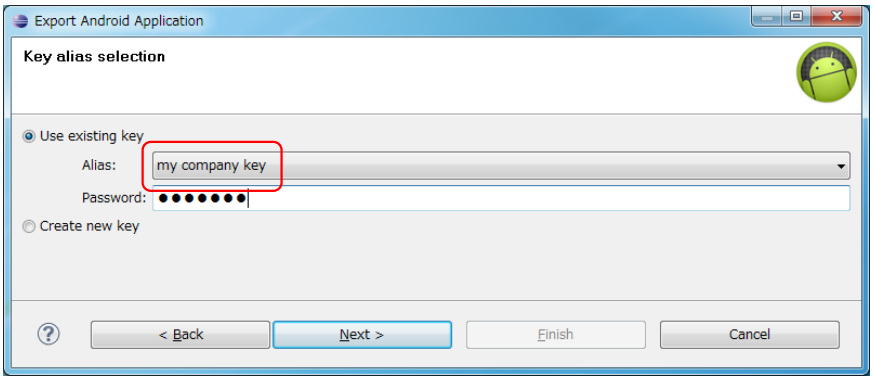


Figure 4.1-2

Sample code for using an In-house Activity is described below.

Points (Using an activity):

10. Declare that you want to use the in-house signature permission.
11. Verify that the in-house signature permission is defined by an in-house application.
12. Verify that the destination application is signed with the in-house certificate.
13. Sensitive information can be sent since the destination application is in-house.
14. Use explicit intents to call an In-house Activity.
15. Handle the received data carefully and securely, even though the data came from an in-house application.
16. When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.inhouseuser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 10 *** Declare to use the in-house signature permission -->
    <uses-permission
        android:name="org.jssec.android.activity.inhouseactivity.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:name="org.jssec.android.activity.inhouseuser.InhouseUserActivity"
            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>
</manifest>
```

InhouseUserActivity.java

```
package org.jssec.android.activity.inhouseuser;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```



```
import android.widget.Toast;

public class InhouseUserActivity extends Activity {

    // Target Activity information
    private static final String TARGET_PACKAGE =  "org.jssec.android.activity.inhouseactivity";
    private static final String TARGET_ACTIVITY = "org.jssec.android.activity.inhouseactivity.InhouseActivity";

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.activity.inhouseactivity.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 11 *** Verify that the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "The in-house signature permission is not declared by in-house application.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // ** POINT 12 *** Verify that the destination application is signed with the in-house certificate.
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this, "Target application is not an in-house application.", Toast.LENGTH_LONG).show();
            return;
        }

        try {
            Intent intent = new Intent();

            // *** POINT 13 *** Sensitive information can be sent since the destination application is in-house.
            intent.putExtra("PARAM", "Sensitive Info");

            // *** POINT 14 *** Use explicit intents to call an In-house Activity.
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
```

```
            Toast.makeText(this, "Target activity not found.", Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // *** POINT 15 *** Handle the received data carefully and securely,
            // even though the data came from an in-house application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this, String.format("Received result: ¥"%s¥"", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
```



```
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 18 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the

destination application.

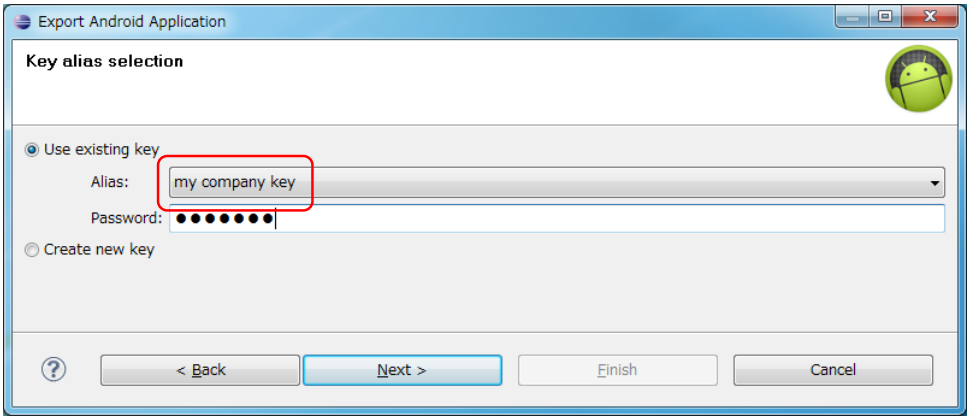


Figure 4.1-3

4.1.2. Rule Book

Be sure to follow the rules below when creating or sending an Intent to an activity.

1. Activities that are Used Only Internally to the Application Must be Set Private (Required)
2. Do Not Specify taskAffinity (Required)
3. Do Not Specify launchMode (Required)
4. Do Not Set the FLAG_ACTIVITY_NEW_TASK Flag for Intents that Start an Activity (Required)
5. Handling the Received Intent Carefully and Securely (Required)
6. Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-House Application (Required)
7. When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)
8. Use the explicit Intents if the destination Activity is predetermined. (Required)
9. Handle the Returned Data from a Requested Activity Carefully and Securely (Required)
10. Verify the Destination Activity if Linking with Another Company's Application (Required)
11. When Providing an Asset Secondhand, the Asset should be Protected with the Same Level of Protection (Required)
12. Sending Sensitive Information Should Be Limited as much as possible (Recommended)

4.1.2.1. Activities that are Used Only Internally to the Application Must be Set Private (Required)

Activities which are only used in a single application are not required to be able to receive any Intents from other applications. Developers often assume that Activities intended to be private will not be attacked but it is necessary to explicitly make these Activities private in order to stop malicious Intents from being received.

AndroidManifest.xml

```
<!-- Private activity -->
<!-- *** POINT 3 *** Explicitly set the exported attribute to false. -->
<activity
    android:name=".PrivateActivity"
    android:label="@string/app_name"
    android:exported="false" />
```

Intent filters should not be set on activities that are only used in a single application. Due to the characteristics of Intent filters, Due to the characteristics of how Intent filters work, even if you intend to send an Intent to a Private Activity internally, if you send the Intent through an Intent filter than you may unintentionally start another Activity. Please see Advanced Topics "4.1.3.1Combining Exported Attributes and Intent Filter Settings (For Activities)" for more details.

AndroidManifest.xml(Not recommended)

```
<!-- Private activity -->
<!-- *** POINT 3 *** Explicitly set the exported attribute to false. -->
<activity
```

```
            android:name=".PictureActivity"
            android:label="@string/picture_name"
            android:exported="false" >
            <intent-filter>
                <action android:name="org.jssec.android.activity.OPEN />
            </intent-filter>
        </activity>
```

4.1.2.2. Do Not Specify taskAffinity (Required)

In Android OS, Activities are managed by tasks. Task names are determined by the affinity that the root Activity has. On the other hand, for Activities other than root Activities, the task to which the Activity belongs is not determined by the Affinity only, but also depends on the Activity's launch mode. Please refer to "4.1.3.4 Root Activity" for more details.

In the default setting, each Activity uses its package name as its affinity. As a result, tasks are allocated according to application, so all Activities in a single application will belong to the same task. To change the task allocation, you can make an explicit declaration for the affinity in the AndroidManifest.xml file or you can set a flag in an Intent sent to an Activity. However, if you change task allocations, there is a risk that another application could read the Intents sent to Activities belonging to another task.

Be sure not to specify android:taskAffinity in the AndroidManifest.xml file and use the default setting keeping the affinity as the package name in order to prevent sensitive information inside sent or received Intents from being read by another application.

Below is an example AndroidManifest.xml file for creating and using Private Activities.

AndroidManifest.xml

```
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Private activity -->
        <!-- *** POINT 1 *** Do not specify taskAffinity -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
```

Please refer to the "Google Android Programming guide"[2], the Google Developers's API Guide "Tasks and Back Stack"[3], "4.1.3.3 Reading Intents Sent to an Activity" and "4.1.3.4 Root Activity" for more

[2] Author Egawa, Fujii, Asano, Fujita, Yamada, Yamaoka, Sano, Takebata, “Google Android Proguramming Guide”, ASCII Media Works, July 2009

[3] http://developer.android.com/guide/components/tasks-and-back-stack.html

details about tasks and affinities.

4.1.2.3. Do Not Specify launchMode (Required)

The Activity launch mode is used to control the settings for creating new tasks and Activity instances when starting an Activity. By default it is set to "standard". In the "standard" setting, new instances are always created when starting an Activity, tasks follow the tasks belonging to the calling Activity, and it is not possible to create a new task. When a new task is created, it is possible for other applications to read the contents of the calling Intent so it is required to use the "standard" Activity launch mode setting when sensitive information is included in an Intent.

The Activity launch mode can be explicitly set in the android:launchMode attribute in the AndroidManifest.xml file, but because of the reason explained above, this should not be set in the Activity declaration and the value should be kept as the default "standard".

AndroidManifest.xml

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >

    <!-- Private activity -->
    <!-- *** POINT 2 *** Do not specify launchMode -->
    <activity
        android:name=".PrivateActivity"
        android:label="@string/app_name"
        android:exported="false" />
</application>
```

Please refer to "4.1.3.3 Reading Intents Sent to an Activity" and "4.1.3.4 Root Activity."

4.1.2.4. Do Not Set the FLAG_ACTIVITY_NEW_TASK Flag for Intents that Start an Activity(Required)

The launch mode of an Activity can be changed when executing startActivity() or startActivityForResult() and in some cases a new task may be generated. Therefore it is necessary to not change the launch mode of Activity during execution.

To change the Activity launch mode, set the Intent flags by using setFlags() or addFlags() and use that Intent as an argument to startActivity() or startActivityForResult(). FLAG_ACTIVITY_NEW_TASK is the flag used to create a new task. When the FLAG_ACTIVITY_NEW_TASK is set, a new task will be created if the called Activity does not exist in the background or foreground.
The FLAG_ACTIVITY_MULTIPLE_TASK flag can be set simultaneously with FLAG_ACTIVITY_NEW_TASK. In this case, a new task will always be created. New tasks may be created with either setting so these should not be set with Intents that handle sensitive information.

Example of sending an intent

```
// *** POINT 6 *** Do not set the FLAG_ACTIVITY_NEW_TASK flag for the intent to start an activity.
 Intent intent = new Intent(this, PrivateActivity.class);
 intent.putExtra("PARAM", "Sensitive Info");
```

```
startActivityForResult(intent, REQUEST_CODE);
```

In addition, you may think that there is a way to prevent the contents of an Intent from being read even if a new task was created by explicitly setting the FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS flag. However, even by using this method, the contents can be read by a third party so you should avoid any usage of FLAG_ACTIVITY_NEW_TASK.

Please refer to "4.1.3.1Combining Exported Attributes and Intent Filter Settings (For Activities)" "4.1.3.3 Reading Intents Sent to an Activity" and "4.1.3.4 Root Activity."

4.1.2.5. Handling the Received Intent Carefully and Securely (Required)

Risks differ depending on the types of Activity, but when processing a received Intent data, the first thing you should do is input validation.

Since Public Activities can receive Intents from untrusted sources, they can be attacked by malware. On the other hand, Private Activities will never receive any Intents from other applications directly, but it is possible that a Public Activity in the targeted application may forward a malicious Intent to a Private Activity so you should not assume that Private Activities cannot receive any malicious input. Since Partner Activities and In-house Activities also have the risk of a malicious intent being forwarded to them as well, it is necessary to perform input validation on these Intents as well.

Please refer to "3.2 Handling Input Data Carefully and Securely"

4.1.2.6. Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-House Application (Required)

Make sure to protect your in-house Activities by defining an in-house signature permission when creating the Activity. Since defining a permission in the AndroidManifest.xml file or declaring a permission request does not provide adequate security, please be sure to refer to "5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission."

4.1.2.7. When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)

When you use setResult() to return data, the reliability of the destination application will depend on the Activity type. When Public Activities are used to return data, the destination may turn out to be malware in which case that information could be used in a malicious way. For Private and In-house Activities, there is not much need to worry about data being returned to be used maliciously because they are being returned to an application you control. Partner Activities are somewhat in the middle.

As above, when returning data from Activities, you need to pay attention to information leakage from

the destination application.

Example of returning data.

```
      public void onReturnResultClick(View view) {

                // *** POINT 6 *** Information that is granted to be disclosed to a partner application can be return
ed.
                Intent intent = new Intent();
                intent.putExtra("RESULT", "Sensitive Info");
                setResult(RESULT_OK, intent);
                finish();
      }
```

4.1.2.8. Use the explicit Intents if the destination Activity is predetermined. (Required)

When using an Activity by implicit Intents, the Activity in which the Intent gets sent to is determined by the Android OS. If the Intent is mistakenly sent to malware then Information leakage can occur. On the other hand, when using an Activity by explicit Intents, only the intended Activity will receive the Intent so this is much safer.

Unless it is absolutely necessary for the user to determine which application's Activity the intent should be sent to, you should use explicit intents and specify the destination in advance.

Using an Activity in the same application by an explicit Intent

```
       Intent intent = new Intent(this, PictureActivity.class);
       intent.putExtra("BARCODE", barcode);
       startActivity(intent);
```

Using other applicaion's Public Activity by an explicit Intent

```
       Intent intent = new Intent();
       intent.setClassName(
           "org.jssec.android.activity.publicactivity",
           "org.jssec.android.activity.publicactivity.PublicActivity");
       startActivity(intent);
```

However, even when using another application's Public Activity by explicit Intents, it is possible that the destination Activity could be malware. This is because even if you limit the destination by package name, it is still possible that a malicious application can fake the same package name as the real application. To eliminate this type of risk, it is necessary to consider using a Partner or In-house.

Please refer to "4.1.3.1Combining Exported Attributes and Intent Filter Settings (For Activities)"

4.1.2.9. Handle the Returned Data from a Requested Activity Carefully and Securely (Required)

While the risks differ slightly according to what type of Activity you accessing, when processing Intent data received as a returned value, you always need to perform input validation on the received data.

Public Activities have to accept returned Intents from untrusted sources so when accessing a Public Activity it is possible that, the returned Intents are actually sent by malware. It is often mistakenly thought that all returned Intents from a Private Activity are safe because they are originating from the same application. However, since it is possible that an intent received from an untrusted source is indirectly forwarded, you should not blindly trust the contents of that Intent. Partner and In-house Activities have a risk somewhat in the middle of Private and Public Activities. Be sure to input validate these Activities as well.

Please refer to "3.2 Handling Input Data Carefully and Securely" for more information.

4.1.2.10. Verify the Destination Activity if Linking with Another Company's Application (Required)

Be sure to sure a whitelist when linking with another company's application. You can do this by saving a copy of the company's certificate hash inside your application and checking it with the certificate hash of the destination application. This will prevent a malicious application from being able to spoof Intents. Please refer to sample code section "4.1.1.3 Creating/Using Partner Activities" for the concrete implementation method. For technical details, please refer to "4.1.3.2 Validating the Requesting Application."

4.1.2.11. When Providing an Asset Secondhand, the Asset should be Protected with the Same Level of Protection (Required)

When an information or function asset, that is protected by a permission, is provided to another application secondhand, you need to make sure that it has the same required permissions needed to access the asset. In the Android OS permission security model, only an application that has been granted proper permissions can directly access a protected asset. However, there is a loophole because an application with permissions to an asset can act as a proxy and allow access to an unprivileged application. Substantially this is the same as redelegating a permission so it is referred to as the "Permission Redelegation" problem. Please refer to "5.2.3.4 Permission Re-delegation Problem."

4.1.2.12. Sending Sensitive Information Should Be Limited as much as possible (Recommended)

You should not send sensitive information to untrusted parties. Even when you are linking with a specific application, there is still a chance that you unintentionally send an Intent to a different application or that a malicious third party can steal your Intents. Please refer to "4.1.3.5 Log Output When using Activities."

You need to consider the risk of information leakage when sending sensitive information to an Activity. You must assume that all data in Intents sent to a Public Activity can be obtained by a malicious third party. In addition, there is a variety of risks of information leakage when sending Intents to Partner or In-house Activities as well depending on the implementation. Even when sending data to Private Activities, there is a risk that the data in the Intent could be leaked through LogCat. Information in the extras part of the Intent is not output to LogCat so it is best to store sensitive information there.

However, not sending sensitive data in the first place is the only perfect solution to prevent information leakage therefore you should limit the amount of sensitive information being sent as much as possible. When it is necessary to send sensitive information, the best practice is to only send to a trusted Activity and to make sure the information cannot be leaked through LogCat.

In addition, sensitive information should never be sent to the root Activity. Root Activities are Activities that are called first when a task is created. For example, the Activity which is launched from launcher is always the root Activity.

Please refer to "4.1.3.3 Reading Intents Sent to an Activity" and "4.1.3.4 Root Activity" for more details on root Activities.

4.1.3. Advanced Topics

4.1.3.1. Combining Exported Attributes and Intent Filter Settings (For Activities)

We have explained how to implement the four types of Activities in this guidebook: Private Activities, Public Activities, Partner Activities, and In-house Activities. The various combinations of permitted settings for each type of exported attribute defined in the AndroidManifest.xml file and the intent-filter elements are defined in the table below. Please verify the compatibility of the exported attribute and intent-filter element with the Activity you are trying to create.

Table 4.1-2

	Value of exported attribute		
	true	False	Not specified
Intent Filter defined	Public, Partner	(Do not Use)	Public, Partner
Intent Filter Not Defined	Public, Partner, In- house	AndroidManifest.xml	Private

The reason why an undefined intent filter and an exported attribute of false should not be used is that there is a loophole in Android's behavior, and because of how Intent filters work, other application's Activities can be called unexpectedly. The following two figures below show this explanation. Figure 4.1-4 is an example of normal behavior in which a Private Activity (Application A) can be called by an implicit Intent only from the same application. The Intent filter (action = "X") is defined to work only inside Application A, so this is the expected behavior.

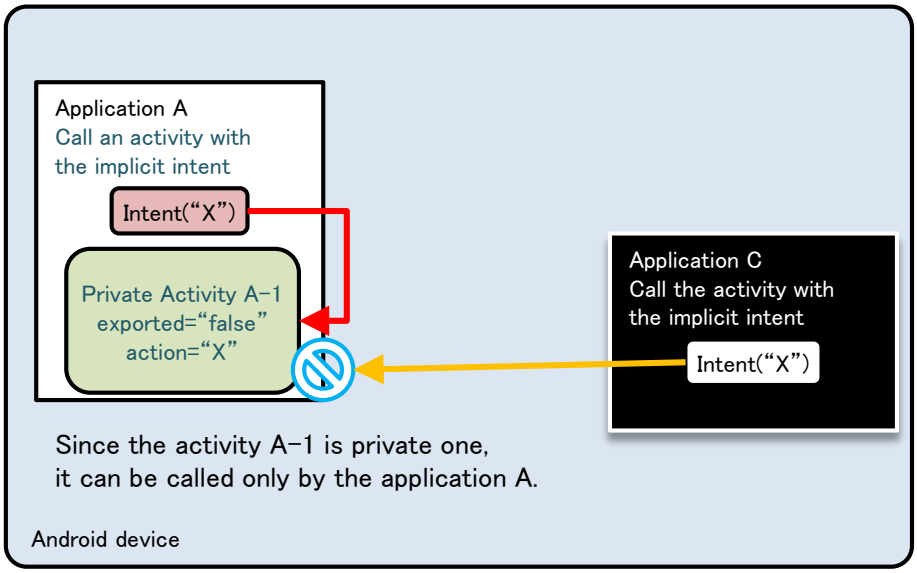


Figure 4.1-4

Figure 4.1-5 below shows a scenario in which the same Intent filter (action="X") is defined in Application B as well as Application A. Application A is trying to call a Private Activity in the same application by sending an implicit Intent, but this time a dialogue box asking the user which application to select is displayed, and the Public Activity B-1 in Application B called by mistake due to

the user selection. Due to this loophole, it is possible that sensitive information can be sent to other applications or application may receive an unexpected retuned value.

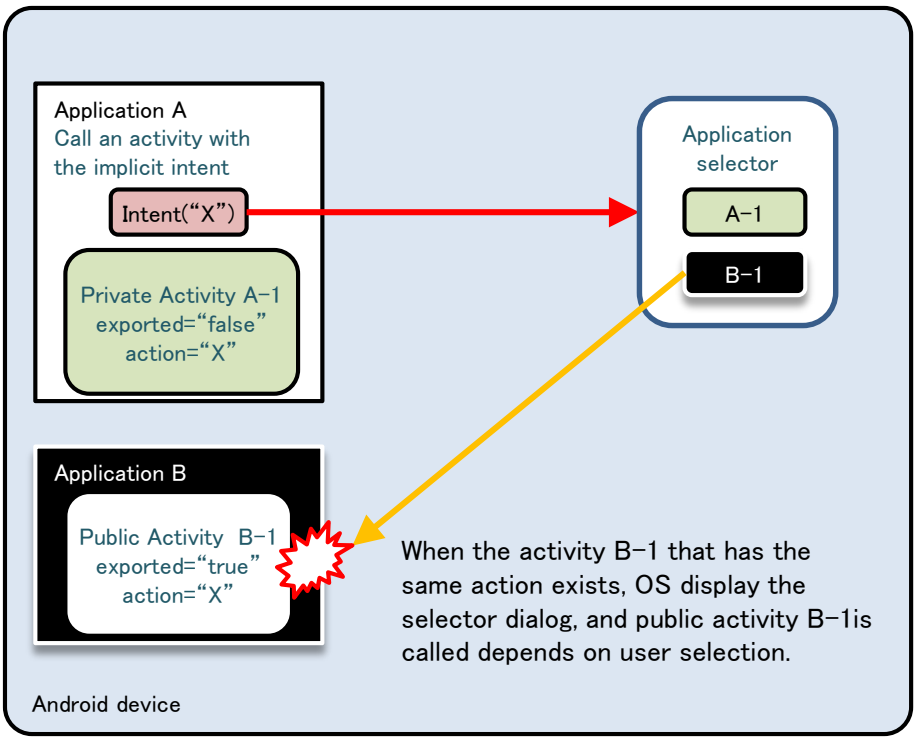


Figure 4.1-5

As shown above, using Intent filters to send implicit Intents to Private Activities may result in unexpected behavior so it is best to avoid this setting. In addition, we have verified that this behavior does not depend on the installation order of Application A and Application B.

4.1.3.2. Validating the Requesting Application

Here we explain the technical information about how to implement a Partner Activity. Partner applications permit that only particular applications which are registered in a whitelist are allowed access and all other applications are denied. Because applications other than in-house applications also need access permission, we cannot use signature permissions for access control.

Simply speaking, we want to validate the application trying to use the Partner Activity by checking if it is registered in a predefined whitelist and allow access if it is and deny access if it is not. Application validation is done by obtaining the certificate from the application requesting access and comparing its hash with the one in the whitelist.

Some developers may think that it is sufficient to just compare the package name without obtaining the certificate, however, it is easy to spoof the package name of a legitimate application so this is not a good method to check for authenticity. Arbitrarily assignable values should not be used for authentication. On the other hand, because only the application developer has the developer key for signing its certificate, this is a better method for identification. Since the certificate cannot be easily

spoofed, unless a malicious third party can steal the developer key, there is a very small chance that malicious application will be trusted. While it is possible to store the entire certificate in the whitelist, it is sufficient to only store the SHA-256 hash value in order to minimize the file size.

There are two restrictions for using this method.

- The requesting application has to use startActivityForResult() instead of startActivity().
- The requesting application can only call from an Activity.

The second restriction is the restriction imposed as a result of the first restriction, so technically there is only a single restriction.
This restriction occurs due to the restriction of Activity.getCallingPackage() which gets the package name of the calling application. Activity.getCallingPackage() returns the package name of source (requesting) application only in case it is called by startActivityForResult(), but unfortunately, when it is called by startActivity(), it only returns null. Because of this, when using the method explained here, the source (requesting) application needs to use startActivityForResult() even if it does not need to obtain a return value. In addition, startActivityForResult() can be used only in Activity classes, so the source (requester) is limited to Activities.

PartnerActivity.java

```
package org.jssec.android.activity.partneractivity;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PartnerActivity extends Activity {

    // *** POINT 4 *** Verify the requesting application's certificate through a predefined whitelist.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application org.jssec.android.activity.partneruser.
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of "partner key" in the keystore.
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
```



```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 4 *** Verify the requesting application's certificate through a predefined whitelist.
        if (!checkPartner(this, getCallingPackage())) {
            Toast.makeText(this,
                    "Requesting application is not a partner application.",
                    Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // *** POINT 5 *** Handle the received intent carefully and securely, even though the intent was sent from a p
artner application.
        // Omitted, since this is a sample. Refer to "3.2 Handling Input Data Carefully and Securely."
        Toast.makeText(this, "Accessed by Partner App", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 6 *** Only return Information that is granted to be disclosed to a partner application.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Information for partner applications");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);
```

```
        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


4.1.3.3. Reading Intents Sent to an Activity

Intents that are sent to the task's root Activity are added to the task history. A root Activity is the first Activity started in a task. It is possible for any application to read the Intents added to the task history by using the ActivityManager class.

Sample code for reading the task history from an application is shown below. To browse the task history, specify the GET_TASKS permission in the AndroidManifest.xml file.

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.intent.maliciousactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MaliciousActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <!-- Use GET_TASKS Permission -->
    <uses-permission android:name="android.permission.GET_TASKS" />
</manifest>
```

MaliciousActivity.java

```
package org.jssec.android.intent.maliciousactivity;

import java.util.List;

import android.app.Activity;
import android.app.ActivityManager;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

public class MaliciousActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.malicious_activity);

        // Get am ActivityManager instance.
        ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
        // Get 100 recent task info.
        List<ActivityManager.RecentTaskInfo> list = activityManager
                .getRecentTasks(100, ActivityManager.RECENT_WITH_EXCLUDED);
        for (ActivityManager.RecentTaskInfo r : list) {
            // Get Intent sent to root Activity and Log it.
            Intent intent = r.baseIntent;
            Log.v("baseIntent", intent.toString());
        }
    }
}
```

You can obtain specified entries of the task history by using the getRecentTasks() function of the AcitivityManager class. Information about each task is stored in an instance of the ActivityManager.RecentTaskInfo class, but Intents that were sent to the task's root Activity are stored in its member variable baseIntent. Since the root Activity is the Activity which was started when the task was created, please be sure to fulfill the following two conditions when calling an Activity.

- A new task is created when the Activity is called.
- The called Activity is the task's root Activity which already exists in the background or foreground.

4.1.3.4. Root Activity

The root Activity is the Activity which is the starting point of a task. In other words, this is the Activity which was launched when task was created. For example, when the default Activity is launched by launcher, this Activity will be the root Activity. According to the Android specifications, the contents of Intents sent to the root Activity can be read from arbitrary applications. So, it is necessary to take countermeasures not to send sensitive information to the root Activity. In this guidebook, the following three rules have been made to avoid a called Activity to become root Activity.

- taskAffinity should not be specified.
- launchMode should not be specified.
- The FLAG_ACTIVITY_NEW_TASK flag should not be set in an Intent sent to an Activity.

We consider the situations that an Activity can become the root Activity below. A called Activity becoming a root Activity depends on the following.

- The launch mode of the called Activity
- The task of a called Activity and its launch mode

First of all, let me explain the "Launch mode of called Activity." Launch mode of Activity can be set by writing android:launchMode in AndroidManifest.xml. When it's not written, it's considered as

"standard". In addition, launch mode can be also changed by a flag to set to Intent. Flag "FLAG_ACTIVITY_NEW_TASK" launches Activity by "singleTask" mode.

The launch modes that can be specified are as per below. I'll explain about the relation with the root activity, mainly.

standard

Activity which is called by this mode won't be root, and it belongs to the caller side task. Every time it's called, Instance of Activity is to be generated.

singleTop

This launch mode is the same as "standard", except for that the instance is not generated when launching an Activity which is displayed in most front side of foreground task.

singleTask

This launch mode determines the task to which the activity would be belonging by Affinity value. When task which is matched with Activity's affinity doesn't exist either in background or in foreground, a new task is generated along with Activity's instance. When task exists, neither of them is to be generated. In the former one, the launched Activity's Instance becomes root.

singleInstance

Same as "singleTask", but following point is different. Only root Activity can belongs to the newly generated task. So instance of Activity which was launched by this mode is always root activity. Now, we need to pay attention to the case that the class name of called Activity and the class name of Activity which is included in a task are different although the task which has the same name of called Activity's affinity already exists.

From as above, we can get to know that Activity which was launched by "singleTask" or "singleInstance" has the possibility to become root. In order to secure the application's safety, it should not be launched by these modes.

Next, I'll explain about "Task of the called Activity and its launch mode". Even if Activity is called by "standard" mode, it becomes root Activity in some cases depends on the task state to which Activity belongs.

For example, think about the case that called Activity's task has being run already in background.
The problem here is the case that Activity Instance of the task is launched by singleInstance". When the affinity of Activity which was called by "standard" is same with the task, new task is to be generated by the restriction of existing "singleInstance" Activity. However, when class name of each Activity is same, task is not generated and existing activity Instance is to be used. In any cases, that called Activity becomes root Activity.

As per above, the conditions that root Activity is called are complicated, for example it depends on the state of execution. So when developing applications, it's better to contrive that Activity is called

by "standard".

As an example of that Intent which is sent to Private Activity is read out form other application, the sample code shows the case that caller side Activity of private Activity is launched by "singleInstance" mode. In this sample code, private activity is launched by "standard" mode, but this private Activity becomes root Activity of new task due the "singleInstance" condition of caller side Activity. At this moment, sensitive information that is sent to Private Activity is recorded task history, so it can be read out from other applications. FYI, both caller side Activity and Private Activity have the same affinity.

AndroidManifest.xml(Not recommended)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.singleinstanceactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Set the launchMode of the root Activity to "singleInstance". -->
        <!-- Do not use taskAffinity -->
        <activity
            android:name="org.jssec.android.activity.singleinstanceactivity.PrivateUserActivity"
            android:label="@string/app_name"
            android:launchMode="singleInstance" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Private activity -->
        <!-- Set the launchMode to "standard." -->
        <!-- Do not use taskAffinity -->
        <activity
            android:name="org.jssec.android.activity.singleinstanceactivity.PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
</manifest>
```

Private Activity only returns the results to the received Intent.

PrivateActivity.java

```
package org.jssec.android.activity.singleinstanceactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```



```
import android.widget.Toast;

public class PrivateActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.private_activity);

        // Handle intent securely, even though the intent sent from the same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("Received param: ¥"%s¥"", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

In caller side of Private Activity, Private Activity is launched by "standard" mode without setting flag to Intent.

PrivateUserActivity.java

```
package org.jssec.android.activity.singleinstanceactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);
    }

    public void onUseActivityClick(View view) {

        // Start the Private Activity with "standard" lanchMode.
        Intent intent = new Intent(this, PrivateActivity.class);
        intent.putExtra("PARAM", "Sensitive Info");

        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
```

```
        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // Handle received result data carefully and securely,
            // even though the data came from the Activity in the same application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this, String.format("Received result: ¥"%s¥"", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

4.1.3.5. Log Output When using Activities

When using an activity, the contents of intent are output to LogCat by ActivityManager. The following contents are to be output to LogCat, so in this case, sensitive information should not be included here.

- Destination Package name
- Destination Class name
- URI which is set by Intent#setData()

For example, when an application sent mails, the mail address is unfortunately outputted to LogCat if the application would specify the mail address to URI. So, better to send by setting Extras.

When sending a mail as below, mail address is shown to the logCat.

MainActivity.java

```
        // URI is output to the LogCat.
        Uri uri = Uri.parse("mailtoest@gmail.com");
        Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
        startActivity(intent);
```

When using Extras, mail address is no more shown to the logCat.

MainActivity.java

```
        // Contents which was set to Extra, is not output to the LogCat.
        Uri uri = Uri.parse("mailto:");
        Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
        intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"test@gmail.com"});
        startActivity(intent);
```


4.2.1. Sample Code

Creating Broadcast Receiver is required to receive Broadcast. Risks and countermeasures of using Broadcast Receiver differ depending on the type of the received Broadcast.
You can find your Broadcast Receiver in the following judgment flow. The receiving applications cannot check the package names of Broadcast-sending applications that are necessary for linking with the partners. As a result, Broadcast Receiver for the partners cannot be created.

Table 4.2-1 Definition of broadcast receiver types

Type	Definition
Private broadcast receiver	A broadcast receiver that can receive broadcasts only from the same application, therefore is the safest broadcast receiver
Public broadcast receiver	A broadcast receiver that can receive broadcasts from an unspecified large number of applications
In-house broadcast receiver	A broadcast receiver that can receive broadcasts only from other In-house applications

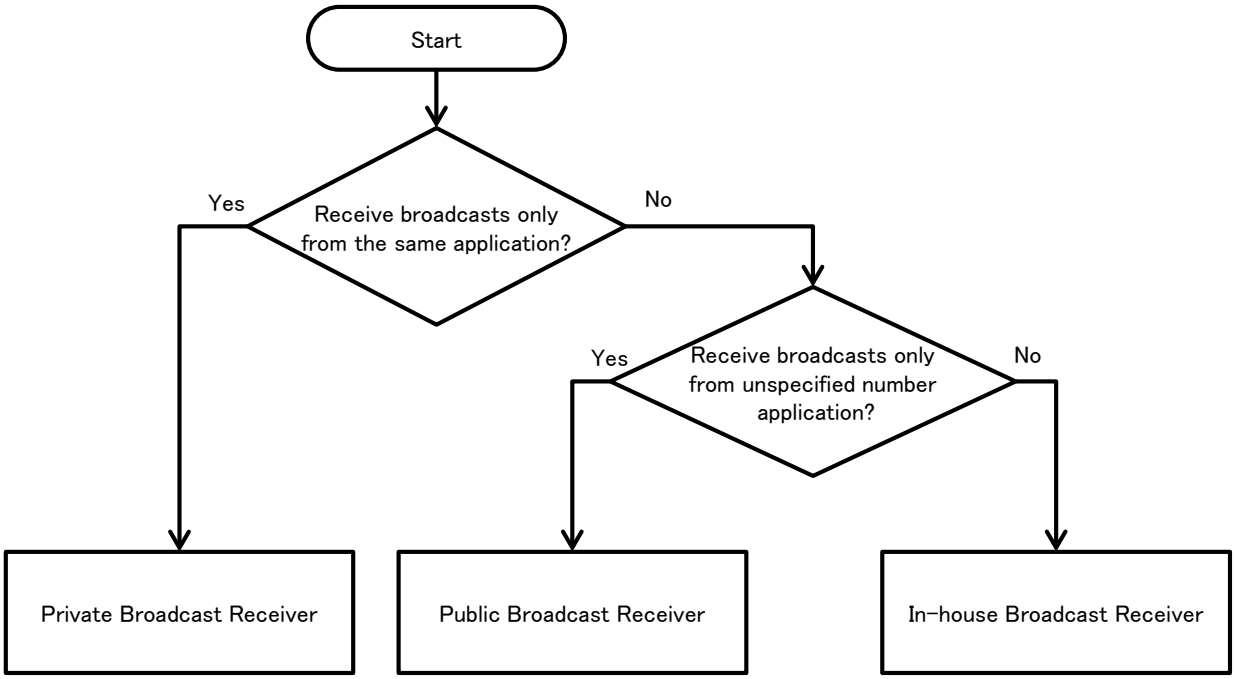


Figure 4.2-1

In addition, Broadcast Receiver can be divided into 2 types based on the definition methods, Static Broadcast Receiver and Dynamic Broadcast Receiver. The differences between them can be found in the following figure. In the sample code, an implementation method for each type is shown. The implementation method for sending applications is also described because the countermeasure for sending information is determined depending on the receivers.

Table 4.2-2

	Definition method	Characteristic
Static Broadcast Receiver	Define by writing <receiver> elements in AndroidManifest.xml	● There is a restriction that some Broadcasts (e.g. ACTION_BATTERY_CHANGED) sent by system cannot be received. ● Broadcast can be received from application's initial boot till uninstallation.
Dynamic Broadcast Receiver	By calling registerReceiver() and unregisterReceiver() in a program, register/unregister Broadcast Receiver dynamically.	● Broadcasts which cannot be received by static Broadcast Receiver can be received. ● The period of receiving Broadcasts can be controlled by the program. For example, Broadcasts can be received only while Activity is on the front side. ● Private Broadcast Receiver cannot be created.

4.2.1.1. Private Broadcast Receiver - Receiving/Sending Broadcasts

Private Broadcast Receiver is the safest Broadcast Receiver because only Broadcasts sent from within the application can be received. Dynamic Broadcast Receiver cannot be registered as Private, so Private Broadcast Receiver consists of only Static Broadcast Receivers.

Points (Receiving Broadcasts):

1. Explicitly set the exported attribute to false.
2. Handle the received intent carefully and securely, even though the intent was sent from within the same application.
3. Sensitive information can be sent as the returned results since the requests come from within the same application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.privatereceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Private Broadcast Receiver -->
        <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
        <receiver
            android:name=".PrivateReceiver"
            android:exported="false" />

        <activity
            android:name=".PrivateSenderActivity"
            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>
</manifest>
```

PrivateReceiver.java

```
package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PrivateReceiver extends BroadcastReceiver {
```

```
    @Override
    public void onReceive(Context context, Intent intent) {

        // *** POINT 2 *** Handle the received intent carefully and securely,
        // even though the intent was sent from within the same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(context,
                String.format("Received param: ¥"%s¥"", param),
                Toast.LENGTH_SHORT).show();

        // *** POINT 3 *** Sensitive information can be sent as the returned results since the requests come from with
in the same application.
        setResultCode(Activity.RESULT_OK);
        setResultData("Sensitive Info from Receiver");
        abortBroadcast();
    }
}
```


The sample code for sending Broadcasts to private Broadcast Receiver is shown below. Please pay attention that Sticky cannot be used here though the method of sending Broadcasts to private Broadcast Receiver is said to be safe from the security point of view.

Points (Sending Broadcasts):

4. Use the explicit Intent with class specified to call a receiver within the same application.
5. Sensitive information can be sent since the destination Receiver is within the same application.
6. Handle the received result data carefully and securely, even though the data came from the Receiver within the same application.

PrivateSenderActivity.java

```
package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateSenderActivity extends Activity {

    public void onSendNormalClick(View view) {
        // *** POINT 4 *** Use the explicit Intent with class specified to call a receiver within the same application
.
        Intent intent = new Intent(this, PrivateReceiver.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination Receiver is within the same applica
tion.
        intent.putExtra("PARAM", "Sensitive Info from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // *** POINT 4 *** Use the explicit Intent with class specified to call a receiver within the same application
.
        Intent intent = new Intent(this, PrivateReceiver.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination Receiver is within the same applica
tion.
        intent.putExtra("PARAM", "Sensitive Info from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            // *** POINT 6 *** Handle the received result data carefully and securely,
            // even though the data came from the Receiver within the same application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String data = getResultData();
            PrivateSenderActivity.this.logLine(
                    String.format("Received result: ¥"%s¥"", data));
        }
```

```
    };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


4.2.1.2. Public Broadcast Receiver – Receiving/Sending Broadcasts

Public Broadcast Receiver is the Broadcast Receiver that can receive Broadcasts from unspecified large number of applications, so it's necessary to pay attention that it may receive Broadcasts from malware.

Points (Receiving Broadcasts):

1. Handle the received Intent carefully and securely.
2. When returning a result, do not include sensitive information..

Public Receiver which is the sample code for public Broadcast Receiver can be used both in static Broadcast Receiver and Dynamic Broadcast Receiver.

PublicReceiver.java

```
package org.jssec.android.broadcast.publicreceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PublicReceiver extends BroadcastReceiver {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    public boolean isDynamic = false;
    private String getName() {
        return isDynamic ? "Public Dynamic Broadcast Receiver" : "Public Static Broadcast Receiver";
    }

    @Override
    public void onReceive(Context context, Intent intent) {

        // *** POINT 1 *** Handle the received Intent carefully and securely.
        // Since this is a public broadcast receiver, the requesting application may be malware.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        if (MY_BROADCAST_PUBLIC.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                    String.format("%s:¥nReceived param: ¥"%s¥"", getName(), param),
                    Toast.LENGTH_SHORT).show();
        }

        // *** POINT 2 *** When returning a result, do not include sensitive information.
        // Since this is a public broadcast receiver, the requesting application may be malware.
        // If no problem when the information is taken by malware, it can be returned as result.
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("Not Sensitive Info from %s", getName()));
        abortBroadcast();
    }
}
```

Static Broadcast Receive is defined in AndroidManifest.xm.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.publicreceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Public Static Broadcast Receiver -->
        <receiver
            android:name=".PublicReceiver"
            android:exported="true" >
            <intent-filter>
                <action android:name="org.jssec.android.broadcast.MY_BROADCAST_PUBLIC" />
            </intent-filter>
        </receiver>

        <service
            android:name=".DynamicReceiverService"
            android:exported="false" />

        <activity
            android:name=".PublicReceiverActivity"
            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>
</manifest>
```

In Dynamic Broadcast Receiver, registration/unregistration is executed by calling registerReceiver() or unregisterReceiver() in the program. In order to execute registration/unregistration by button operations, the button is allocated on PublicReceiverActivity. Since the scope of Dynamic Broadcast Receiver Instance is longer than PublicReceiverActivity, it cannot be kept as the member variable of PublicReceiverActivity. In this case, keep the Dynamic Broadcast Receiver Instance as the member variable of DynamicReceiverService, and then start/end DynamicReceiverService from PublicReceiverActivity to register/unregister Dynamic Broadcast Receiver indirectly.

DynamicReceiverService.java

```
package org.jssec.android.broadcast.publicreceiver;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.widget.Toast;
```



```
public class DynamicReceiverService extends Service {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    private PublicReceiver mReceiver;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        // Register Public Dynamic Broadcast Receiver.
        mReceiver = new PublicReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_PUBLIC);
        filter.setPriority(1);  // Prioritize Dynamic Broadcast Receiver, rather than Static Broadcast Receiver.
        registerReceiver(mReceiver, filter);
        Toast.makeText(this,
                "Registered Dynamic Broadcast Receiver.",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        // Unregister Public Dynamic Broadcast Receiver.
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                "Unregistered Dynamic Broadcast Receiver.",
                Toast.LENGTH_SHORT).show();
    }
}
```

PublicReceiverActivity.java

```
package org.jssec.android.broadcast.publicreceiver;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class PublicReceiverActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
```

```
    public void onRegisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        startService(intent);
    }

    public void onUnregisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        stopService(intent);
    }
}
```


Next, the sample code for sending Broadcasts to public Broadcast Receiver is shown. When sending Broadcasts to public Broadcast Receiver, it's necessary to pay attention that Broadcasts can be received by malware.

Points (Sending Broadcasts):

3. Do not send sensitive information.
4. When receiving a result, handle the result data carefully and securely.

PublicSenderActivity.java

```
package org.jssec.android.broadcast.publicsender;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicSenderActivity extends Activity {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    public void onSendNormalClick(View view) {
        // *** POINT 3 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // *** POINT 3 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    public void onSendStickyClick(View view) {
        // *** POINT 3 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        sendStickyBroadcast(intent);
    }

    public void onSendStickyOrderedClick(View view) {
        // *** POINT 3 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        sendStickyOrderedBroadcast(intent, mResultReceiver, null, 0, null, null);
    }

    public void onRemoveStickyClick(View view) {
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        removeStickyBroadcast(intent);
    }
```

```
    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            // *** POINT 4 *** When receiving a result, handle the result data carefully and securely.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String data = getResultData();
            PublicSenderActivity.this.logLine(
                    String.format("Received result: ¥"%s¥"", data));
        }
    };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


4.2.1.3. In-house Broadcast Receiver - Receiving/Sending Broadcasts

In-house Broadcast Receiver is the Broadcast Receiver that will never receive any Broadcasts sent from other than in-house applications. It consists of several in-house applications, and it's used to protect the information or functions that in-house application handles.

Points (Receiving Broadcasts):

1. Define an in-house signature permission to receive Broadcasts.
2. Declare to use the in-house signature permission to receive results.
3. Require the in-house signature permission by the Static Broadcast Receiver definition.
4. Require the in-house signature permission to register Dynamic Broadcast Receiver.
5. Verify that the in-house signature permission is defined by an in-house application.
6. Handle the received intent carefully and securely, even though the Broadcast was sent from an in-house application.
7. Sensitive information can be returned since the requesting application is in-house.
8. When Exporting an APK from Eclipse, sign the APK with the same developer key as the sending application.

In-house Receiver which is a sample code of in-house Broadcast Receiver is to be used both in Static Broadcast Receiver and Dynamic Broadcast Receiver.

InhouseReceiver.java

```
package org.jssec.android.broadcast.inhousereceiver;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class InhouseReceiver extends BroadcastReceiver {

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }
```

```
    private static final String MY_BROADCAST_INHOUSE =
        "org.jssec.android.broadcast.MY_BROADCAST_INHOUSE";

    public boolean isDynamic = false;
    private String getName() {
        return isDynamic ? "In-house Dynamic Broadcast Receiver" : "In-house Static Broadcast Receiver";
    }

    @Override
    public void onReceive(Context context, Intent intent) {

        // *** POINT 5 *** Verify that the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(context, MY_PERMISSION, myCertHash(context))) {
            Toast.makeText(context, "The in-house signature permission is not declared by in-house application.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 6 *** Handle the received intent carefully and securely,
        // even though the Broadcast was sent from an in-house application..
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        if (MY_BROADCAST_INHOUSE.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                    String.format("%s:¥nReceived param: ¥"%s¥"", getName(), param),
                    Toast.LENGTH_SHORT).show();
        }

        // *** POINT 7 *** Sensitive information can be returned since the requesting application is in-house.
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("Sensitive Info from %s", getName()));
        abortBroadcast();
    }
}
```

Static Broadcast Receiver is to be defined in AndroidManifest.xml.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.inhousereceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 1 *** Define an in-house signature permission to receive Broadcasts -->
    <permission
        android:name="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- *** POINT 2 *** Declare to use the in-house signature permission to receive results. -->
    <uses-permission
        android:name="org.jssec.android.broadcast.inhousesender.MY_PERMISSION" />

    <application
```



```
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- *** POINT 3 *** Require the in-house signature permission by the Static Broadcast Receiver definition. -
->
        <receiver
            android:name=".InhouseReceiver"
            android:permission="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION">
            <intent-filter>
                <action android:name="org.jssec.android.broadcast.MY_BROADCAST_INHOUSE" />
            </intent-filter>
        </receiver>

        <service
            android:name=".DynamicReceiverService"
            android:exported="false" />

        <activity
            android:name=".InhouseReceiverActivity"
            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>

</manifest>
```

Dynamic Broadcast Receiver executes registration/unregistration by calling registerReceiver() or unregisterReceiver() in the program. In order to execute registration/unregistration by the button operations, the button is arranged on InhouseReceiverActivity. Since the scope of Dynamic Broadcast Receiver Instance is longer than InhouseReceiverActivity, it cannot be kept as the member variable of InhouseReceiverActivity. So, keep Dynamic Broadcast Receiver Instance as the member variable of DynamicReceiverService, and then start/end DynamicReceiverService from InhouseReceiverActivity to register/unregister Dynamic Broadcast Receiver indirectly.

InhouseReceiverActivity.java

```
package org.jssec.android.broadcast.inhousereceiver;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class InhouseReceiverActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onRegisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        startService(intent);
```

```
    }

    public void onUnregisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        stopService(intent);
    }}
```

DynamicReceiverService.java

```
package org.jssec.android.broadcast.inhousereceiver;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.widget.Toast;

public class DynamicReceiverService extends Service {

    private static final String MY_BROADCAST_INHOUSE =
        "org.jssec.android.broadcast.MY_BROADCAST_INHOUSE";

    private InhouseReceiver mReceiver;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        mReceiver = new InhouseReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_INHOUSE);
        filter.setPriority(1);  // Prioritize Dynamic Broadcast Receiver, rather than Static Broadcast Receiver.

        // *** POINT 4 *** When registering a dynamic broadcast receiver, require the in-house signature permission.
        registerReceiver(mReceiver, filter, "org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION", null);

        Toast.makeText(this,
                "Registered Dynamic Broadcast Receiver.",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                "Unregistered Dynamic Broadcast Receiver.",
                Toast.LENGTH_SHORT).show();
    }
}
```


SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
```

```
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


*** Point 8 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the sending application.

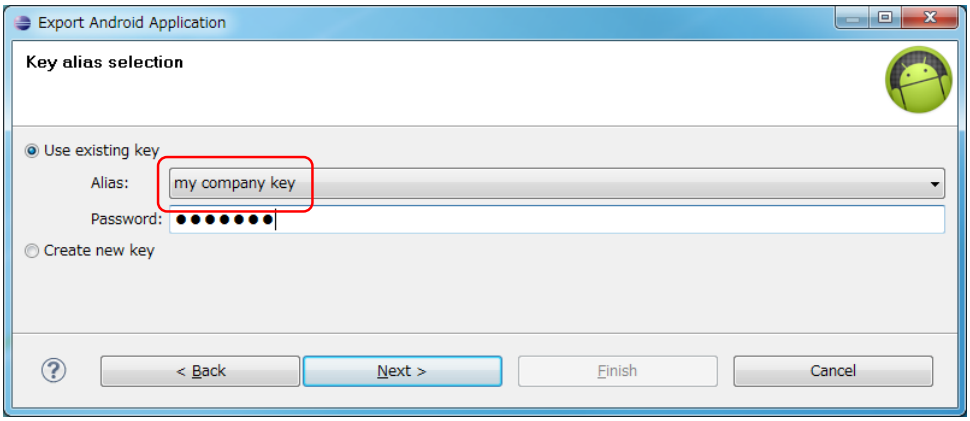


Figure 4.2-2

Next, the sample code for sending Broadcasts to in-house Broadcast Receiver is shown. When sending Broadcasts to in-house Broadcast Receiver, it's necessary to require In-house-defined Signature Permission of Broadcast Receiver side. So it's necessary to pay attention that there is a restriction that Sticky cannot be used.

Points (Sending Broadcasts):

9. Define an in-house signature permission to receive results.
10. Declare to use the in-house signature permission to receive Broadcasts.
11. Verify that the in-house signature permission is defined by an in-house application.
12. Sensitive information can be returned since the requesting application is the in-house one.
13. Require the in-house signature permission of Receivers.
14. Handle the received result data carefully and securely.
15. When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.inhousesender"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.BROADCAST_STICKY"/>

    <!-- *** POINT 9 *** Define an in-house signature permission to receive results. -->
    <permission
        android:name="org.jssec.android.broadcast.inhousesender.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- *** POINT 10 *** Declare to use the in-house signature permission to receive Broadcasts. -->
    <uses-permission
        android:name="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name="org.jssec.android.broadcast.inhousesender.InhouseSenderActivity"
            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>
</manifest>
```

InhouseSenderActivity.java

```
package org.jssec.android.broadcast.inhousesender;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;
```



```
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class InhouseSenderActivity extends Activity {

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.broadcast.inhousesender.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final String MY_BROADCAST_INHOUSE =
        "org.jssec.android.broadcast.MY_BROADCAST_INHOUSE";

    public void onSendNormalClick(View view) {

        // *** POINT 11 *** Verify that the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "The in-house signature permission is not declared by in-house application.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 12 *** Sensitive information can be returned since the requesting application is in-house.
        Intent intent = new Intent(MY_BROADCAST_INHOUSE);
        intent.putExtra("PARAM", "Sensitive Info from Sender");

        // *** POINT 13 *** Require the in-house signature permission to limit receivers.
        sendBroadcast(intent, "org.jssec.android.broadcast.inhousesender.MY_PERMISSION");
    }

    public void onSendOrderedClick(View view) {

        // *** POINT 11 *** Verify that the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "The in-house signature permission is not declared by in-house application.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 12 *** Sensitive information can be returned since the requesting application is in-house.
```

```
        Intent intent = new Intent(MY_BROADCAST_INHOUSE);
        intent.putExtra("PARAM", "Sensitive Info from Sender");

        // *** POINT 13 *** Require the in-house signature permission to limit receivers.
        sendOrderedBroadcast(intent, "org.jssec.android.broadcast.inhousesender.MY_PERMISSION",
                mResultReceiver, null, 0, null, null);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            // *** POINT 14 *** Handle the received result data carefully and securely,
            // even though the data came from an in-house application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String data = getResultData();
            InhouseSenderActivity.this.logLine(String.format("Received result: ¥"%s¥"", data));
        }
    };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
```



```
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
```

```
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 15 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

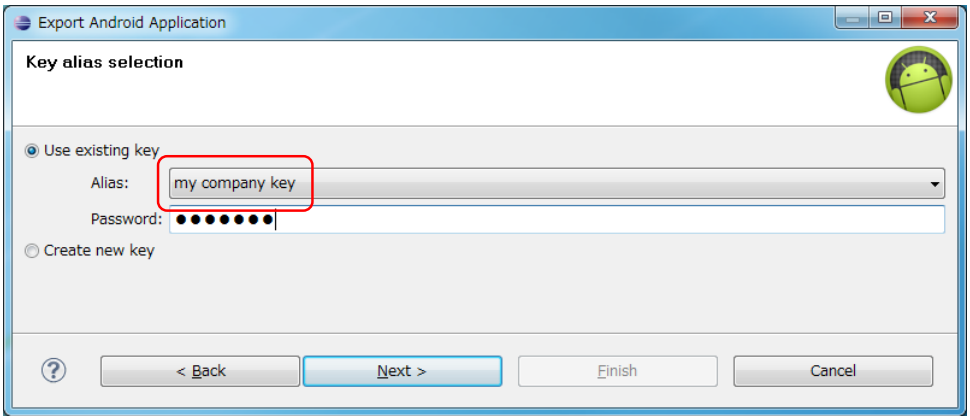


Figure 4.2-3

4.2.2. Rule Book

Follow the rules below to Send or receive Broadcasts.

1.	Broadcast Receiver that Is Used Only in an Application Must Be Set as Private	(Required)
2.	Handle the Received Intent Carefully and Securely	(Required)
3.	Use the In-house Defined Signature Permission after Verifying that it's Defined by an In-house Application	(Required)
4.	When Returning a Result Information, Pay Attention to the Result Information Leakage from the Destination Application	(Required)
5.	When Sending Sensitive Information with a Broadcast, Limit the Receivable Receiver	(Required)
6.	Sensitive Information Must Not Be Included in the Sticky Broadcast	(Required)
7.	Pay Attention that the Ordered Broadcast without Specifying the receiverPermission May Not Be Delivered	(Required)
8.	Handle the Returned Result Data from the Broadcast Receiver Carefully and Securely	(Required)
9.	When Providing an Asset Secondarily, the Asset should be protected with the Same Protection Level	(Required)

4.2.2.1. Broadcast Receiver that Is Used Only in an Application Must Be Set as Private (Required)

Broadcast Receiver which is used only in the application should be set as private to avoid from receiving any Broadcasts from other applications unexpectedly. It will prevent the application function abuse or the abnormal behaviors.

Receiver used only within the same application should not be designed with setting Intent-filter. Because of the Intent-filter characteristics, a public Receiver of other application may be called unexpectedly by calling through Intent-filter even though a private Receiver within the same application is to be called.

AndroidManifest.xml(Not recommended)

```
<!-- Private Broadcast Receiver -->
<!-- *** POINT 1 *** Set the exported attribute to false explicitly. -->
<receiver
    android:name=".PrivateReceiver"
    android:exported="false" >
    <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_ACTION" />
    </intent-filter>
</receiver>
```

Please refer to "4.2.3.1 Combinations of the exported Attribute and the Intent-filter setting (For Receiver)."

4.2.2.2. Handle the Received Intent Carefully and Securely (Required)

Though risks are different depending on the types of the Broadcast Receiver, firstly verify the safety

of Intent when processing received Intent data.

Since Public Broadcast Receiver receives the Intents from unspecified large number of applications, it may receive malware's attacking Intents. Private Broadcast Receiver will never receive any Intent from other applications directly, but Intent data which a public Component received from other applications may be forwarded to Private Broadcast Receiver. So don't think that the received Intent is totally safe without any qualification. In-house Broadcast Receivers have some degree of the risks, so it also needs to verify the safety of the received Intents.

Please refer to "3.2 Handling Input Data Carefully and Securely"

4.2.2.3. Use the In-house Defined Signature Permission after Verifying that it's Defined by an In-house Application (Required)

In-house Broadcast Receiver which receives only Broadcasts sent by an In-house application should be protected by in-house-defined Signature Permission. Permission definition/Permission request declarations in AndroidManifest.xml are not enough to protecting, so please refer to "5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission." ending Broadcasts by specifying in-house-defined Signature Permission to receiverPermission parameter requires verification in the same way.

4.2.2.4. When Returning a Result Information, Pay Attention to the Result Information Leakage from the Destination Application (Required)

The Reliability of the application which returns result information by setResult() varies depending on the types of the Broadcast Receiver. In case of Public Broadcast Receiver, the destination application may be malware, and there may be a risk that the result information is used maliciously. In case of Private Broadcast Receiver and In-house Broadcast Receiver, the result destination is In-house developed application, so no need to mind the result information handling.

Need to pay attention to the result information leakage from the destination application when result information is returned from Broadcast Receivers as above.

4.2.2.5. When Sending Sensitive Information with a Broadcast, Limit the Receivable Receiver (Required)

Broadcast is the created system to broadcast information to unspecified large number of applications or notify them of the timing at once. So, broadcasting sensitive information requires the careful designing for preventing the illicit obtainment of the information by malware.

For broadcasting sensitive information, only reliable Broadcast Receiver can receive it, and other Broadcast Receivers cannot. The following are some examples of Broadcast sending methods.

- The method is to fix the address by Broadcast-sending with an explicit Intent for sending

Broadcasts to the intended reliable Broadcast Receivers only. There are 2 patterns in this method.

- When it's addressed to a Broadcast Receiver within the same application, specify the address by Intent#setClass(Context, Class). Refer to sample code section "4.2.1.1 Private Broadcast Receiver - Receiving/Sending Broadcast" for the concrete code.
- When it's addressed to a Broadcast Receiver in other applications, specify the address by Intent#setClassName(String, String). Confirm the permitted application by comparing the developer key of the APK signature in the destination package with the white list to send Broadcasts. Actually the following method of using implicit Intents is more practical.

- The Method is to send Broadcasts by specifying in-house-defined Signature Permission to receiverPermission parameter and make the reliable Broadcast Receiver declare to use this Signature Permission. Refer to the sample code section "4.2.1.3 In-house Broadcast Receiver - Receiving/Sending Broadcast" for the concrete code. In addition, implementing this Broadcast-sending method needs to apply the rule "4.2.2.3 Use the In-house Defined Signature Permission after Verifying that it's Defined by an In-house Application (Required)."

4.2.2.6. Sensitive Information Must Not Be Included in the Sticky Broadcast (Required)

Usually, the Broadcasts will be disappeared when they are processed to be received by the available Broadcast Receivers. On the other hand, Sticky Broadcasts (hereafter, Sticky Broadcasts including Sticky Ordered Broadcasts), will not be disappeared from the system even when they processed to be received by the available Broadcast Receivers and will be able to be received by registerReceiver(). When Sticky Broadcast becomes unnecessary, it can be deleted anytime arbitrarily with removeStickyBroadcast().

As it's presupposed that Sticky Broadcast is used by the implicit Intent. Broadcasts with specified receiverPermission Parameter cannot be sent. So information sent by Sticky Broadcast may be taken by unspecified large number of applications including malware. As a result, sensitive information should not be sent by Sticky Broadcast.

4.2.2.7. Pay Attention that the Ordered Broadcast without Specifying the receiverPermission May Not Be Delivered (Required)

Ordered Broadcast without specified receiverPermission Parameter can be received by unspecified large number of applications including malware. Ordered Broadcast is used to receive the returned information from Receiver, and to make several Receivers execute processing one by one. Broadcasts are sent to the Receivers in order of priority. So if the high- priority malware receives Broadcast first and executes abortBroadcast(), Broadcasts won't be delivered to the following Receivers.

4.2.2.8. Handle the Returned Result Data from the Broadcast Receiver Carefully and Securely (Required)

Basically the result data should be processed safely considering the possibility that received results may be the attacking data though the risks vary depending on the types of the Broadcast Receiver

which has returned the result data.
When sender (source) Broadcast Receiver is public Broadcast Receiver, it receives the returned data from unspecified large number of applications. So it may also receive malware's attacking data. When sender (source) Broadcast Receiver is private Broadcast Receiver, it seems no risk. However the data received by other applications may be forwarded as result data indirectly. So the result data should not be considered as safe without any qualification. When sender (source) Broadcast Receiver is In-house Broadcast Receiver, it has some degree of the risks. So it should be processed in a safe way considering the possibility that the result data may be an attacking data.

Please refer to "3.2 Handling Input Data Carefully and Securely"

4.2.2.9. When Providing an Asset Secondarily, the Asset should be protected with the Same Protection Level (Required)

When information or function assets protected by Permission are provided to other applications secondarily, it's necessary to keep the protection standard by claiming the same Permission of the destination application. In the Android Permission security models, privileges are managed only for the direct access to the protected assets from applications. Because of the characteristics, acquired assets may be provided to other applications without claiming Permission which is necessary for protection. This is actually same as re-delegating Permission, as it is called, Permission re-delegation problem. Please refer to "5.2.3.4 Permission Re-delegation Problem."

4.2.3. Advanced Topics

4.2.3.1. Combinations of the exported Attribute and the Intent-filter setting (For Receiver)

Table 4.2 2 represents the permitted combination of export settings and Intent-filter elements when implementing Receivers. The reason why the usage of exported="false" with Intent-filter definition is principally prohibited, is described below.

Table 4.2-3 Usable or not; Combination of exported attribute and intent-filter elements

	Value of exported attribute		
	true	false	Not specified
Intent-filter defined	OK	(Do not Use)	OK
Intent Filter Not Defined	OK	OK	Prohibited

Public Receivers in other applications may be called unexpectedly even though Broadcasts are sent to the private Receivers within the same applications. This is the reason why specifying exported="false" with Intent-filter definition is prohibited. The following 2 figures show how the unexpected calls occur.

Figure 4.2-4 is an example of the normal behaviors which a private Receiver (application A) can be called by implicit Intent only within the same application. Intent-filter (in the figure, action="X") is defined only in application A, so this is the expected behavior.

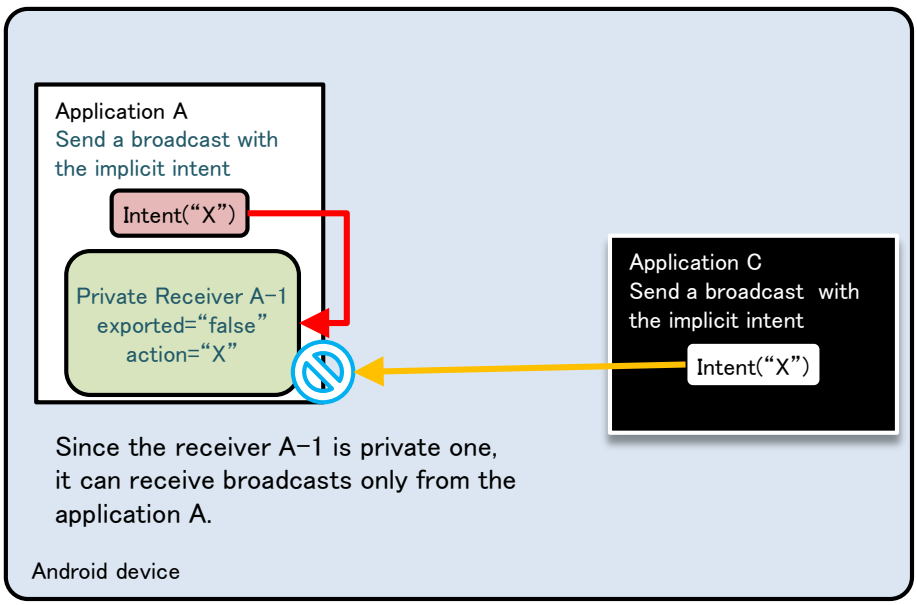


Figure 4.2-4

Figure 4.2-5 is an example that Intent-filter (see action="X" in the figure) is defined in the application B as well as in the application A. First of all, when another application (application C) sends Broadcasts by implicit Intent, they are not received by a private Receiver (A-1) side. So there

won't be any security problem. (See the orange arrow marks in the Figure.)

From security point of view, the problem is application A's call to the private Receiver within the same application. When the application A broadcasts implicit Intent, not only private Receiver within the same application, but also public Receiver (B-1) with the same Intent-filter definition can also receive the Intent. (Red arrow marks in the Figure). In this case, sensitive information may be sent from the application A to B. When the application B is malware, it will cause the leakage of sensitive information. When the Broadcast is Ordered Broadcast, it may receive the unexpected result information.

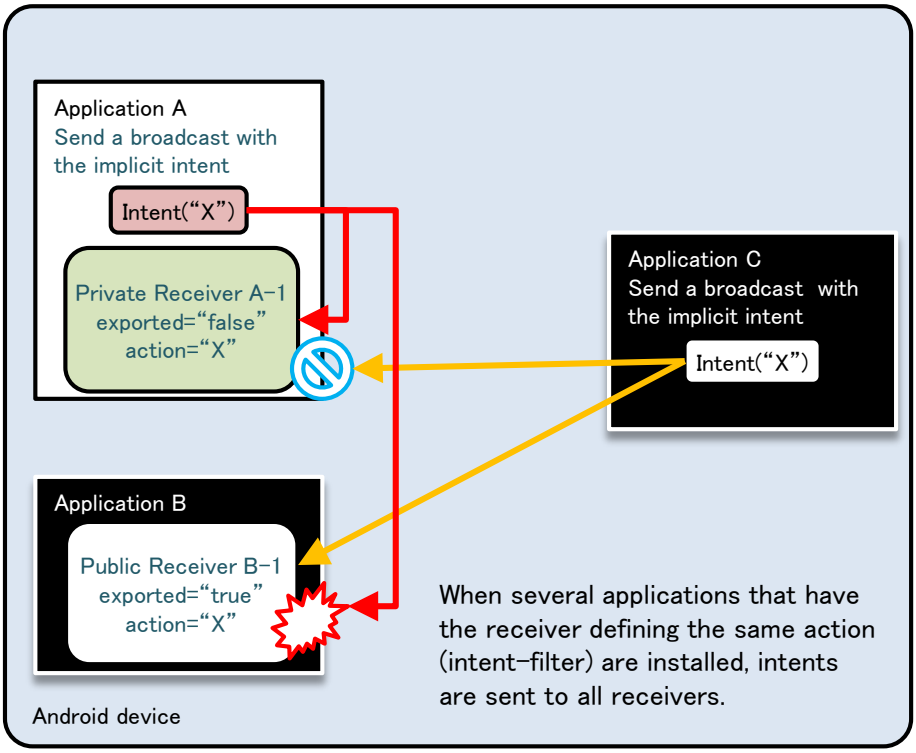


Figure 4.2-5

However, exported="false" with Intent-filter definition should be used when Broadcast Receiver to receive only Broadcast Intent sent by the system is implemented. Other combination should not be used. This is based on the fact that Broadcast Intent sent by the system can be received by exported="false". If other applications send Intent which has same ACTION with Broadcast Intent sent by system, it may cause an unexpected behavior by receiving it. However, this can be prevented by specifying exported="false".

4.2.3.2. Receiver Won't Be Registered before Launching the Application in Android 3.1 or later

In Android 3.1 or later, it's necessary to pay attention that Broadcast Receiver which is statically defined in AndroidManifest.xml won't be enable by just installing. By launching an application once, then it will be able to receive Broadcasts. After installing, processes cannot be launched by receiving Broadcasts as a trigger. By setting Intent to Intent.FLAG_INCLUDE_STOPPED_PACKAGES in Broadcast sender side, the application can receive the Broadcasts even though the application has never been

launched.

4.2.3.3. Private Broadcast Receiver Can Receive the Broadcast that Was Sent by the Same UID Application

Same UID can be provided to several applications. Even if it's private Broadcast Receiver, the Broadcasts sent from the same UID application can be received.

However, it won't be a security problem. Since it's guaranteed that applications with the same UID have the consistent developer keys for signing APK. It means that what private Broadcast Receiver receives is only the Broadcast sent from In-house applications.

4.2.3.4. Types and Features of Broadcasts

Regarding Broadcasts, there are 4 types based on the combination of whether it's Ordered or not, and Sticky or not. Based on Broadcast sending methods, a type of Broadcast to send is determined.

Table 4.2-4

Type of Broadcast	Method for sending	Ordered?	Sticky?
Normal Broadcast	sendBroadcast()	No	No
Ordered Broadcast	sendOrderedBroadcast()	Yes	No
Sticky Broadcast	sendStickyBroadcast()	No	Yes
Sticky Ordered Broadcast	sendStickyOrderedBroadcast()	Yes	Yes

The feature of each Broad cast is described.

Table 4.2-5

Type of Broadcast	Features for each type of Broadcast
Normal Broadcast	Normal Broadcast disappears when it is sent to receivable Broadcast Receiver. Broadcasts are received by several Broadcast Receivers simultaneously. This is a difference from Ordered Broadcast. Broadcasts are allowed to be received by the particular Broadcast Receivers.
Ordered Broadcast	Ordered Broadcast is characterized by receiving Broadcasts one by one in order with receivable Broadcast Receivers. The higher-priority Broadcast Receiver receives earlier. Broadcasts will disappear when Broadcasts are delivered to all Broadcast Receivers or a Broadcast Receiver in the process calls abortBroadcast(). Broadcasts are allowed to be received by the Broadcast Receivers which declare the specified Permission. In addition, the result information sent from Broadcast Receiver can be received by the sender with Ordered Broadcasts. The Broadcast of SMS-receiving notice (SMS_RECEIVED) is a representative example of Ordered Broadcast.

Sticky Broadcast	Sticky Broadcast does not disappear and remains in the system, and then the application that calls registerReceiver() can receive Sticky Broadcast later. Since Sticky Broadcast is different from other Broadcasts, it will never disappear automatically. So when Sticky Broadcast is not necessary, calling removeStickyBroadcast() explicitly is required to delete Sticky Broadcast. Also, Broadcasts cannot be received by the limited Broadcast Receivers with particular Permission. The Broadcast of changing battery-state notice (ACTION_BATTERY_CHANGED) is the representative example of Sticky Broadcast.
Sticky Ordered Broadcast	This is the Broadcast which has both characteristics of Ordered Broadcast and Sticky Broadcast. Same as Sticky Broadcast, it cannot allow only Broadcast Receivers with the particular Permission to receive the Broadcast.

From the Broadcast characteristic behavior point of view, above table is conversely arranged in the following one.

Table 4.2-6

Characteristic behavior of Broadcast	Normal Broadcast	Ordered Broadcast	Sticky Broadcast	Sticky Ordered Broadcast
Limit Broadcast Receivers which can receive Broadcast, by Permission	OK	OK	-	-
Get the results of process from Broadcast Receiver	-	OK	-	OK
Make Broadcast Receivers process Broadcasts in order	-	OK	-	OK
Receive Broadcasts later, which have been already sent	-	-	OK	OK

4.2.3.5. Broadcasted Information May be Output to the LogCat

Basically sending/receiving Broadcasts is not output to LogCat. However, the error log will be output when lacking Permission causes errors in receiver/sender side. Intent information sent by Broadcast is included in the error log, so after an error occurs it's necessary to pay attention that Intent information is displayed in LogCat when Broadcast is sent.

Erorr of lacking Permission in sender side

```
W/ActivityManager(266): Permission Denial: broadcasting Intent { act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION } from org.jssec.android.broadcast.sending (pid=4685, uid=10058) requires org.jssec.android.permission.MY_PERMISSION due to receiver org.jssec.android.broadcastreceiver.creating/org.jssec.android.broadcastreceiver.creating.CreatingType3Receiver
```


Erorr of lacking Permission in receiver side

```
W/ActivityManager(275): Permission Denial: receiving Intent { act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION } to org.jssec.android.broadcastreceiver.creating requires org.jssec.android.permission.MY_PERMISSION due to sender org.jssec.android.broadcast.sending (uid 10158)
```

Since the interface of ContentResolver and SQLiteDatabase are so much alike, it's often misunderstood that Content Provider is so closely related to SQLiteDatabase. However, actually Content Provider simply provides the interface of inter-application data sharing, so it's necessary to pay attention that it does not interfere each data saving format. To save data in Content Provider, SQLiteDatabase can be used, and other saving formats, such as an XML file format, also can be used. Any data saving process is not included in the following sample code, so please add it if needed.

4.3.1. Sample Code

The risks and countermeasures of using Content Provider differ depending on how that Content Provider is being used. In this section, we have classified 5 types of Content Provider based on how the Content Provider is being used. You can find out which type of Content Provider you are supposed to create through the following chart shown below.

Table 4.3-1 Definition of content provider types

Type	Definition
Private Content Provider	A content provider that cannot be used by another application, and therefore is the safest content provider
Public Content Provider	A content provider that is supposed to be used by an unspecified large number of applications
Partner Content Provider	A content provider that can be used by specific applications made by a trusted partner company.
In-house Content Provider	A content provider that can only be used by other in-house applications
Temporary permit Content Provider	A content provider that is basically private content provider, but permits specific applications to access the particular URI.

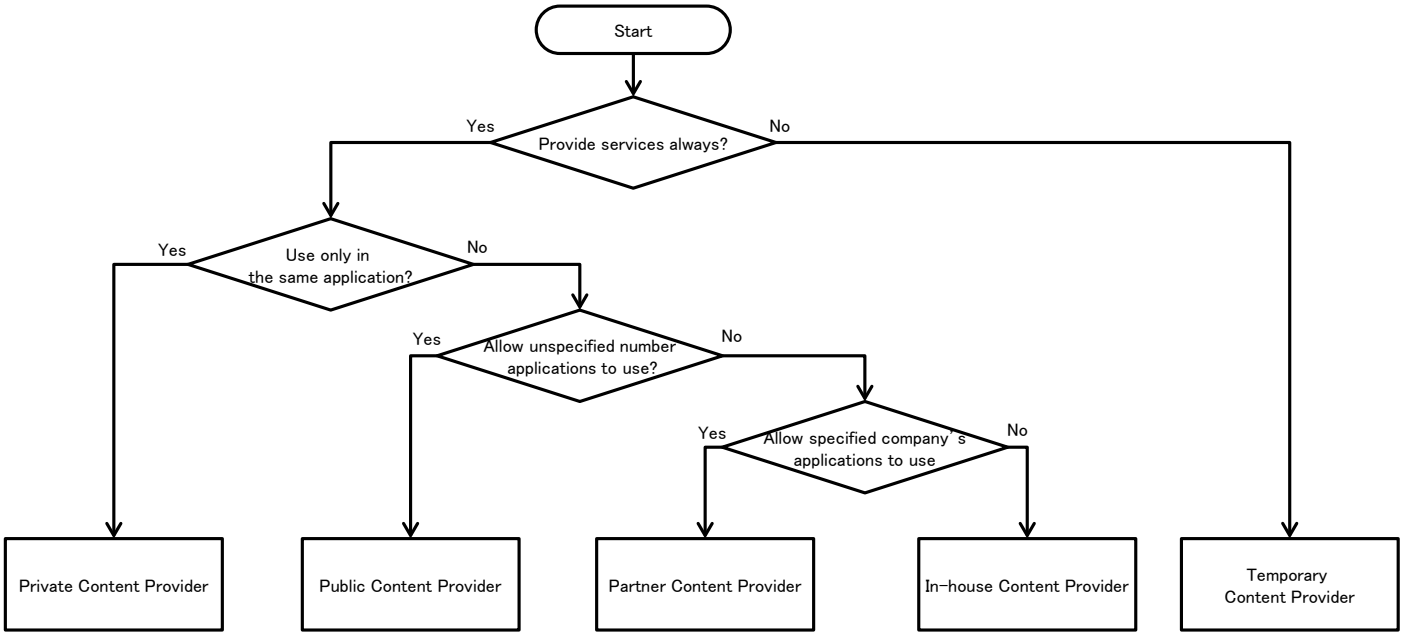


Figure 4.3-1

4.3.1.1. Creating/Using Private Content Providers

Private Content Provider is the Content Provider which is used only in the single application, and the safest Content Provider. However, it's necessary to pay attention that private setting for Content Provider does not work in Android 2.2 (API Level 8) or earlier.

Sample code of how to implement a private Content Provider is shown below.

Points (Creating a Content Provider):

1. Do not (Cannot) implement Private Content Provider in Android 2.2 (API Level 8) or earlier.
2. Explicitly set the exported attribute to false.
3. Handle the received request data carefully and securely, even though the data comes from the same application.
4. Sensitive information can be sent since it is sending and receiving all within the same application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.privateprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <!-- *** POINT 1 *** Do not (Cannot) implement Private Content Provider in Android 2.2 (API Level 8) or earlier.
-->
    <uses-sdk android:minSdkVersion="9" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- *** POINT 2 *** Explicitly set the exported attribute to false. -->
        <provider
            android:name=".PrivateProvider"
            android:authorities="org.jssec.android.provider.privateprovider"
            android:exported="false" />
    </application>
</manifest>
```

PrivateProvider.java

```
package org.jssec.android.provider.privateprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
```

```
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PrivateProvider extends ContentProvider {

    public static final String AUTHORITY = "org.jssec.android.provider.privateprovider";
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using database.
    private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "Longon" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {
        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."
```



```
        // *** POINT 4 *** Sensitive information can be sent since it is sending and receiving all within the same app
lication.
        // However, the result of getType rarely has the sensitive meaning.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Sensitive information can be sent since it is sending and receiving all within the same app
lication.
        // It depends on application whether the query result has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Sensitive information can be sent since it is sending and receiving all within the same app
lication.
        // It depends on application whether the issued ID has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);
```

```
        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Sensitive information can be sent since it is sending and receiving all within the same app
lication.
        // It depends on application whether the number of updated records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Sensitive information can be sent since it is sending and receiving all within the same app
lication.
        // It depends on application whether the number of deleted records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
```



```
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

Next is an example of Activity which uses Private Content Provider.

Points (Using a Content Provider):

5. Sensitive information can be sent since the destination provider is in the same application.
6. Handle received result data carefully and securely, even though the data comes from the same application.

PrivateUserActivity.java

```
package org.jssec.android.provider.privateprovider;

import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateUserActivity extends Activity {

    public void onQueryClick(View view) {

        logLine("[Query]");

        Cursor cursor = null;
        try {
            // *** POINT 5 *** Sensitive information can be sent since the destination provider is in the same applica
tion.
            cursor = getContentResolver().query(
                    PrivateProvider.Download.CONTENT_URI, null, null, null, null);

            // *** POINT 6 *** Handle received result data carefully and securely,
            // even though the data comes from the same application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // *** POINT 5 *** Sensitive information can be sent since the destination provider is in the same application
.
        Uri uri = getContentResolver().insert(PrivateProvider.Download.CONTENT_URI, null);

        // *** POINT 6 *** Handle received result data carefully and securely,
```



```
        // even though the data comes from the same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // *** POINT 5 *** Sensitive information can be sent since the destination provider is in the same application
.
        int count = getContentResolver().update(PrivateProvider.Download.CONTENT_URI, null, null, null);

        // *** POINT 6 *** Handle received result data carefully and securely,
        // even though the data comes from the same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // *** POINT 5 *** Sensitive information can be sent since the destination provider is in the same application
.
        int count = getContentResolver().delete(
                PrivateProvider.Download.CONTENT_URI, null, null);

        // *** POINT 6 *** Handle received result data carefully and securely,
        // even though the data comes from the same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

4.3.1.2. Creating/Using Public Content Providers

Public Content Provider is the Content Provider which is supposed to be used by unspecified large number of applications. It's necessary to pay attention that since this doesn't specify clients, it may be attacked and tampered by Malware. For example, a saved data may be taken by select(), a data may be changed by update(), or a fake data may be inserted/deleted by insert()/delete().

In addition, when using a custom Public Content Provider which is not provided by Android OS, it's necessary to pay attention that request parameter may be received by Malware which masquerades as the custom Public Content Provider, and also the attack result data may be sent. Contacts and MediaStore provided by Android OS are also Public Content Providers, but Malware cannot masquerades as them.

Sample code to implement a Public Content Provider is shown below.

Points (Creating a Content Provider):

1. Handle the received request data carefully and securely.
2. When returning a result, do not include sensitive information.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.publicprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <provider
            android:name=".PublicProvider"
            android:authorities="org.jssec.android.provider.publicprovider"
            android:exported="true" />
    </application>
</manifest>
```

PublicProvider.java

```
package org.jssec.android.provider.publicprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PublicProvider extends ContentProvider {
```



```
public static final String AUTHORITY = "org.jssec.android.provider.publicprovider";
public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

// Expose the interface that the Content Provider provides.
public interface Download {
    public static final String PATH = "downloads";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
}
public interface Address {
    public static final String PATH = "addresses";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
}

// UriMatcher
private static final int DOWNLOADS_CODE = 1;
private static final int DOWNLOADS_ID_CODE = 2;
private static final int ADDRESSES_CODE = 3;
private static final int ADDRESSES_ID_CODE = 4;
private static UriMatcher sUriMatcher;
static {
    sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
    sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
    sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
    sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
}

// Since this is a sample program,
// query method returns the following fixed result always without using database.
private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "city" });
static {
    sAddressCursor.addRow(new String[] { "1", "New York" });
    sAddressCursor.addRow(new String[] { "2", "London" });
    sAddressCursor.addRow(new String[] { "3", "Paris" });
}
private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
static {
    sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
    sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
}

@Override
public boolean onCreate() {
    return true;
}

@Override
public String getType(Uri uri) {

    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
    case ADDRESSES_CODE:
        return CONTENT_TYPE;

    case DOWNLOADS_ID_CODE:
    case ADDRESSES_ID_CODE:
        return CONTENT_ITEM_TYPE;
```

```
        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // *** POINT 1 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 2 *** When returning a result, do not include sensitive information.
        // It depends on application whether the query result has sensitive meaning or not.
        // If no problem when the information is taken by malware, it can be returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 1 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 2 *** When returning a result, do not include sensitive information.
        // It depends on application whether the issued ID has sensitive meaning or not.
        // If no problem when the information is taken by malware, it can be returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // *** POINT 1 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
```



```
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 2 *** When returning a result, do not include sensitive information.
        // It depends on application whether the number of updated records has sensitive meaning or not.
        // If no problem when the information is taken by malware, it can be returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 1 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 2 *** When returning a result, do not include sensitive information.
        // It depends on application whether the number of deleted records has sensitive meaning or not.
        // If no problem when the information is taken by malware, it can be returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

Next is an example of Activity which uses Public Content Provider.

Points (Using a Content Provider):

3. Do not send sensitive information.
4. When receiving a result, handle the result data carefully and securely.

PublicUserActivity.java

```
package org.jssec.android.provider.publicuser;

import android.app.Activity;
import android.content.ContentValues;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicUserActivity extends Activity {

    // Target Content Provider Information
    private static final String AUTHORITY = "org.jssec.android.provider.publicprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        Cursor cursor = null;
        try {
            // *** POINT 3 *** Do not send sensitive information.
            // since the target Content Provider may be malware.
            // If no problem when the information is taken by malware, it can be included in the request.
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // *** POINT 4 *** When receiving a result, handle the result data carefully and securely.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
```



```
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        // *** POINT 3 *** Do not send sensitive information.
        // since the target Content Provider may be malware.
        // If no problem when the information is taken by malware, it can be included in the request.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // *** POINT 4 *** When receiving a result, handle the result data carefully and securely.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        // *** POINT 3 *** Do not send sensitive information.
        // since the target Content Provider may be malware.
        // If no problem when the information is taken by malware, it can be included in the request.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // *** POINT 4 *** When receiving a result, handle the result data carefully and securely.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        // *** POINT 3 *** Do not send sensitive information.
        // since the target Content Provider may be malware.
        // If no problem when the information is taken by malware, it can be included in the request.
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);
```

```
        // *** POINT 4 *** When receiving a result, handle the result data carefully and securely.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi = getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


4.3.1.3. Creating/Using Partner Content Providers

Partner Content Provider is the Content Provider which can be used only by the particular applications. The system consists of a partner company's application and In-house application, and it is used to protect the information and features which are handled between a partner application and an In-house application.

Sample code to implement a partner-only Content Provider is shown below.

Points (Creating a Content Provider):

1. Verify if the certificate of a requesting application has been registered in the own white list.
2. Handle the received request data carefully and securely, even though the data comes from a partner application.
3. Information that is granted to disclose to partner applications can be returned.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.partnerprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <provider
            android:name=".PartnerProvider"
            android:authorities="org.jssec.android.provider.partnerprovider"
            android:exported="true" />
    </application>
</manifest>
```

PartnerProvider.java

```
package org.jssec.android.provider.partnerprovider;

import java.util.List;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.ActivityManager;
import android.app.ActivityManager.RunningAppProcessInfo;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;
```

```
import android.os.Binder;

public class PartnerProvider extends ContentProvider {

    public static final String AUTHORITY = "org.jssec.android.provider.partnerprovider";
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using database.
    private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
    }

    // *** POINT 1 *** Verify if the certificate of a requesting application has been registered in the own white lis
t.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application org.jssec.android.provider.partneruser.
        sWhitelists.add("org.jssec.android.provider.partneruser", isdebug ?
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of "partner key" in the keystore.
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");
```



```
        // Register following other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
    // Get the package name of the calling application.
    private String getCallingPackage(Context context) {
        String pkgname = null;
        ActivityManager am = (ActivityManager)context.getSystemService(Context.ACTIVITY_SERVICE);
        List<RunningAppProcessInfo> procList = am.getRunningAppProcesses();
        int callingPid = Binder.getCallingPid();
        if (procList != null) {
            for (RunningAppProcessInfo proc : procList) {
                if (proc.pid == callingPid) {
                    pkgname = proc.pkgList[proc.pkgList.length - 1];
                    break;
                }
            }
        }
        return pkgname;
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // *** POINT 1 *** Verify if the certificate of a requesting application has been registered in the own white
list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner application.");
        }

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."
```

```
        // *** POINT 3 *** Information that is granted to disclose to partner applications can be returned.
        // It depends on application whether the query result can be disclosed or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 1 *** Verify if the certificate of a requesting application has been registered in the own white
list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner application.");
        }

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Information that is granted to disclose to partner applications can be returned.
        // It depends on application whether the issued ID has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // *** POINT 1 *** Verify if the certificate of a requesting application has been registered in the own white
list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner application.");
        }

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
```



```
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Information that is granted to disclose to partner applications can be returned.
        // It depends on application whether the number of updated records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 1 *** Verify if the certificate of a requesting application has been registered in the own white
list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner application.");
        }

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Information that is granted to disclose to partner applications can be returned.
        // It depends on application whether the number of deleted records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

Next is an example of Activity which use partner only Content Provider.

Points (Using a Content Provider):

4. Verify if the certificate of the target application has been registered in the own white list.
5. Information that is granted to disclose to partner applications can be sent.
6. Handle the received result data carefully and securely, even though the data comes from a partner application.

PartnerActivity.java

```
package org.jssec.android.provider.partneruser;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PartnerUserActivity extends Activity {

    // Target Content Provider Information
    private static final String AUTHORITY = "org.jssec.android.provider.partnerprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // *** POINT 4 *** Verify if the certificate of the target application has been registered in the own white list.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application org.jssec.android.provider.partnerprovider.
        sWhitelists.add("org.jssec.android.provider.partnerprovider", isdebug ?
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of "partner key" in the keystore.
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // Register following other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // Get package name of target content provider.
    private String providerPkgname(Uri uri) {
        String pkgname = null;
        ProviderInfo pi = getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
```



```
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // *** POINT 4 *** Verify if the certificate of the target application has been registered in the own white li
st.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner applications.");
            return;
        }

        Cursor cursor = null;
        try {
            // *** POINT 5 *** Information that is granted to disclose to partner applications can be sent.
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // *** POINT 6 *** Handle the received result data carefully and securely,
            // even though the data comes from a partner application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // *** POINT 4 *** Verify if the certificate of the target application has been registered in the own white li
st.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner applications.");
            return;
        }

        // *** POINT 5 *** Information that is granted to disclose to partner applications can be sent.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // *** POINT 6 *** Handle the received result data carefully and securely,
        // even though the data comes from a partner application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }
```

```
    public void onUpdateClick(View view) {

        logLine("[Update]");

        // *** POINT 4 *** Verify if the certificate of the target application has been registered in the own white li
st.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner applications.");
            return;
        }

        // *** POINT 5 *** Information that is granted to disclose to partner applications can be sent.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // *** POINT 6 *** Handle the received result data carefully and securely,
        // even though the data comes from a partner application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // *** POINT 4 *** Verify if the certificate of the target application has been registered in the own white li
st.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner applications.");
            return;
        }

        // *** POINT 5 *** Information that is granted to disclose to partner applications can be sent.
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // *** POINT 6 *** Handle the received result data carefully and securely,
        // even though the data comes from a partner application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
```

```
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


4.3.1.4. Creating/Using In-house Content Providers

In-house Content Provider is the Content Provider which prohibits to be used by applications other than In house only applications.

Sample code of how to implement an In house only Content Provider is shown below.

Points (Creating a Content Provider):

1. Define an in-house signature permission.
2. Require the in-house signature permission.
3. Verify if the in-house signature permission is defined by an in-house application.
4. Verify the safety of the parameter even if it's a request from In house only application.
5. Sensitive information can be returned since the requesting application is in-house.
6. When exporting an APK from Eclipse, sign the APK with the same developer key as that of the requesting application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.inhouseprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 1 *** Define an in-house signature permission -->
    <permission
        android:name="org.jssec.android.provider.inhouseprovider.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- *** POINT 2 *** Require the in-house signature permission -->
        <provider
            android:name=".InhouseProvider"
            android:authorities="org.jssec.android.provider.inhouseprovider"
            android:permission="org.jssec.android.provider.inhouseprovider.MY_PERMISSION"
            android:exported="true" />
    </application>
</manifest>
```

InhouseProvider.java

```
package org.jssec.android.provider.inhouseprovider;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
```

```
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class InhouseProvider extends ContentProvider {

    public static final String AUTHORITY = "org.jssec.android.provider.inhouseprovider";
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using database.
    private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
    }

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.provider.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
```



```
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // *** POINT 3 *** Verify if the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not declared by in-house application.")
;
        }

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Sensitive information can be returned since the requesting application is in-house.
        // It depends on application whether the query result has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
```

```
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 3 *** Verify if the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not declared by in-house application.")
;
        }

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Sensitive information can be returned since the requesting application is in-house.
        // It depends on application whether the issued ID has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // *** POINT 3 *** Verify if the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not declared by in-house application.")
;
        }

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Sensitive information can be returned since the requesting application is in-house.
        // It depends on application whether the number of updated records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
```



```
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 3 *** Verify if the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not declared by in-house application.")
;
        }

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Sensitive information can be returned since the requesting application is in-house.
        // It depends on application whether the number of deleted records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }
```

```
    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
```



```
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 6 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the requesting application.

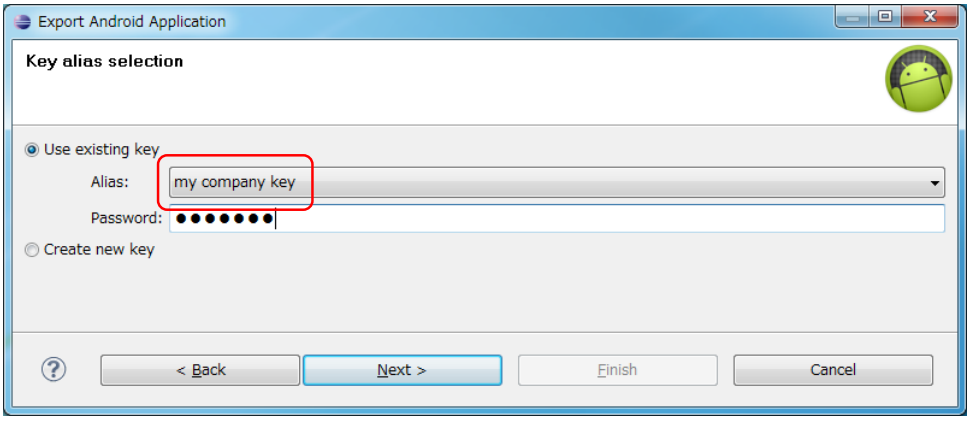


Figure 4.3-2

Next is the example of Activity which uses In house only Content Provider.

Point (Using a Content Provider):

7. Declare to use the in-house signature permission.
8. Verify if the in-house signature permission is defined by an in-house application.
9. Verify if the destination application is signed with the in-house certificate.
10. Sensitive information can be sent since the destination application is in-house one.
11. Handle the received result data carefully and securely, even though the data comes from an in-house application.
12. When exporting an APK from Eclipse, sign the APK with the same developer key as that of the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.inhouseuser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 7 *** Declare to use the in-house signature permission. -->
    <uses-permission
        android:name="org.jssec.android.provider.inhouseprovider.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".InhouseUserActivity"
            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>
</manifest>
```

InhouseUserActivity.java

```
package org.jssec.android.provider.inhouseuser;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
```



```
import android.view.View;
import android.widget.TextView;

public class InhouseUserActivity extends Activity {

    // Target Content Provider Information
    private static final String AUTHORITY = "org.jssec.android.provider.inhouseprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.provider.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Get package name of target content provider.
    private static String providerPkgname(Context context, Uri uri) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
        ProviderInfo pi = pm.resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // *** POINT 8 *** Verify if the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  The in-house signature permission is not declared by in-house application.");
            return;
        }

        // *** POINT 9 *** Verify if the destination application is signed with the in-house certificate.
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  The target content provider is not served by in-house applications.");
            return;
        }

        Cursor cursor = null;
        try {
            // *** POINT 10 *** Sensitive information can be sent since the destination application is in-house one.
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);
```

```
        // *** POINT 11 *** Handle the received result data carefully and securely,
        // even though the data comes from an in-house application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        if (cursor == null) {
            logLine("  null cursor");
        } else {
            boolean moved = cursor.moveToFirst();
            while (moved) {
                logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                moved = cursor.moveToNext();
            }
        }
    }
    finally {
        if (cursor != null) cursor.close();
    }
}

public void onInsertClick(View view) {

    logLine("[Insert]");

    // *** POINT 8 *** Verify if the in-house signature permission is defined by an in-house application.
    String correctHash = myCertHash(this);
    if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
        logLine("  The in-house signature permission is not declared by in-house application.");
        return;
    }

    // *** POINT 9 *** Verify if the destination application is signed with the in-house certificate.
    String pkgname = providerPkgname(this, Address.CONTENT_URI);
    if (!PkgCert.test(this, pkgname, correctHash)) {
        logLine("  The target content provider is not served by in-house applications.");
        return;
    }

    // *** POINT 10 *** Sensitive information can be sent since the destination application is in-house one.
    ContentValues values = new ContentValues();
    values.put("city", "Tokyo");
    Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

    // *** POINT 11 *** Handle the received result data carefully and securely,
    // even though the data comes from an in-house application.
    // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
    logLine("  uri:" + uri);
}

public void onUpdateClick(View view) {

    logLine("[Update]");

    // *** POINT 8 *** Verify if the in-house signature permission is defined by an in-house application.
    String correctHash = myCertHash(this);
    if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
        logLine("  The in-house signature permission is not declared by in-house application.");
        return;
    }

    // *** POINT 9 *** Verify if the destination application is signed with the in-house certificate.
```



```
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  The target content provider is not served by in-house applications.");
            return;
        }

        // *** POINT 10 *** Sensitive information can be sent since the destination application is in-house one.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // *** POINT 11 *** Handle the received result data carefully and securely,
        // even though the data comes from an in-house application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // *** POINT 8 *** Verify if the in-house signature permission is defined by an in-house application.
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  The target content provider is not served by in-house applications.");
            return;
        }

        // *** POINT 9 *** Verify if the destination application is signed with the in-house certificate.
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  The target content provider is not served by in-house applications.");
            return;
        }

        // *** POINT 10 *** Sensitive information can be sent since the destination application is in-house one.
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // *** POINT 11 *** Handle the received result data carefully and securely,
        // even though the data comes from an in-house application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }
```



```
    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 12 *** When exporting an APK from Eclipse, sign the APK with the same developer key as that of the destination application.

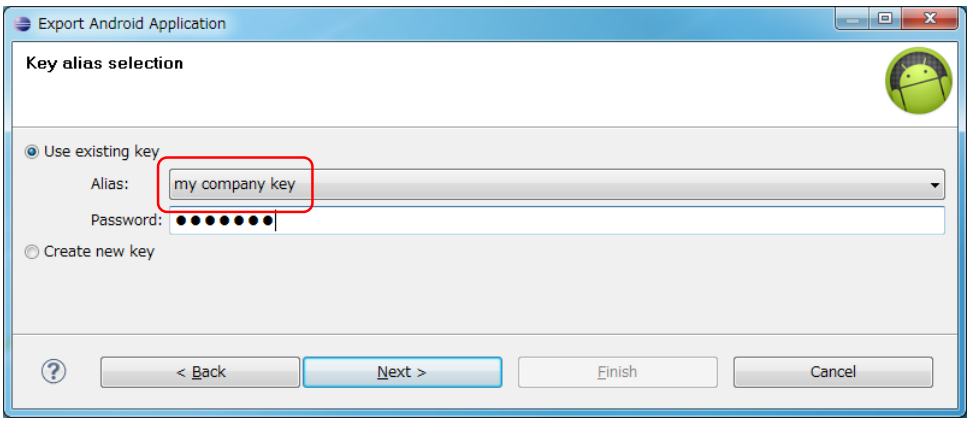


Figure 4.3-3

4.3.1.5. Creating/Using Temporary permit Content Providers

Temporary permit Content Provider is basically a private Content Provider, but this permits the particular applications to access the particular URI. By sending an Intent which special flag is specified to the target applications, temporary access permission is provided to those applications. Contents provider side application can give the access permission actively to other applications, and it can also give access permission passively to the application which claims the temporary access permission.

Sample code of how to implement a temporary permit Content Provider is shown below.

Points (Creating a Content Provider):

1. Do not (Cannot) implement temporary permit content provider in Android 2.2 (API Level 8) or earlier.
2. Explicitly set the exported attribute to false.
3. Specify the path to grant access temporarily with the grant-uri-permission.
4. Handle the received request data carefully and securely, even though the data comes from the application granted access temporarily.
5. Information that is granted to disclose to the temporary access applications can be returned.
6. Specify URI for the intent to grant temporary access.
7. Specify access rights for the intent to grant temporary access.
8. Send the explicit intent to an application to grant temporary access.
9. Return the intent to the application that requests temporary access.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.temporaryprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <!-- *** POINT 1 *** Do not (Cannot) implement temporary permit content provider in Android 2.2 (API Level 8) or
earlier. -->
    <uses-sdk android:minSdkVersion="9" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:name=".TemporaryActiveGrantActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Temporary Content Provider -->
        <!-- *** POINT 2 *** Explicitly set the exported attribute to false. -->
        <provider
```



```
            android:name=".TemporaryProvider"
            android:authorities="org.jssec.android.provider.temporaryprovider"
            android:exported="false" >

            <!-- *** POINT 3 *** Specify the path to grant access temporarily with the grant-uri-permission. -->
            <grant-uri-permission android:path="/addresses" />

        </provider>

        <activity
            android:name=".TemporaryPassiveGrantActivity"
            android:label="@string/app_name"
            android:exported="true" />
    </application>
</manifest>
```

TemporaryProvider.java

```
package org.jssec.android.provider.temporaryprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class TemporaryProvider extends ContentProvider {
    public static final String AUTHORITIY = "org.jssec.android.provider.temporaryprovider";
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using database.
```

```
    private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access temporarily.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Information that is granted to disclose to the temporary access applications can be returne
d.
        // It depends on application whether the query result can be disclosed or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
```



```
    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access temporarily.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Information that is granted to disclose to the temporary access applications can be returne
d.
        // It depends on application whether the issued ID has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // *** POINT 4 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access temporarily.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Information that is granted to disclose to the temporary access applications can be returne
d.
        // It depends on application whether the number of updated records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 4 *** Handle the received request data carefully and securely,
```

```
        // even though the data comes from the application granted access temporarily.
        // Here, whether uri is within expectations or not, is verified by UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 5 *** Information that is granted to disclose to the temporary access applications can be returne
d.
        // It depends on application whether the number of deleted records has sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

TemporaryActiveGrantActivity.java

```
package org.jssec.android.provider.temporaryprovider;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class TemporaryActiveGrantActivity extends Activity {

    // User Activity Information
    private static final String TARGET_PACKAGE =  "org.jssec.android.provider.temporaryuser";
    private static final String TARGET_ACTIVITY = "org.jssec.android.provider.temporaryuser.TemporaryUserActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.active_grant);
    }

    // In the case that Content Provider application grants access permission to other application actively.
    public void onSendClick(View view) {
        try {
            Intent intent = new Intent();

            // *** POINT 6 *** Specify URI for the intent to grant temporary access.
            intent.setData(TemporaryProvider.Address.CONTENT_URI);

            // *** POINT 7 *** Specify access rights for the intent to grant temporary access.
```



```
            intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

            // *** POINT 8 *** Send the explicit intent to an application to grant temporary access.
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);

        } catch (ActivityNotFoundException e) {
            Toast.makeText(this, "User Activity not found.", Toast.LENGTH_LONG).show();
        }
    }
}
```

TemporaryPassiveGrantActivity.java

```
package org.jssec.android.provider.temporaryprovider;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class TemporaryPassiveGrantActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.passive_grant);
    }

    // In the case that Content Provider application passively grants access permission
    // to the application that requested Content Provider access.
    public void onGrantClick(View view) {
        Intent intent = new Intent();

        // *** POINT 6 *** Specify URI for the intent to grant temporary access.
        intent.setData(TemporaryProvider.Address.CONTENT_URI);

        // *** POINT 7 *** Specify access rights for the intent to grant temporary access.
        intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        // *** POINT 9 *** Return the intent to the application that requests temporary access.
        setResult(Activity.RESULT_OK, intent);
        finish();
    }

    public void onCloseClick(View view) {
        finish();
    }
}
```

Next is the example of temporary permit Content Provider.

Points (Using a Content Provider):

10. Do not send sensitive information.
11. When receiving a result, handle the result data carefully and securely.

TemporaryUserActivity.java

```
package org.jssec.android.provider.temporaryuser;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class TemporaryUserActivity extends Activity {

    // Information of the Content Provider's Activity to request temporary content provider access.
    private static final String TARGET_PACKAGE =  "org.jssec.android.provider.temporaryprovider";
    private static final String TARGET_ACTIVITY = "org.jssec.android.provider.temporaryprovider.TemporaryPassiveGran
tActivity";

    // Target Content Provider Information
    private static final String AUTHORITY = "org.jssec.android.provider.temporaryprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    private static final int REQUEST_CODE = 1;

    public void onQueryClick(View view) {

        logLine("[Query]");

        Cursor cursor = null;
        try {
            if (!providerExists(Address.CONTENT_URI)) {
                logLine("  Content Provider doesn't exist.");
                return;
            }

            // *** POINT 10 *** Do not send sensitive information.
            // If no problem when the information is taken by malware, it can be included in the request.
            cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // *** POINT 11 *** When receiving a result, handle the result data carefully and securely.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
```



```
                    moved = cursor.moveToNext();
                }
            }
        } catch (SecurityException ex) {
            logLine("  Exception:" + ex.getMessage());
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    // In the case that this application requests temporary access to the Content Provider
    // and the Content Provider passively grants temporary access permission to this application.
    public void onGrantRequestClick(View view) {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
        try {
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            logLine("Content Provider's Activity not found.");
        }
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi = getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    // In the case that the Content Provider application grants temporary access
    // to this application actively.
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

4.3.2. Rule Book

Be sure to follow the rules below when Implementing or using a content provider.

1. Content Provider that Is Used Only in an Application Can Not Be Created in Android 2.2 (API Level 8) or Earlier (Required)
2. Content Provider that Is Used Only in an Application Must Be Set as Private (Required)
3. Handle the Received Request Parameter Carefully and Securely (Required)
4. Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-house Application (Required)
5. When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)
6. When Providing an Asset Secondarily, the Asset should be Protected with the Same Level of Protection (Required)

And user side should follow the below rules, too.

7. Handle the Returned Result Data from the Content Provider Carefully and Securely (Required)

4.3.2.1. Content Provider that Is Used Only in an Application Can Not Be Created in Android 2.2 (API Level 8) or Earlier (Required)

Private setting for a Content Provider does not work in Android 2.2 (API Level 8) or earlier. To share a data in the same application, access a data storage location such as a data base instead of using a Content Provider.

4.3.2.2. Content Provider that Is Used Only in an Application Must Be Set as Private (Required)

Content Provider which is used only in a single application is not necessary to be accessed by other applications, and the access which attacks the Content Provider is not often considered by developers. A Content Provider is basically the system to share data, so it's handled as public by default. A Content Provider which is used only in a single application should be set as private explicitly, and it should be a private Content Provider. In Android 2.3.1 (API Level 9) or later, a Content Provider can be set as private by specifying android:exported="false" in provider element.

AndroidManifest.xml

```
    <!-- *** POINT 1 *** Do not (Cannot) implement Private Content Provider in Android 2.2 (API Level 8) or earlier.
-->
    <uses-sdk android:minSdkVersion="9" />
-abbreviation-

        <!-- *** POINT 2 *** Set false for the exported attribute explicitly. -->
        <provider
            android:name=".PrivateProvider"
            android:authorities="org.jssec.android.provider.privateprovider"
            android:exported="false" />
```


4.3.2.3. Handle the Received Request Parameter Carefully and Securely (Required)

Risks differ depending on the types of Content Providers, but when processing request parameters, the first thing you should do is input validation.

Although each method of a Content Provider has the interface which is supposed to receive the component parameter of SQL statement, actually it simply hands over the arbitrary character string in the system, so it's necessary to pay attention that Contents Provider side needs to suppose the case that unexpected parameter may be provided.

Since Public Content Providers can receive requests from untrusted sources, they can be attacked by malware. On the other hand, Private Content Providers will never receive any requests from other applications directly, but it is possible that a Public Activity in the targeted application may forward a malicious Intent to a Private Content Provider so you should not assume that Private Content Providers cannot receive any malicious input.
Since other Content Providers also have the risk of a malicious intent being forwarded to them as well, it is necessary to perform input validation on these requests as well.
Please refer to "3.2 Handling Input Data Carefully and Securely"

4.3.2.4. Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-house Application (Required)

Make sure to protect your in-house Content Providers by defining an in-house signature permission when creating the Content Provider. Since defining a permission in the AndroidManifest.xml file or declaring a permission request does not provide adequate security, please be sure to refer to "5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission."

4.3.2.5. When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)

In case of query() or insert(), Cursor or Uri is returned to the request sending application as a result information. When sensitive information is included in the result information, the information may be leaked from the destination application. In case of update() or delete(), number of updated/deleted records is returned to the request sending application as a result information. In rare cases, depending on some application specs, the number of updated/deleted records has the sensitive meaning, so please pay attention to this.

4.3.2.6. When Providing an Asset Secondarily, the Asset should be Protected with the Same Level of Protection (Required)

When an information or function asset, that is protected by a permission, is provided to another application secondhand, you need to make sure that it has the same required permissions needed to

access the asset. In the Android OS permission security model, only an application that has been granted proper permissions can directly access a protected asset. However, there is a loophole because an application with permissions to an asset can act as a proxy and allow access to an unprivileged application. Substantially this is the same as redelegating a permission, so it is referred to as the "Permission Redelegation" problem. Please refer to "5.2.3.4 Permission Re-delegation Problem."

4.3.2.7. Handle the Returned Result Data from the Content Provider Carefully and Securely

(Required)

Risks differ depending on the types of Content Provider, but when processing a result data, the first thing you should do is input validation.

In case that the destination Content Provider is a public Content Provider, Malware which masquerades as the public Content Provider may return the attack result data. On the other hand, in case that the destination Content Provider is a private Content Provider, it is less risk because it receives the result data from the same application, but you should not assume that private Content Providers cannot receive any malicious input.Since other Content Providers also have the risk of a malicious data being returned to them as well, it is necessary to perform input validation on that result data as well.

Please refer to "3.2 Handling Input Data Carefully and Securely"

4.4.1. Sample Code

The risks and countermeasures of using Services differ depending on how that Service is being used. You can find out which type of Service you are supposed to create through the following chart shown below. Since the secure coding best practice varies according to how the service is created, we will also explain about the implementation of the Service as well.

Table 4.4-1 Definition of service types

Type	Definition
Private Service	A service that cannot be used another application, and therefore is the safest service.
Public Service	A service that is supposed to be used by an unspecified large number of applications
Partner Service	A service that can only be used by the specific applications made by a trusted partner company.
In-house Service	A service that can only be used by other in-house applications.

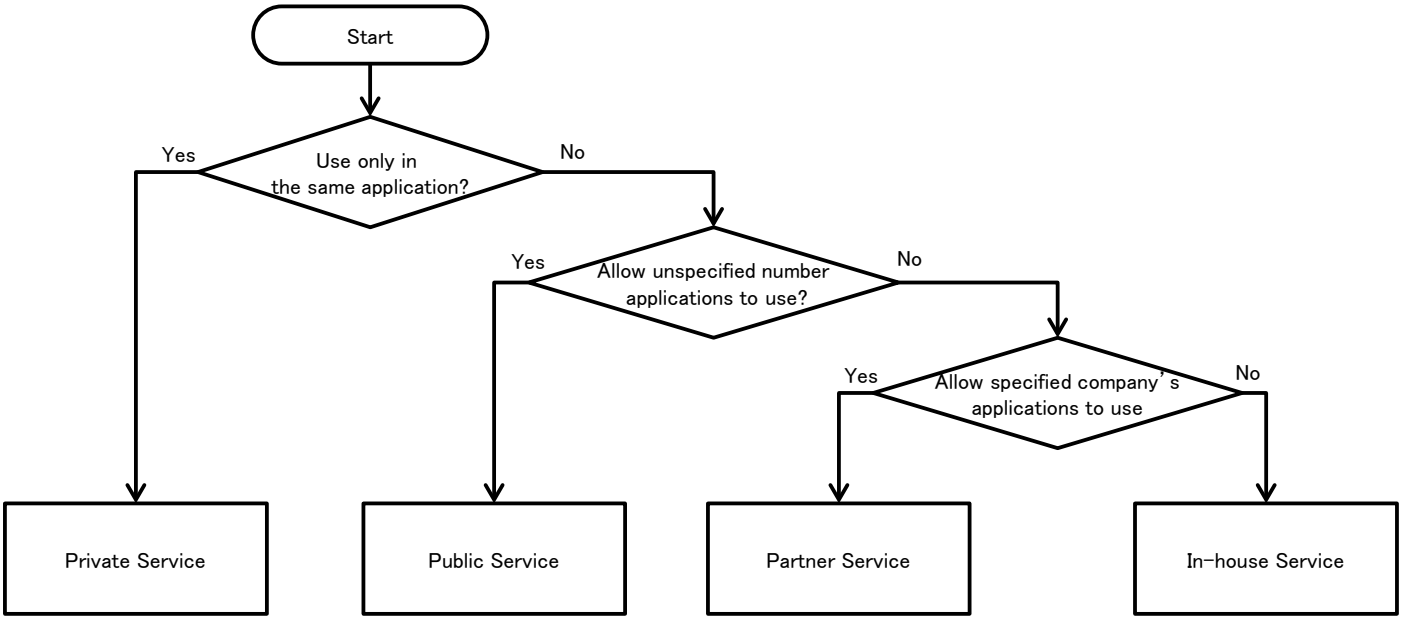


Figure 4.4-1

There are several implementation methods for Service, and you will select the method which matches with the type of Service that you suppose to create. The items of vertical columns in the table show the implementation methods, and these are divided into 5 types. "OK" stands for the possible combination and others show impossible/difficult combinations in the table.

Please refer to "4.4.3.2 How to Implement Service" and Sample code of each Service type (with * mark in a table) for detailed implementation methods of Service.

Table 4.4-2

Category	Private Service	Public Service	Partner Service	In-house Service
startService type	**OK***	OK	-	OK
IntentService type	OK	**OK***	-	OK
local bind type	OK	-	-	-
Messenger bind type	OK	OK	-	**OK***
AIDL bind type	OK	OK	**OK***	OK

Sample code for each security type of Service are shown as below, by using combination of * mark in Table 4.4-2.

4.4.1.1. Creating/Using Private Services

Private Services are Services which cannot be launched by the other applications and therefore it is the safest Service.
When using Private Services that are only used within the application, as long as you use explicit Intents to the class then you do not have to worry about accidently sending it to any other application.

Sample code of how to use the startService type Service is shown below.

Points (Creating a Service):

1. Explicitly set the exported attribute to false.
2. Handle the received intent carefully and securely, even though the intent was sent from the same application.
3. Sensitive information can be sent since the requesting application is in the same application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.privateservice"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Private Service derived from Service class -->
        <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
        <service android:name=".PrivateStartService" android:exported="false"/>

        <!-- Private Service derived from IntentService class -->
        <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
        <service android:name=".PrivateIntentService" android:exported="false"/>

        </application>

</manifest>
```

PrivateStartService.java

```
package org.jssec.android.service.privateservice;
```

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class PrivateStartService extends Service {

    // The onCreate gets called only one time when the service starts.
    @Override
    public void onCreate() {
        Toast.makeText(this, "PrivateStartService - onCreate()", Toast.LENGTH_SHORT).show();
    }

    // The onStartCommand gets called each time after the startService gets called.
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // *** POINT 2 *** Handle the received intent carefully and securely,
        // even though the intent was sent from the same application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                String.format("PrivateStartService¥nReceived param: ¥"%s¥"", param),
                Toast.LENGTH_LONG).show();

        return Service.START_NOT_STICKY;
    }

    // The onDestroy gets called only one time when the service stops.
    @Override
    public void onDestroy() {
        Toast.makeText(this, "PrivateStartService - onDestroy()", Toast.LENGTH_SHORT).show();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // This service does not provide binding, so return null
        return null;
    }
}
```


Next is sample code for Activity which uses Private Service.

Points (Using a Service):

4. Use the explicit intent with class specified to call a service in the same application.
5. Sensitive information can be sent since the destination service is in the same application.
6. Handle the received result data carefully and securely, even though the data came from a service in the same application.

PrivateUserActivity.java

```
package org.jssec.android.service.privateservice;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class PrivateUserActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.privateservice_activity);
    }

    // --- StartService control ---

    public void onStartServiceClick(View v) {
        // *** POINT 4 *** Use the explicit intent with class specified to call a service in the same application.
        Intent intent = new Intent(this, PrivateStartService.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination service is in the same application.
        intent.putExtra("PARAM", "Sensitive information");

        startService(intent);
    }

    public void onStopServiceClick(View v) {
        doStopService();
    }

    @Override
    public void onStop() {
        super.onStop();
        // Stop service if the service is running.
        doStopService();
    }

    private void doStopService() {
        // *** POINT 4 *** Use the explicit intent with class specified to call a service in the same application.
        Intent intent = new Intent(this, PrivateStartService.class);
        stopService(intent);
    }

    // --- IntentService control ---

    public void onIntentServiceClick(View v) {
        // *** POINT 4 *** Use the explicit intent with class specified to call a service in the same application.
```

```
        Intent intent = new Intent(this, PrivateIntentService.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination service is in the same application.
        intent.putExtra("PARAM", "Sensitive information");

        startService(intent);
    }
}
```


4.4.1.2. Creating/Using Public Services

Public Service is the Service which is supposed to be used by the unspecified large number of applications. It's necessary to pay attention that it may receive the information (Intent etc.) which was sent by Malware. In case using public Service, It's necessary to pay attention that information(Intent etc.) to send may be received by Malware.

Sample code of how to use the startService type Service is shown below.

Points (Creating a Service):

1. Handle the received intent carefully and securely.
2. When returning a result, do not include sensitive information.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.publicservice"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Most standard Service -->
        <!-- If the service is disclosed to the public, set exported=true -->
        <service android:name=".PublicStartService" android:exported="true">
            <intent-filter>
                <action android:name="org.jssec.android.service.publicservice.action.startservice" />
            </intent-filter>
        </service>

        <!-- Public Service derived from IntentService class -->
        <!-- If the service is disclosed to the public, set exported=true -->
        <service android:name=".PublicIntentService" android:exported="true">
            <intent-filter>
                <action android:name="org.jssec.android.service.publicservice.action.intentservice" />
            </intent-filter>
        </service>

        </application>

</manifest>
```

PublicIntentService.java

```
package org.jssec.android.service.publicservice;

import android.app.IntentService;
import android.content.Intent;
import android.widget.Toast;

public class PublicIntentService extends IntentService{
```

```
    /**
     * Default constructor must be provided when a service extends IntentService class.
     * If it does not exist, an error occurs.
     */
    public PublicIntentService() {
        super("CreatingTypeBService");
    }

    // The onCreate gets called only one time when the Service starts.
    @Override
    public void onCreate() {
        super.onCreate();

        Toast.makeText(this, this.getClass().getSimpleName() + " - onCreate()", Toast.LENGTH_SHORT).show();
    }

    // The onHandleIntent gets called each time after the startService gets called.
    @Override
    protected void onHandleIntent(Intent intent) {
        // *** POINT 1 *** Handle intent carefully and securely.
        // Since it's public service, the intent may come from malicious application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this, String.format("Recieved parameter ¥"%s¥"", param), Toast.LENGTH_LONG).show();
    }


    // The onDestroy gets called only one time when the service stops.
    @Override
    public void onDestroy() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - onDestroy()", Toast.LENGTH_SHORT).show();
    }

}
```


Next is sample code for Activity which uses Public Service.

Points (Using a Service):

3. Do not send sensitive information.
4. When receiving a result, handle the result data carefully and securely.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.publicserviceuser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">
        <activity
            android:name=".PublicUserActivity"
            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>

</manifest>
```

PublicUserActivity.java

```
package org.jssec.android.service.publicserviceuser;

import org.jssec.android.service.publicserviceuser.R;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class PublicUserActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.publicservice_activity);
    }

    // --- StartService control ---

    public void onStartServiceClick(View v) {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.startservice");

        // *** POINT 3 *** Do not send sensitive information.
```

```
        intent.putExtra("PARAM", "Not sensitive information");

        startService(intent);
    }


    public void onStopServiceClick(View v) {
        doStopService();
    }

    // --- IntentService control ---

    public void onIntentServiceClick(View v) {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.intentservice");

        // *** POINT 3 *** Do not send sensitive information.
        intent.putExtra("PARAM", "Not sensitive information");

        startService(intent);
    }

    @Override
    public void onStop(){
        super.onStop();
        // Stop service if the service is running.
        doStopService();
    }

    // Stop service
    private void doStopService() {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.startservice");
        stopService(intent);
    }
}
```


4.4.1.3. Creating/Using Partner Services

Partner Service is Service which can be used only by the particular applications. System consists of partner company's application and In house application, this is used to protect the information and features which are handled between a partner application and In house application.

Following is an example of AIDL bind type Service.

Points (Creating a Service):

1. Do not define the intent filter.
2. Verify that the certificate of the requesting application has been registered in the own white list.
3. Do not (Cannot) recognize whether the requesting application is partner or not by onBind (onStartCommand, onHandleIntent).
4. Handle the received intent carefully and securely, even though the intent was sent from a partner application.
5. Return only information that is granted to be disclosed to a partner application.

In addition, refer to "5.2.1.3 How to verify the hash value of an application's certificate" for how to verify the certification hash value of destination application which is specified to white list.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.partnerservice.aidl"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

       <!-- Service using AIDL -->
       <!-- *** POINT 1 *** 1.  Do not define the intent filter. -->
       <service
           android:name="org.jssec.android.service.partnerservice.aidl.PartnerAIDLService"
           android:exported="true" />
    </application>

</manifest>
```

In this example, 2 AIDL files are to be created. One is for callback interface to give data from Service to Activity. The other one is Interface to give data from Activity to Service and to get information. In addition, package name that is described in AIDL file should be consistent with directory hierarchy in which AIDL file is created, same like package name described in java file.

IExclusiveAIDLServiceCallback.aidl

```
package org.jssec.android.service.exclusiveservice.aidl;
```

```
interface IExclusiveAIDLServiceCallback {
    /**
     * It's called when the value is changed.
     */
    void valueChanged(String info);
}
```

IExclusiveAIDLService.aidl

```
package org.jssec.android.service.exclusiveservice.aidl;

import org.jssec.android.service.exclusiveservice.aidl.IExclusiveAIDLServiceCallback;

interface IExclusiveAIDLService {

    /**
     * Register Callback.
     */
    void registerCallback(IExclusiveAIDLServiceCallback cb);

    /**
     * Get Information
     */
    String getInfo(String param);

    /**
     * Unregister Callback
     */
    void unregisterCallback(IExclusiveAIDLServiceCallback cb);
}
```

PartnerAIDLService.java

```
package org.jssec.android.service.partnerservice.aidl;

import org.jssec.android.service.partnerservice.aidl.IPartnerAIDLService;
import org.jssec.android.service.partnerservice.aidl.IPartnerAIDLServiceCallback;
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteCallbackList;
import android.os.RemoteException;
import android.widget.Toast;

public class PartnerAIDLService extends Service {
       private static final int REPORT_MSG = 1;
       private static final int GETINFO_MSG = 2;

    // The value which this service informs to client
    private int mValue = 0;

    // *** POINT 2 *** Verify that the certificate of the requesting application has been registered in the own white
 list.
    private static PkgCertWhitelists sWhitelists = null;
```



```
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application "org.jssec.android.service.exclusiveservice.aidluse
r"
        sWhitelists.add("org.jssec.android.service.exclusiveservice.aidluser", isdebug ?
                // Certificate hash value of  debug.keystore "androiddebugkey"
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of keystore "partner key"
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // Register other partner applications in the same way
    }

    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // Object to register callback
    // Methods which RemoteCallbackList provides are thread-safe.
    private final RemoteCallbackList<IPartnerAIDLServiceCallback> mCallbacks =
        new RemoteCallbackList<IPartnerAIDLServiceCallback>();

    // Handler to send data when callback is called.
    private final Handler mHandler = new Handler() {
        @Override public void handleMessage(Message msg) {
            switch (msg.what) {
            case REPORT_MSG: {
                // Start broadcast
                // To call back on to the registered clients, use beginBroadcast().
                // beginBroadcast() makes a copy of the currently registered callback list.
                final int N = mCallbacks.beginBroadcast();
                for (int i = 0; i < N; i++) {
                    IPartnerAIDLServiceCallback target = mCallbacks.getBroadcastItem(i);
                    try {
                        // *** POINT 5 *** Information that is granted to disclose to partner applications can be retur
ned.
                        target.valueChanged("Information disclosed to partner application (callback from Service) No."
 + (++mValue));

                    } catch (RemoteException e) {
                        // Callbacks are managed by RemoteCallbackList, do not unregister callbacks here.
                        // RemoteCallbackList.kill() unregister all callbacks
                    }
                }
                // finishBroadcast() cleans up the state of a broadcast previously initiated by calling beginBroadcast
().
                mCallbacks.finishBroadcast();

                // Repeat after 10 seconds
                sendEmptyMessageDelayed(REPORT_MSG, 10000);
                break;
             }
            case GETINFO_MSG: {
                Toast.makeText(PartnerAIDLService.this,
                      (String)msg.obj, Toast.LENGTH_LONG).show();
                break;
              }
```

```
            default:
                super.handleMessage(msg);
                break;
            } // switch
        }
    };

    // Interfaces defined in AIDL
    private final IPartnerAIDLService.Stub mBinder = new IPartnerAIDLService.Stub() {
        private final boolean checkPartner() {
            Context ctx = PartnerAIDLService.this;
            if (!PartnerAIDLService.checkPartner(ctx, Utils.getPackageNameFromPid(ctx, getCallingPid()))) {
                Toast.makeText(ctx, "Requesting application is not partner application.", Toast.LENGTH_LONG).show();
                return false;
            }
           return true;
        }
        public void registerCallback(IPartnerAIDLServiceCallback cb) {
            // *** POINT 2 *** Verify that the certificate of the requesting application has been registered in the ow
n white list.
          if (!checkPartner()) {
                return;
            }
          if (cb != null) mCallbacks.register(cb);
        }
        public String getInfo(String param) {
            // *** POINT 2 *** Verify that the certificate of the requesting application has been registered in the ow
n white list.
            if (!checkPartner()) {
                return null;
            }
            // *** POINT 4 *** Handle the received intent carefully and securely,
            // even though the intent was sent from a partner application
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            Message msg = new Message();
            msg.what = GETINFO_MSG;
            msg.obj = String.format("Method calling from partner application. Recieved ¥"%s¥"", param);
            PartnerAIDLService.this.mHandler.sendMessage(msg);

            // *** POINT 5 *** Return only information that is granted to be disclosed to a partner application.
            return new String("Information disclosed to partner application (method from Service)");
        }

        public void unregisterCallback(IPartnerAIDLServiceCallback cb) {
            // *** POINT 2 *** Verify that the certificate of the requesting application has been registered in the ow
n white list.
           if (!checkPartner()) {
                return;
            }

           if (cb != null) mCallbacks.unregister(cb);
        }
    };

    @Override
    public IBinder onBind(Intent intent) {
        // *** POINT 2 *** Verify that the certificate of the requesting application has been registered in the own wh
ite list.
        // So requesting application must be validated in methods defined in AIDL every time.
        return mBinder;
```



```
    }

    @Override
    public void onCreate() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - onCreate()", Toast.LENGTH_SHORT).show();

        // During service is running, inform the incremented number periodically.
        mHandler.sendEmptyMessage(REPORT_MSG);
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, this.getClass().getSimpleName() + " - onDestroy()", Toast.LENGTH_SHORT).show();

        // Unregister all callbacks
        mCallbacks.kill();

        mHandler.removeMessages(REPORT_MSG);
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;
```

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


Next is sample code of Activity which uses partner only Service.

Points (Using a Service):

6. Verify if the certificate of the target application has been registered in the own white list.
7. Return only information that is granted to be disclosed to a partner application.
8. Use the explicit intent to call a partner service.
9. Handle the received result data carefully and securely, even though the data came from a partner application.

ExclusiveAIDLUserActivity.java

```
package org.jssec.android.service.partnerservice.aidluser;

import org.jssec.android.service.partnerservice.aidl.IPartnerAIDLService;
import org.jssec.android.service.partnerservice.aidl.IPartnerAIDLServiceCallback;
import org.jssec.android.service.partnerservice.aidluser.R;
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteException;
import android.view.View;
import android.widget.Toast;

public class PartnerAIDLUserActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    private final static int MGS_VALUE_CHANGED = 1;

    // *** POINT 6 *** Verify if the certificate of the target application has been registered in the own white list.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner service application "org.jssec.android.service.partnerservice.a
idl"
        sWhitelists.add("org.jssec.android.service.partnerservice.aidl", isdebug ?
                // Certificate hash value of debug.keystore "androiddebugkey"
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // Certificate hash value of  keystore "my company key"
                "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // Register other partner service applications in the same way
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
```

```
    }

    // Information about destination (requested) partner activity.
    private static final String TARGET_PACKAGE =  "org.jssec.android.service.partnerservice.aidl";
    private static final String TARGET_CLASS = "org.jssec.android.service.partnerservice.aidl.partnerAIDLService";

    private final Handler mHandler = new Handler() {
        @Override public void handleMessage(Message msg) {
            switch (msg.what) {
                case MGS_VALUE_CHANGED: {
                      String info = (String)msg.obj;
                    Toast.makeText(mContext, String.format("Received ¥"%s¥" with callback.", info), Toast.LENGTH_SHORT
).show();
                    break;
                  }
                default:
                    super.handleMessage(msg);
                    break;
               } // switch
        }
    };

    // Interfaces defined in AIDL. Receive notice from service
    private final IPartnerAIDLServiceCallback.Stub mCallback =
        new IPartnerAIDLServiceCallback.Stub() {
            @Override
            public void valueChanged(String info) throws RemoteException {
                Message msg = mHandler.obtainMessage(MGS_VALUE_CHANGED, info);
                mHandler.sendMessage(msg);
            }
    };

    // Interfaces defined in AIDL. Inform service.
    private IPartnerAIDLService mService = null;

    // Connection used to connect with service. This is necessary when service is implemented with bindService().
    private ServiceConnection mConnection = new ServiceConnection() {

        // This is called when the connection with the service has been established.
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            mService = IPartnerAIDLService.Stub.asInterface(service);

            try{
                // connect to service
                mService.registerCallback(mCallback);

            }catch(RemoteException e){
                // service stopped abnormally
            }

            Toast.makeText(mContext, "Connected to service", Toast.LENGTH_SHORT).show();
        }

        // This is called when the service stopped abnormally and connection is disconnected.
        @Override
        public void onServiceDisconnected(ComponentName className) {
            Toast.makeText(mContext, "Disconnected from service", Toast.LENGTH_SHORT).show();
        }
    };
```



```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.partnerservice_activity);

        mContext = this;
    }

    // --- StartService control ---

    public void onStartServiceClick(View v) {
        // Start bindService
        doBindService();
    }

    public void onGetInfoClick(View v) {
        getServiceinfo();
    }

    public void onStopServiceClick(View v) {
        doUnbindService();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        doUnbindService();
    }

    /**
     * Connect to service
     */
    private void doBindService() {
        if (!mIsBound){
            // *** POINT 6 *** Verify if the certificate of the target application has been registered in the own whit
e list.
            if (!checkPartner(this, TARGET_PACKAGE)) {
                Toast.makeText(this, "Destination(Requested) sevice application is not registered in white list.", Toa
st.LENGTH_LONG).show();
                return;
            }

            Intent intent = new Intent();

            // *** POINT 7 *** Return only information that is granted to be disclosed to a partner application.
            intent.putExtra("PARAM", "Information disclosed to partner application");

            // *** POINT 8 *** Use the explicit intent to call a partner service.
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

          bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
          mIsBound = true;
        }
    }

    /**
     * Disconnect service
     */
```

```
    private void doUnbindService() {
        if (mIsBound) {
            // Unregister callbacks which have been registered.
            if(mService != null){
                try{
                    mService.unregisterCallback(mCallback);
                }catch(RemoteException e){
                    // Service stopped abnormally
                    // Omitted, since it' s sample.
                }
            }

          unbindService(mConnection);

          Intent intent = new Intent();

        // *** POINT 8 *** Use the explicit intent to call a partner service.
          intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

          stopService(intent);

          mIsBound = false;
        }
    }

    /**
     * Get information from service
     */
    void getServiceinfo() {
        if (mIsBound && mService != null) {
            String info = null;

            try {
                // *** POINT 7 *** Return only information that is granted to be disclosed to a partner application.
                info = mService.getInfo(new String("Information disclosed to partner application (method from activity
)"));
            } catch (RemoteException e) {
            }
            // *** POINT 9 *** Handle the received result data carefully and securely,
            // even though the data came from a partner application.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(mContext, String.format("Received ¥"%s¥" from service.", info), Toast.LENGTH_SHORT).show()
;
         }
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();
```



```
    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
```

```
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


4.4.1.4. Creating/Using In-house Services

In-house Services are the Services which are prohibited to be used by applications other than in-house applications. They are used in applications developed internally that want to securely share information and functionality.

Following is an example which uses Messenger bind type Service.

Points (Creating a Service):

1. Define an in-house signature permission.
2. Require the in-house signature permission.
3. Do not define the intent filter.
4. Verify that the in-house signature permission is defined by an in-house application.
5. Handle the received intent carefully and securely, even though the intent was sent from an in-house application.
6. Sensitive information can be returned since the requesting application is in-house.
7. When exporting an APK from Eclipse, sign the APK with the same developer key as the requesting application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.inhouseservice.messenger"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 1 *** Define an in-house signature permission -->
    <permission
        android:name="org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Service using Messenger -->
         <!-- *** POINT 2 *** Require the in-house signature permission -->
         <!-- *** POINT 3 *** Do not define the intent filter -->
        <service
            android:name="org.jssec.android.service.inhouseservice.messenger.InhouseMessengerService"
            android:exported="true"
            android:permission="org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION" />
    </application>

</manifest>
```

InhouseMessengerService.java

```
package org.jssec.android.service.inhouseservice.messenger;
```

```
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import java.util.ArrayList;
import java.util.Iterator;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.widget.Toast;

public class InhouseMessengerService extends Service{
    // In-house signature permission
    private static final String MY_PERMISSION = "org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }


    // Manage clients(destinations of sending data) in a list
    private ArrayList<Messenger> mClients = new ArrayList<Messenger>();

    // Messenger used when service receive data from client
    private final Messenger mMessenger = new Messenger(new ServiceSideHandler());

    // Handler which handles message received from client
    private class ServiceSideHandler extends Handler{
        @Override
        public void handleMessage(Message msg){
            switch(msg.what){
            case CommonValue.MSG_REGISTER_CLIENT:
                // Add messenger received from client
                mClients.add(msg.replyTo);
                break;
            case CommonValue.MSG_UNREGISTER_CLIENT:
                mClients.remove(msg.replyTo);
                break;
            case CommonValue.MSG_SET_VALUE:
                // Send data to client
                sendMessageToClients();
                break;
            default:
```



```
                super.handleMessage(msg);
               break;
            }
        }
    }

    /**
     * Send data to client
     */
    private void sendMessageToClients(){

        // *** POINT 6 *** Sensitive information can be returned since the requesting application is in-house.
        String sendValue = "Sensitive information (from Service)";

        // Send data to the registered client one by one.
        // Use iterator to send all clients even though clients are removed in the loop process.
        Iterator<Messenger> ite = mClients.iterator();
        while(ite.hasNext()){
            try {
                Message sendMsg = Message.obtain(null, CommonValue.MSG_SET_VALUE, null);

                Bundle data = new Bundle();
                data.putString("key", sendValue);
                sendMsg.setData(data);

                Messenger next = ite.next();
                next.send(sendMsg);

            } catch (RemoteException e) {
                // If client does not exits, remove it from a list.
                ite.remove();
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {

        // *** POINT 4 *** Verify that the in-house signature permission is defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "In-house defined signature permission is not defined by in-house application.", Toa
st.LENGTH_LONG).show();
            return null;
        }

        // *** POINT 5 *** Handle the received intent carefully and securely,
        // even though the intent was sent from an in-house application.
        // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this, String.format("Received parameter ¥"%s¥".", param), Toast.LENGTH_LONG).show();

        return mMessenger.getBinder();
    }

    @Override
    public void onCreate() {
        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();
    }

    @Override
```

```
    public void onDestroy() {
        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
```



```
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 7 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the requesting application.

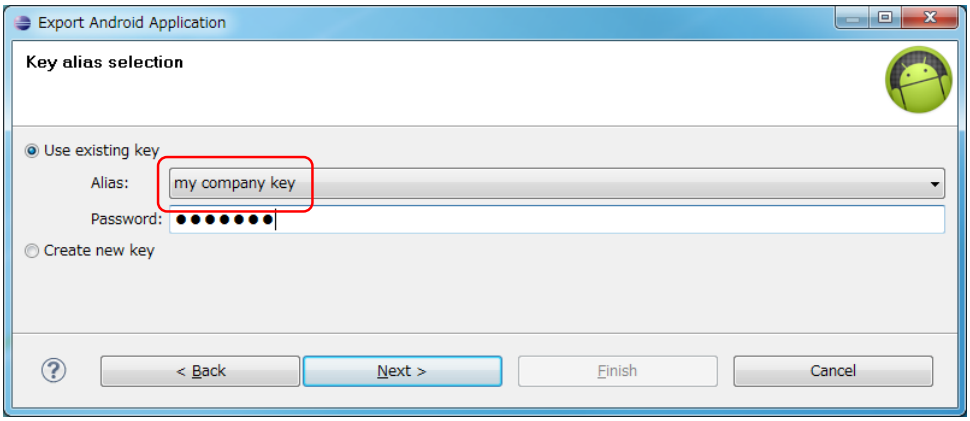


Figure 4.4-2

Next is the sample code of Activity which uses In house only Service.

Points (Using a Service):

8. Declare to use the in-house signature permission.
9. Verify that the in-house signature permission is defined by an in-house application.
10. Verify that the destination application is signed with the in-house certificate.
11. Sensitive information can be sent since the destination application is in-house.
12. Use the explicit intent to call an in-house service.
13. Handle the received result data carefully and securely, even though the data came from an in-house application.
14. When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.inhouseservice.messengeruser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 8 *** Declare to use the in-house signature permission.  -->
    <uses-permission
        android:name="org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name="org.jssec.android.service.inhouseservice.messengeruser.InhouseMessengerUserActivity"
            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>

</manifest>
```

InhouseMessengerUserActivity.java

```
package org.jssec.android.service.inhouseservice.messengeruser;

import org.jssec.android.service.inhouseservice.messengeruser.R;
import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
```

```
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.view.View;
import android.widget.Toast;

public class InhouseMessengerUserActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    // Destination (Requested) service application information
    private static final String TARGET_PACKAGE =  "org.jssec.android.service.inhouseservice.messenger";
    private static final String TARGET_CLASS = "org.jssec.android.service.inhouseservice.messenger.inhouseMessengerS
ervice";

    // In-house signature permission
    private static final String MY_PERMISSION = "org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Messenger used when this application receives data from service.
    private Messenger mServiceMessenger = null;

    // Messenger used when this application sends data to service.
    private final Messenger mActivityMessenger = new Messenger(new ActivitySideHandler());

    // Handler which handles message received from service
    private class ActivitySideHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case CommonValue.MSG_SET_VALUE:
                    Bundle data = msg.getData();
                    String info = data.getString("key");
                    // *** POINT 13 *** Handle the received result data carefully and securely,
                    // even though the data came from an in-house application
                    // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely
."
                    Toast.makeText(mContext, String.format("Received ¥"%s¥" from service.", info),
                            Toast.LENGTH_SHORT).show();
                    break;
                default:
                    super.handleMessage(msg);
```



```
        }
    }
}

// Connection used to connect with service. This is necessary when service is implemented with bindService().
private ServiceConnection mConnection = new ServiceConnection() {

    // This is called when the connection with the service has been established.
    @Override
    public void onServiceConnected(ComponentName className, IBinder service) {
        mServiceMessenger = new Messenger(service);
        Toast.makeText(mContext, "Connect to service", Toast.LENGTH_SHORT).show();

        try {
            // Send own messenger to service
            Message msg = Message.obtain(null, CommonValue.MSG_REGISTER_CLIENT);
            msg.replyTo = mActivityMessenger;
            mServiceMessenger.send(msg);
        } catch (RemoteException e) {
            // Service stopped abnormally
        }
    }

    // This is called when the service stopped abnormally and connection is disconnected.
    @Override
    public void onServiceDisconnected(ComponentName className) {
        mServiceMessenger = null;
        Toast.makeText(mContext, "Disconnected from service", Toast.LENGTH_SHORT).show();
    }
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.inhouseservice_activity);

    mContext = this;
}

// --- StartService control ---

public void onStartServiceClick(View v) {
    // Start bindService
    doBindService();
}

public void onGetInfoClick(View v) {
    getServiceinfo();
}

public void onStopServiceClick(View v) {
    doUnbindService();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    doUnbindService();
}
```

```
    /**
     * Connect to service
     */
    void doBindService() {
        if (!mIsBound){
            // *** POINT 9 *** Verify that the in-house signature permission is defined by an in-house application.
            if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
                Toast.makeText(this, "In-house defined signature permission is not defined by in-house application.",
Toast.LENGTH_LONG).show();
                return;
            }

            // *** POINT 10 *** Verify that the destination application is signed with the in-house certificate.
            if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
                Toast.makeText(this, "Destination(Requested) service application is not in-house application.", Toast
.LENGTH_LONG).show();
                return;
            }

          Intent intent = new Intent();

            // *** POINT 11 *** Sensitive information can be sent since the destination application is in-house one.
          intent.putExtra("PARAM", "Sensitive information");

            // *** POINT 12 *** Use the explicit intent to call an in-house service.
          intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

          bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
          mIsBound = true;
        }
    }

    /**
     * Disconnect service
     */
    void doUnbindService() {
        if (mIsBound) {
            unbindService(mConnection);
            mIsBound = false;
        }
    }

    /**
     * Get information from service
     */
    void getServiceinfo() {
        if (mServiceMessenger != null) {
            try {
                // Request sending information
                Message msg = Message.obtain(null, CommonValue.MSG_SET_VALUE);
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service stopped abnormally
            }
         }
    }

}
```


SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }
```

```
    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


*** Point14 *** When exporting an APK from Eclipse, sign the APK with the same developer key as the destination application.

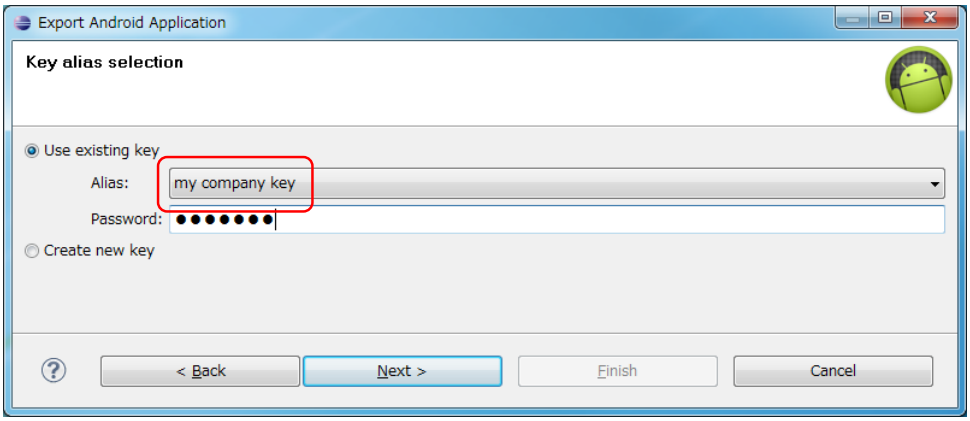


Figure 4.4-3

4.4.2. Rule Book

Implementing or using service, follow the rules below.

1.	Service that Is Used Only in an application, Must Be Set as Private	(Required)
2.	Handle the Received Data Carefully and Securely	(Required)
3.	Use the In-house Defined Signature Permission after Verifying If it's Defined by an In-house Application	(Required)
4.	Do Not Determine Whether the Service Provides its Functions, in onCreate	(Required)
5.	When Returning a Result Information, Pay Attention the Result Information Leakage from the Destination Application	(Required)
6.	Use the Explicit Intent if the Destination Service Is fixed	(Required)
7.	Verify the Destination Service If Linking with the Other Company's Application	(Required)
8.	When Providing an Asset Secondarily, the Asset should be protected with the Same Level Protection	(Required)
9.	Sensitive Information Should Not Be Sent As Much As Possible	(Recommended)

4.4.2.1. Service that Is Used Only in an application, Must Be Set as Private (Required)

Service that is used only in an application (or in same UID) must be set as Private. It avoids the application from receiving Intents from other applications unexpectedly and eventually prevents from damages such as application functions are used or application behavior becomes abnormal.

All you have to do in implementation is set exported attribute false when defining Service in AndroidManifest.xml.

AndroidManifest.xml

```
<!-- Private Service derived from Service class -->
<!-- *** POINT 1 *** Set false for the exported attribute explicitly. -->
<service android:name=".PrivateStartService" android:exported="false"/>
```

In addition, this is a rare case, but do not set Intent Filter when service is used only within the application. The reason is that, due to the characteristics of Intent Filter, public service in other application may be called unexpectedly though you intend to call Private Service within the application.

AndroidManifest.xml(Not recommended)

```
<!-- Private Service derived from Service class -->
<!-- *** POINT 1 *** Set false for the exported attribute explicitly. -->
<service android:name=".PrivateStartService" android:exported="false">
    <intent-filter>
        <action android:name="org.jssec.android.service.OPEN />
    </intent-filter>
</service>
```


See "4.4.3.1 Combination of Exported Attribute and Intent-filter Setting (In the Case of Service)."

4.4.2.2. Handle the Received Data Carefully and Securely (Required)

Same like Activity, In case of Service, when processing a received Intent data, the first thing you should do is input validation. Also in Service user side, it's necessary to verify the safety of result information from Service. Please refer to "4.1.2.5 Handling the Received Intent Carefully and Securely (Required)" and "4.1.2.9 Handle the Returned Data from a Requested Activity Carefully and Securely (Required)."

In Service, you should also implement calling method and exchanging data by Message carefully.

Please refer to "3.2 Handling Input Data Carefully and Securely"

4.4.2.3. Use the In-house Defined Signature Permission after Verifying If it's Defined by an In-house Application (Required)

Make sure to protect your in-house Services by defining in-house signature permission when creating the Service. Since defining a permission in the AndroidManifest.xml file or declaring a permission request does not provide adequate security, please be sure to refer to "5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission."

4.4.2.4. Do Not Determine Whether the Service Provides its Functions, in onCreate (Required)

Security checks such as Intent parameter verification or in-house-defined Signature Permission verification should not be included in onCreate, because when receiving new request during Service is running, process of onCreate is not executed. So, when implementing Service which is started by startService, judgment should be executed by onStartCommand (In case of using IntentService, judgment should be executed by onHandleIntent.) It's also same in the case when implementing Service which is started by bindService, judgment should be executed by onBind.

4.4.2.5. When Returning a Result Information, Pay Attention the Result Information Leakage from the Destination Application (Required)

Depends on types of Service, the reliability of result information destination application (callback receiver side/ Message destination) are different. Need to consider seriously about the information leakage considering the possibility that the destination may be Malware.

See, Activity "4.1.2.7 When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)", for details.

4.4.2.6. Use the Explicit Intent if the Destination Service Is fixed (Required)

When using a Service by implicit Intents, in case the definition of Intent Filter is same, Intent is sent to

the Service which was installed earlier. If Malware with the same Intent Filter defined intentionally was installed earlier, Intent is sent to Malware and information leakage occurs. On the other hand, when using a Service by explicit Intents, only the intended Service will receive the Intent so this is much safer.

There are some other points which should be considered, please refer to "4.1.2.8 Use the explicit Intents if the destination Activity is predetermined. (Required)."

4.4.2.7. Verify the Destination Service If Linking with the Other Company's Application (Required)

Be sure to sure a whitelist when linking with another company's application. You can do this by saving a copy of the company's certificate hash inside your application and checking it with the certificate hash of the destination application. This will prevent a malicious application from being able to spoof Intents. Please refer to sample code section "4.4.1.3 Creating/Using Partner Service" for the concrete implementation method.

4.4.2.8. When Providing an Asset Secondarily, the Asset should be protected with the Same Level Protection (Required)

When an information or function asset, that is protected by permission, is provided to another application secondhand, you need to make sure that it has the same required permissions needed to access the asset. In the Android OS permission security model, only an application that has been granted proper permissions can directly access a protected asset. However, there is a loophole because an application with permissions to an asset can act as a proxy and allow access to an unprivileged application. Substantially this is the same as redelegating permission so it is referred to as the "Permission Redelegation" problem. Please refer to "5.2.3.4 Permission Re-delegation Problem."

4.4.2.9. Sensitive Information Should Not Be Sent As Much As Possible (Recommended)

You should not send sensitive information to untrusted parties.

You need to consider the risk of information leakage when exchanging sensitive information with a Service. You must assume that all data in Intents sent to a Public Service can be obtained by a malicious third party. In addition, there is a variety of risks of information leakage when sending Intents to Partner or In-house Services as well depending on the implementation.

Not sending sensitive data in the first place is the only perfect solution to prevent information leakage therefore you should limit the amount of sensitive information being sent as much as possible. When it is necessary to send sensitive information, the best practice is to only send to a trusted Service and to make sure the information cannot be leaked through LogCat

4.4.3. Advanced Topics

4.4.3.1. Combination of Exported Attribute and Intent-filter Setting (In the Case of Service)

We have explained how to implement the four types of Services in this guidebook: Private Services, Public Services, Partner Services, and In-house Services. The various combinations of permitted settings for each type of exported attribute defined in the AndroidManifest.xml file and the intent-filter elements are defined in the table below. Please verify the compatibility of the exported attribute and intent-filter element with the Service you are trying to create.

Table 4.4-3

	Value of exported attribute		
	True	false	Not specified
Intent Filter defined	Public, Partner	(Do not Use)	Public, Partner
Intent Filter Not Defined	Public, Partner, In-house	Private	Private

The reason why an undefined intent filter and an exported attribute of false should not be used is that there is a loophole in Android's behavior, and because of how Intent filters work, other application's Services can be called unexpectedly.

Concretely, Android behaves as per below, so it's necessary to consider carefully when application designing.

- When multiple Services define the same content of intent-filter, the definition of Service within application installed earlier is prioritized.
- In case explicit Intent is used, prioritized Service is automatically selected and called by OS.

The system that unexpected call is occurred due to Android's behavior is described in the three figures below. Figure 4.4-4 is an example of normal behavior that Private Service (application A) can be called by implicit Intent only from the same application. Because only application A defines Intent-filter (action="X" in the Figure), it behaves normally. This is the normal behavior.

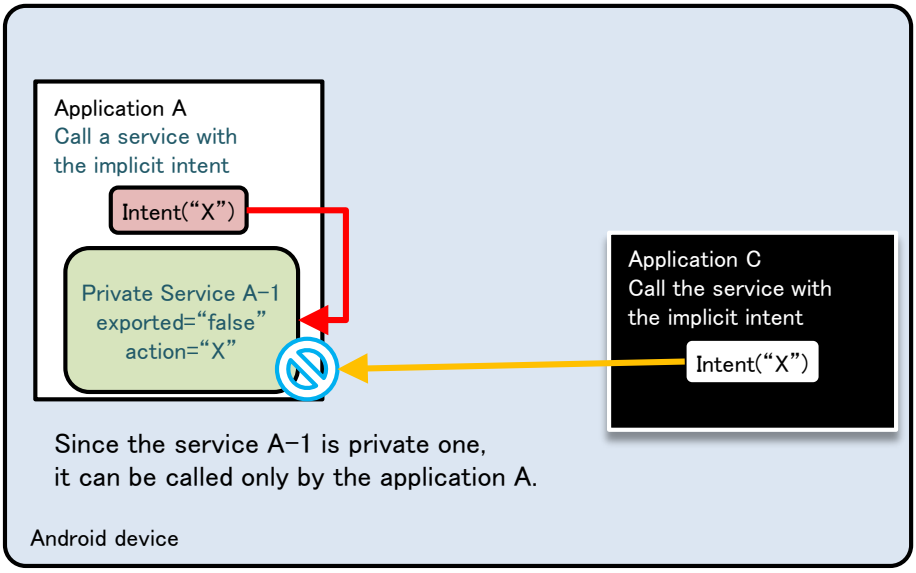


Figure 4.4-4

Figure 4.4-5 and Figure 4.4-6 below show a scenario in which the same Intent filter (action="X") is defined in Application B as well as Application A.

Figure 4.4-5 shows the scenario that applications are installed in the order, application A -> application B. In this case, when application C sends implicit Intent, calling Private Service (A-1) fails. On the other hand, since application A can successfully call Private Service within the application by implicit Intent as expected, there won't be any problems in terms of security (counter-measure for Malware).

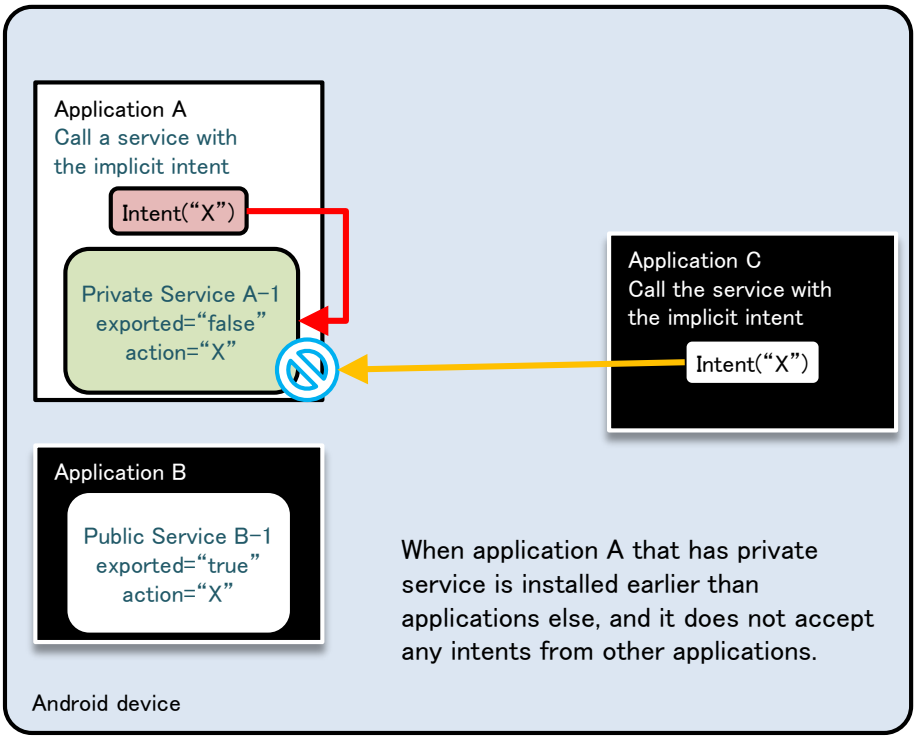


Figure 4.4-5

Figure 4.4-6 shows the scenario that applications are installed in the order, applicationB->applicationA. There is a problem here, in terms of security. It shows an example that application A tries to call Private Service within the application by sending implicit Intent, but actually Public Activity (B-1) in application B which was installed earlier, is called. Due to this loophole, it is possible that sensitive information can be sent from applicationA to applicationB. If applicationB is Malware, it will lead the leakage of sensitive information.

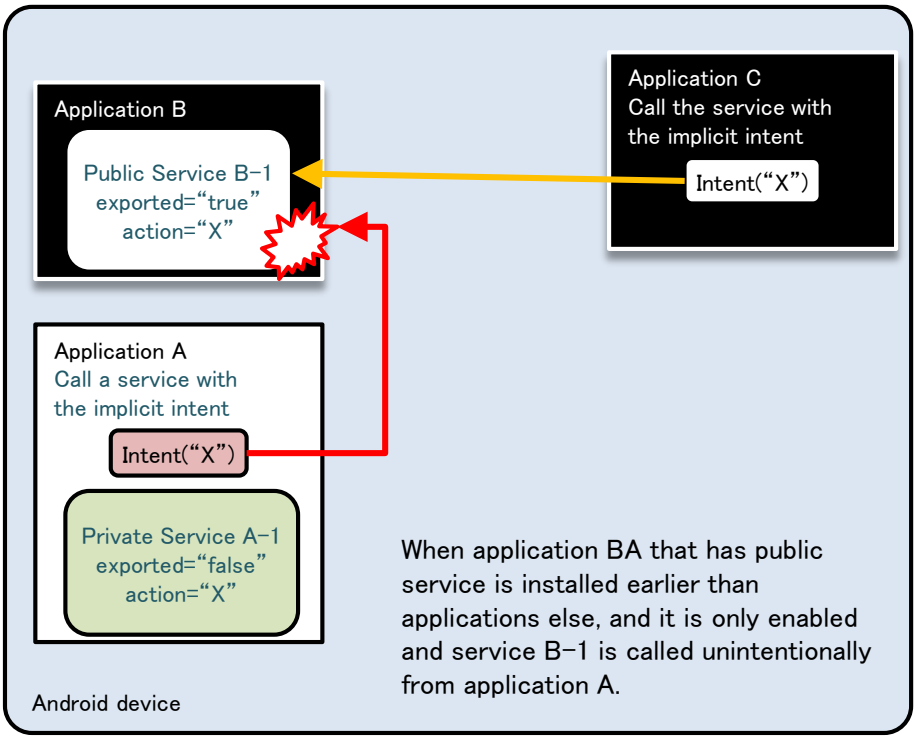


Figure 4.4-6

As shown above, using Intent filters to send implicit Intents to Private Service may result in unexpected behavior so it is best to avoid this setting.

4.4.3.2. How to Implement Service

Because methods for Service implementation are various and should be selected with consideration of security type which is categorized by sample code, each characteristics are briefly explained. It's divided roughly into the case using startService and the case using bindService. And it's also possible to create Service which can be used in both startService and bindService. Following items should be investigated to determine the implementation method of Service.

- Whether to disclose Service to other applications or not (Disclosure of Service)
- Whether to exchange data during running or not (Mutual sending /receiving data)
- Whether to control Service or not (Launch or complete)
- Whether to execute as another process (communication between processes)
- Whether to execute multiple processes in parallel (Parallel process)

Table 4.4-3 shows category of implementation methods and feasibility of each item.
"NG" stands for impossible case or case that another frame work which is different from the provided function is required.

Table 4.4-4 Category of implementation methods for Service

Category	Disclosure of Service	Mutual sending/receiving data	Control Service (Boot /Exit)	Communication between processes	Parallel process
startService type	OK	NG	OK	OK	NG
IntentService type	OK	NG	NG	OK	NG
local bind type	NG	OK	OK	NG	NG
Messenger bind type	OK	OK	OK	OK	NG
AIDL bind type	OK	OK	OK	OK	OK

startService type

This is the most basic Service. This inherits Service class, and executes processes by onStartCommand.

In user side, specify Service by Intent, and call by startService. Because data such as results cannot be returned to source of Intent directly, it should be achieved in combination with another method such as Broadcast. Please refer to "4.4.1.1 Creating/Using Private Service" for the concrete example.

Checking in terms of security should be done by onStartCommand, but it cannot be used for partner only Service since the package name of the source cannot be obtained.

IntentService type

IntentService is the class which was created by inheriting Service. Calling method is same as startService type. Following are characteristics compared with standard service (startService type.)

- Processing Intent is done by onHandleIntent (onStartCommand is not used.)
- It's executed by another thread.
- Process is to be queued.

Call is immediately returned because process is executed by another thread, and process towards Intents is sequentially executed by Queuing system. Each Intent is not processed in parallel, but it is also selectable depending on the product's requirement, as an option to simplify implementation. Since data such as results cannot be returned to source of Intent, it should be achieved in combination with another method such as Broadcast. Please refer to "4.4.1.2 Creating/Using Public Service" for the concrete example of implementation.

Checking in terms of security should be done by onHandleIntent, but it cannot be used for partner only Service since the package name of the source cannot be obtained.

local bind type

This is a method to implement local Service which works only within the process same as an application. Define the class which was derived from Binder class, and prepare to provide the feature (method) which was implemented in Service to caller side.

From user side, specify Service by Intent and call Service by using bindService. This is the most simple implementation method among all methods of binding Service, but it has limited usages since it cannot be launched by another process and also Service cannot be disclosed. See project "Service PrivateServiceLocalBind" which is included in Sample code, for the concrete implementation example.

From the security point of view, only private Service can be implemented.

Messenger bind type

This is the method to achieve the linking with Service by using Messenger system.

Since Messenger can be given as a Message destination from Service user side, the mutual data exchanging can be achieved comparatively easily. In addition, since processes are to be queued, it has a characteristic that behaves "thread-safe"ly. Parallel process for each process is not possible, but it is also selectable as an option to simplify the implementation depending on the product's requirement. Regarding user side, specify Service by Intent, and call Service by using bindService. See "4.4.1.4 Creating/Using In-house Service" for the concrete implementation example.

Security check in onBind or by Message Handler is necessary, however, it cannot be used for partner only Service since package name of source cannot be obtained.

AIDL bind type

This is a method to achieve linking with Service by using AIDL system. Define interface by AIDL, and provide features that Service has as a method. In addition, call back can be also achieved by implementing interface defined by AIDL in user side, Multi-thread calling is possible, but it's necessary to implement explicitly in Service side for exclusive process.

User side can call Service, by specifying Intent and using bindService. Please refer to "4.4.1.3 Creating/Using Partner Service" for the concrete implementation example.

Security must be checked in onBind for In-house only Service and by each method of interface defined by AIDL for partner only Service.

This can be used for all security types of Service which are described in this Guidebook.

Herein after, some cautions in terms of security when creating/operating database by using SQLite. Main points are appropriate setting of access right to database file, and counter-measures for SQL injection. Database which permits reading/writing database file from outside directly (sharing among multiple applications) is not supposed here, but suppose the usage in backend of Content Provider and in an application itself. In addition, it is recommended to adopt counter-measures mentioned below in case of handling not so much sensitive information, though handling a certain level of sensitive information is supposed here.

4.5.1. Sample Code

4.5.1.1. Creating/Operating Database

When handling database in Android application, appropriate arrangements of database files and access right setting (Setting for denying other application's access) can be achieved by using SQLiteOpenHelper[4]. Here is an example of easy application that creates database when it's launched, and executes searching /adding/changing/deleting data through UI. Sample code is what counter-measure for SQL injection is done, to avoid from incorrect SQL being executed against the input from outside.

[4] As regarding file storing, the absolute file path can be specified as the 2nd parameter (name) of SQLiteOpenHelper constructor. Therefore, need attention that the stored files can be read and written by the other applications if the SDCard path is specified.

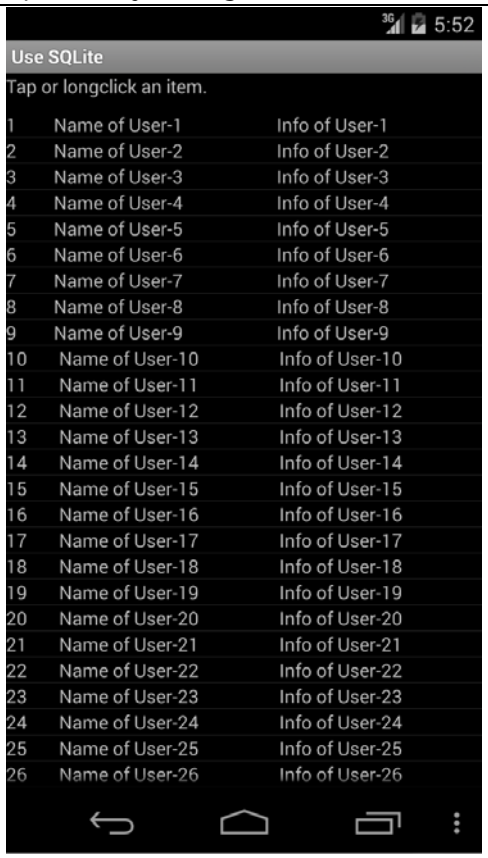


Figure 4.5-1

Points:

1. SQLiteOpenHelper should be used for database creation.
2. Use place holder.
3. Validate the input value according the application requirements.

SampleDbOpenHelper.java

```
package org.jssec.android.sqlite;

import org.jssec.android.sqlite.R;

import android.content.Context;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
import android.widget.Toast;

public class SampleDbOpenHelper extends SQLiteOpenHelper {
    private SQLiteDatabase          mSampleDb;            //Database to store the data to be handled

    public static SampleDbOpenHelper newHelper(Context context)
    {
        //*** POINT 1 *** SQLiteOpenHelper should be used for database creation.
        return new SampleDbOpenHelper(context);
    }

    public SQLiteDatabase getDb() {
        return mSampleDb;
    }
```



```
    //Open DB by Writable mode
    public void openDatabaseWithHelper() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (!mSampleDb.isReadOnly())//  Already opened by writable mode
                    return;
                mSampleDb.close();
              }
            mSampleDb  = getWritableDatabase(); //It's opened here.
        } catch (SQLException e) {
            //In case fail to construct database, output to log
            Log.e(mContext.getClass().toString(), mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext, R.string.DATABASE_OPEN_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return;
        }
    }

    //Open DB by ReadOnly mode.
    public void openDatabaseReadOnly() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (mSampleDb.isReadOnly())// Already opened by ReadOnly.
                    return;
                mSampleDb.close();
            }
            SQLiteDatabase.openDatabase(mContext.getDatabasePath(CommonData.DBFILE_NAME).getPath(), null, SQLiteDatab
ase.OPEN_READONLY);
        } catch (SQLException e) {
            //In case failed to construct database, output to log
            Log.e(mContext.getClass().toString(), mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext, R.string.DATABASE_OPEN_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return;
        }
    }

    //Database Close
    public void closeDatabase() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                mSampleDb.close();
            }
        } catch (SQLException e) {
            //In case failed to construct database, output to log
            Log.e(mContext.getClass().toString(), mContext.getString(R.string.DATABASE_CLOSE_ERROR_MESSAGE));
            Toast.makeText(mContext, R.string.DATABASE_CLOSE_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return;
        }
    }

    //Remember Context
    private Context mContext;

    //Table creation command
    private static final String CREATE_TABLE_COMMANDS
            = "CREATE TABLE " + CommonData.TABLE_NAME + " ("
            + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "idno INTEGER UNIQUE, "
            + "name VARCHAR(" + CommonData.TEXT_DATA_LENGTH_MAX + ") NOT NULL, "
            + "info VARCHAR(" + CommonData.TEXT_DATA_LENGTH_MAX + ")"
```

```
            + ");";

    public SampleDbOpenHelper(Context context) {
        super(context, CommonData.DBFILE_NAME, null, CommonData.DB_VERSION);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(CREATE_TABLE_COMMANDS);  //Execute DB construction command
        } catch (SQLException e) {
            //In case failed to construct database, output to log
            Log.e(this.getClass().toString(), mContext.getString(R.string.DATABASE_CREATE_ERROR_MESSAGE));
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
        // It's to be executed when database version up. Write processes like data transition.
    }

}
```

DataSearchTask.java (SQLite Database project)

```
package org.jssec.android.sqlite.task;

import org.jssec.android.sqlite.CommonData;
import org.jssec.android.sqlite.DataValidator;
import org.jssec.android.sqlite.MainActivity;
import org.jssec.android.sqlite.R;

import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.AsyncTask;
import android.util.Log;


//Data search task
public class DataSearchTask extends AsyncTask<String, Void, Cursor> {
    private MainActivity    mActivity;
    private SQLiteDatabase      mSampleDB;

    public DataSearchTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Cursor doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];
        String  cols[]  =   {"_id", "idno","name","info"};

        Cursor cur;
```



```
        //*** POINT 3 *** Validate the input value according the application requirements.
        if (!DataValidator.validateData(idno, name, info))
        {
            return null;
        }

        //When all parameters are null, execute all search
        if ((idno == null || idno.length() == 0) &&
                (name == null || name.length() == 0) &&
                (info == null || info.length() == 0) ) {
            try {
                cur = mSampleDB.query(CommonData.TABLE_NAME, cols, null, null, null, null, null);
            } catch (SQLException e) {
                Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
                return null;
            }
            return cur;
        }

        //When No is specified, execute searching by No
        if (idno != null && idno.length() > 0) {
            String selectionArgs[] = {idno};

            try {
                //*** POINT 2 *** Use place holder.
                cur = mSampleDB.query(CommonData.TABLE_NAME, cols, "idno = ?", selectionArgs, null, null, null);
            } catch (SQLException e) {
                Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
                return null;
            }
            return cur;
        }

        //When Name is specified, execute perfect match search by Name
        if (name != null && name.length() > 0) {
            String selectionArgs[] = {name};
            try {
                //*** POINT 2 *** Use place holder.
                cur = mSampleDB.query(CommonData.TABLE_NAME, cols, "name = ?", selectionArgs, null, null, null);
            } catch (SQLException e) {
                Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
                return null;
            }
            return cur;
        }

        //Other than above, execute partly match searching with the condition of info.
        String argString = info.replaceAll("@", "@@"); //Escape $ in info which was received as input.
        argString = argString.replaceAll("%", "@%"); //Escape % in info which was received as input.
        argString = argString.replaceAll("_", "@_"); //Escape _ in info which was received as input.
        String selectionArgs[] = {argString};

        try {
            //*** POINT 2 *** Use place holder.
            cur = mSampleDB.query(CommonData.TABLE_NAME, cols, "info LIKE '%' || ? || '%' ESCAPE '@'", selectionArgs,
null, null, null);
        } catch (SQLException e) {
            Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
            return null;
        }
```

```
        return cur;
    }

    @Override
    protected void onPostExecute(Cursor resultCur) {
        mActivity.updateCursor(resultCur);
    }
}
```

DataValidator.java

```
package org.jssec.android.sqlite;

public class DataValidator {
    //Validate the Input value
    //validate numeric characters
    public static boolean validateNo(String idno) {
        //null and blank are OK
        if (idno == null || idno.length() == 0) {
           return true;
        }

        //Validate that it's numeric character.
        try {
            if (!idno.matches("[1-9][0-9]*")) {
                //Error if it's not numeric value
                return false;
            }
        } catch (NullPointerException e) {
            //Detected an error
            return false;
        }

        return true;
    }

    // Validate the length of a character string
    public static boolean validateLength(String str, int max_length) {
       //null and blank are OK
       if (str == null || str.length() == 0) {
              return true;
       }

       //Validate the length of a character string is less than MAX
       try {
           if (str.length() > max_length) {
               //When it's longer than MAX, error
               return false;
           }
       } catch (NullPointerException e) {
           //Bug
           return false;
       }

       return true;
    }

   // Validate the Input value
    public static boolean validateData(String idno, String name, String info) {
```



```
        if (!validateNo(idno)) {
            return false;
         }
        if (!validateLength(name, CommonData.TEXT_DATA_LENGTH_MAX)) {
            return false;
         }
        if (!validateLength(info, CommonData.TEXT_DATA_LENGTH_MAX)) {
            return false;
         }
        return true;
    }
}
```

4.5.2. Rule Book

Using SQLite, follow the rules below accordingly.

1.	Set DB File Location and Access Right Correctly (Required)
2.	Use Content Provider for Access Control When Sharing DB Data with Other Application (Required)
3.	Place Holder Must Be Used in the Case Handling Variable Parameter during DB Operation. (Required)

4.5.2.1. Set DB File Location and Access Right Correctly (Required)

Considering the protection of DB file data, DB file location and access right setting is the very important elements that need to be considered together.

For example, even if file access right is set correctly, a DB file can be accessed from anybody in case that it is arranged in a location which access right cannot be set, e.g. SD card. And in case that it's arranged in application directory, if the access right is not correctly set, it will eventually allow the unexpected access. Following are some points to be met regarding the correct allocation and access right setting, and the methods to realize them.

About location and access right setting, considering in terms of protecting DB file (data), it's necessary to execute 2 points as per below.

1. Location
   Locate in file path that can be obtained by Context#getDatabasePath(String name),or in some cases, directory that can be obtained by Context#getFilesDir[5].
2. Access right
   Set to MODE_PRIVATE (=it can be accessed only by the application which creates file) mode.

By executing following 2 points, DB file which cannot be accessed by other applications can be created. Here are some methods to execute them.

1. Use SQLiteOpenHelper
2. Use Context#openOrCreateDatabase

When creating DB file, SQLiteDatabase#openOrCreateDatabase can be used. However, when using this method, DB files which can be read out from other applications are created, in some Android smartphone devices. So it is recommended to avoid this method, and using other methods. Each characteristics for the above 2 methods are as per below.

[5] Both methods provide the path under (package) directory which is able to be read and written only by the specified application.

Using SQLiteOpenHelper

When using SQLiteOpenHelper, developers don't need to be worried about many things. Create a class derived from SQLiteOpenHelper, and specify DB name (which is used for file name)[6] to constructer's parameter, then DB file which meets above security requirements, are to be created automatically.

Refer to specific usage method for "4.5.1.1 Creating/Operating Database" for how to use.

Using Context#openOrCreateDatabase

When creating DB by using Context#openOrCreateDatabase method, file access right should be specified by option, in this case specify MODE_PRIVATE explicitly.

Regarding file arrangement, specifying DB name (which is to be used to file name) can be done as same as SQLiteOpenHelper, a file is to be created automatically, in the file path which meets the above mentioned security requirements. However, full path can be also specified, so it's necessary to pay attention that when specifying SD card, even though specifying MODE_PRIVATE, other applications can also access.

Example to execute accsee permission setting to DB explicitly:MainActivity.java

```
public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       //Construct database
       try {
          //Create DB by setting MODE_PRIVATE
          db = Context.openOrCreateDatabase("Sample.db",
                                              MODE_PRIVATE, null);
       } catch (SQLException e) {
           //In case failed to construct DB, log output
           Log.e(this.getClass().toString(), getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
           return;
       }
       //Ommit other initial process
   }
```

FYI, there are following 3 types of access right setting including MODE_PRIVATE. MODE_WORLD_READABLE and MODE_WORLD_WRITABLE can be specified together by OR calculation. When using other than MODE_PRIVATE, need to consider carefully along with the application requirements.

- MODE_PRIVATE Only creator application can read and write

[6] (Undocumented in Android reference) Since the full file path can be specified as the database name inSQLiteOpenHelper implementation, need attention that specifying the place (path) which does not have access control feature (e.g. sdcards) unintentionally.

- MODE_WORLD_READABLE Creator application can read and write, Others can only read in
- MODE_WORLD_WRITABLE Creator application can read and write, Others can only write in

4.5.2.2. Use Content Provider for Access Control When Sharing DB Data with Other Application (Required)

The method to share DB data with other application is that create DB file as WORLD_READABLE, WORLD_WRITABLE, to other applications to access directly. However, this method cannot limit applications which access to DB or operations to DB, so data can be read-in or written by unexpected party (application). As a result, it can be considered that some problems may occur in confidentiality or consistency of data, or it may be an attack target of Malware.

As mentioned above, when sharing DB data with other applications in Android, it's strongly recommended to use Content Provider. By using Content Provider, there are some merits, not only the merits from the security point of view which is the access control on DB can be achieved, but also merits from the designing point of view which is DB scheme structure can be hidden into Content Provider.

4.5.2.3. Place Holder Must Be Used in the Case Handling Variable Parameter during DB Operation. (Required)

In the sense that preventing from SQL injection, when incorporating the arbitrary input value to SQL statement, placeholder should be used. There are 2 methods as per below to execute SQL using placeholder.

1. Get SQLiteStatement by using SQLiteDatabase#compileStatement(), and after that place parameter to placeholder by using SQLiteStatement#bindString() or bindLong() etc.
2. When calling execSQL(), insert(), update(), delete(), query(), rawQuery() and replace() etc in SQLiteDatabese class, use SQL statement which has placeholder.

In addition, when executing SELECT command, by using SQLiteDatabase#compileStatement(), there is a limitation that "only the top 1 element can be obtained as a result of SELECT command," so usages are limited.

In either method, the data content which is given to placeholder is better to be checked in advance according the application requirements. Following is the further explanation for each method.

When Using SQLiteDatabase#compileStatement():

This is so called prepared statement; data is given to placeholder in the following steps.

1. Get the SQL statement which includes placeholder by using SQLiteDatabase#compileStatement(), as SQLiteStatement.
2. Set the created as SQLiteStatement objects to placeholder by using the method like bindLong() and bindString().

3. Execute SQL by method like execute() etc of ExecSQLiteStatement object.

Use case of prepared statement:DataInsertTask.java (an extra)

```
//Adding data task
public class DataInsertTask extends AsyncTask<String, Void, Void> {
    private MainActivity    mActivity;
    private SQLiteDatabase  mSampleDB;

    public DataInsertTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Void doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];

        //*** POINT 3 *** Validate the input value according the application requirements.
        if (!DataValidator.validateData(idno, name, info))
        {
                 return null;
        }
        // Adding data task
        //*** POINT 2 *** Use place holder
        String commandString = "INSERT INTO " + CommonData.TABLE_NAME + " (idno, name, info) VALUES (?, ?, ?)";
        SQLiteStatement sqlStmt = mSampleDB.compileStatement(commandString);
        sqlStmt.bindString(1, idno);
        sqlStmt.bindString(2, name);
        sqlStmt.bindString(3, info);
        try {
            sqlStmt.executeInsert();
        } catch (SQLException e) {
            Log.e(DataInsertTask.class.toString(), mActivity.getString(R.string.UPDATING_ERROR_MESSAGE));
        } finally {
                 sqlStmt.close();
        }
        return null;
    }

    ... Abbreviation ...
}
```

This is a type that SQL statement to be executed as object is created in advance, and parameters are allocated to it. The process to execute is fixed, so there's no room for SQL injection to occur. In addition, there is a merit that process efficiency is enhanced by reutilizing SQLiteStatement object.

In the Case Using Method for Each Process which SQLiteDatabase provides:

There are 2 types of DB operation methods that SQLiteDatabase provides. One is what SQL statement is used, and another is what SQL statement is not used. Methods that SQL statement is used are SQLiteDatabase# execSQL()/rawQuery() etc, and it's executed in the following steps.

1. Prepare SQL statement which includes placeholder.
2. Create data to allocate to placeholder.
3. Send SQL statement and data as parameter, and execute a method for process.

On the other hand, SQLiteDatabase#insert()/update()/delete()/query()/replace() etc is the method that SQL statement is not used. When using them, data should be sent as per the following steps.

1. In case there's data to insert /update to DB, register to ContentValues.
2. Send ContentValues as parameter, and execute a method for each process (In the following example, SQLiteDatabase#insert())

Use case of metod for each process (SQLiteDatabase#insert())

```
private SQLiteDatabase  mSampleDB;
private void addUserData(String idno, String name, String info) {

  //Validity check of the value(Type, range), escape process
  if (!validateInsertData(idno, name, info)) {
      //If failed to pass the validation, log output
      Log.e(this.getClass().toString(), getString(R.string.VALIDATION_ERROR_MESSAGE));
      return
  }

   //Prepare data to insert
   ContentValues insertValues = new ContentValues();
   insertValues.put("idno", idno);
   insertValues.put("name", name);
   insertValues.put("info", info);

   //Execute Inser
   try {
       mSampleDb.insert("SampleTable", null, insertValues);
   } catch (SQLException e) {
       Log.e(this.getClass().toString(), getString(R.string.DB_INSERT_ERROR_MESSAGE));
       return;
   }
}
```

In this example, SQL command is not directly written, for instead, a method for inserting which SQLiteDatabase provides, is used. SQL command is not directly used, so there's no room for SQL injection in this method, too.

4.5.3. Advanced Topics

4.5.3.1. When Using Wild Card in LIKE Predicate of SQL Statement, Escape Process Should Be Implemented

When using character string which includes wild card (%, _) of LIKE predicate, as input value of place holder, it will work as a wild card unless it is processed properly, so it's necessary to implement escape process in advance according the necessity. It is the case which escape process is necessary that wild card should be used as a single character ("%" or "_").

The actual escape process is executed by using ESCAPE clause as per below sample code.

Example of ESCAPE process in case of using LIKE

```
//Data search task
public class DataSearchTask extends AsyncTask<String, Void, Cursor> {
    private MainActivity       mActivity;
    private SQLiteDatabase     mSampleDB;
    private ProgressDialog     mProgressDialog;

    public DataSearchTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Cursor doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];
        String  cols[]  =   {"_id", "idno","name","info"};

        Cursor cur;

    ... Abbreviation ...

        //Execute like search(partly match) with the condition of info
        //Point:Escape process should be performed on characters which is applied to wild card
        String argString = info.replaceAll("@", "@@"); // Escape $ in info which was received as input
        argString = argString.replaceAll("%", "@%"); // Escape % in info which was received as input
        argString = argString.replaceAll("_", "@_"); // Escape _ in info which was received as input
        String selectionArgs[] = {argString};

        try {
            //Point:Use place holder
            cur = mSampleDB.query("SampleTable", cols, "info LIKE '%' || ? || '%' ESCAPE '@'",
                                  selectionArgs, null, null, null);
        } catch (SQLException e) {
            Toast.makeText(mActivity, R.string.SERCHING_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return null;
        }
        return cur;
    }

    @Override
    protected void onPostExecute(Cursor resultCur) {
```

```
        mProgressDialog.dismiss();
        mActivity.updateCursor(resultCur);
    }
}
```

4.5.3.2. Use External Input to SQL Command in which Place Holder Cannot Be Used

When executing SQL statement which process targets are DB objects like table creation/deletion etc, placeholder cannot be used for the value of table name. Basically, DB should not be designed using arbitrary character string which was input from outside in case that placeholder cannot be used for the value.

When placeholder cannot be used due to the restriction of specifications or features, whether the Input value is dangerous or not, should be verified before execution, and it's necessary to implement necessary processes.

Basically,

1. When using as character string parameter, escape or quote process for character should be made.
2. When using as numeric value parameter, verify that characters other than numeric value are not included.
3. When using as identifier or command, verify whether characters which cannot be used are not included, along with 1.

should be executed.

Reference: http://www.ipa.go.jp/security/vuln/documents/website_security_sql.pdf (Japanese)

4.5.3.3. Take a Countermeasure that Database Is Not Overwritten Unexpectedly

In case getting instance of DB by SQLiteOpenHelper#getReadableDatabase, getWritableDatabase, DB is to be opened in readable/writable state by using either method[7]. In addition, it's same to Context#openOrCreateDatabase, SQLiteDatabase#openOrCreateDatabase, etc. It means that contents of DB may be overwritten unexpectedly by application operation or by defects in implementation. Basically, it can be supported by the application's spec and range of implementation, but when implementing the function which requires only read in function like application's searching function etc, opening database by read-only, it may lead to simplify designing or inspection and furthermore, lead to enhance application quality, so it's recommended depends on the situation.

[7] getReableDatabase() returns the same object which can be got by getWritableDatabase. This spec is, in case writable object cannot be generated due to disc full etc, it will return Read- only object. (getWritableDatabase() will be execution error under the situation like disc full etc.)

Specifically, open database by specifying OPEN_READONLY to SQLiteDatabase#openDatabase.

Open database by read-only

```
... Ommit ...
// Open DB(DB should be created in advance)
SQLiteDatabase db
      = SQLiteDatabase.openDatabase(SQLiteDatabase.getDatabasePath("Sample.db"), null, OPEN_READONLY);
```

Reference: http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getReadableDatabase()

4.5.3.4. Verify the Validity of Input/Output Data of DB, According to Application's Requirement

SQLite is the database which is tolerant types, and it can store character type data into columns which is declared as Integer in DB. Regarding data in database, all data including numeric value type is stored in DB as character data of plain text. So searching of character string type, can be executed to Integer type column. (LIKE '%123%' etc.) In addition, the limitation for the value in SQLite (validity verification) is untrustful since data which is longer than limitation can be input in some case, e.g. VARCHAR(100).

So, applications which use SQLite, need to be very careful about this characteristics of DB, and it's necessary take actions according to application requirements, not to store unexpected data to DB or not to get unexpected data. Countermeasures are as per below 2 points.

1. When storing data in database, verify that type and length are matched.
2. When getting the value from database, verify whether data is beyond the supposed type and length, or not.

Following is an example of the code which verifies that the Input value is more than 1.

Verify that the Input value is more than 1 (Extract from MainActivity.java)

```
public class MainActivity extends Activity {

    ... Abbreviation ...

    //Process for adding
    private void addUserData(String idno, String name, String info) {
        //Check for No
        if (!validateNo(idno, CommonData.REQUEST_NEW)) {
            return;
        }

        //Inserting data process
        DataInsertTask task = new DataInsertTask(mSampleDbyhis);
        task.execute(idno, name, info);
    }

    ... Abbreviation ...
```

```
    private boolean validateNo(String idno, int request) {
        if (idno == null || idno.length() == 0) {
            if (request == CommonData.REQUEST_SEARCH) {
                //When search process, unspecified is considered as OK.
                return true;
            } else {
                //Other than search process, null and blank are error.
                Toast.makeText(this, R.string.IDNO_EMPTY_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        }

        //Verify that it's numeric character
        try {
            // Value which is more than 1
            if (!idno.matches("[1-9][0-9]*")) {
                //In case of not numeric character, error
                Toast.makeText(this, R.string.IDNO_NOT_NUMERIC_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        } catch (NullPointerException e) {
            //It never happen in this case
            return false;
        }

        return true;
    }

    ... Abbreviation...
}
```

4.5.3.5. Consideration – the Data Stored into Database

In SQLite implementation, when storing data to file is as per below.

- All data including numeric value type are stored into DB file as character data of plain text.
- When executing data deletion to DB, data itself is not deleted form DB file. (Only deletion mark is added.)
- When updating data, data before updating has not been deleted, and still remains there in DB file.

So, the information which "must have" been deleted may still remain in DB file. Even in this case, take counter-measures according this Guidebook, and when Android security function is enabled, data/file may not be directly accessed by the third party including other applications. However, considering the case that files are picked out by passing through Android's protection system like root privilege is taken, in case the data which gives huge influence on business is stored, data protection which doesn't depend on Android protection system, should be considered.

As above reasons, the important data which is necessary to be protected even when device's root privilege is taken, should not be stored in DB of SQLite, as it is. In case need to store the important data, it's necessary to implement counter-measures, or encrypt overall DB.

When encryption is necessary, there are so many issues that are beyond the range of this Guidebook, like handling the key which is used for encryption or code obfuscation, so as of now it's recommended to consult the specialist when developing an application which handles data that has huge business impact.

Please refer to "4.5.3.6 [Reference] Encrypt SQLite Database (SQLCipher for Android," library which encrypts database is introduced here.

4.5.3.6. [Reference] Encrypt SQLite Database (SQLCipher for Android)

SQLCipher is the SQLite extension that provides encryption of transparent 256 bit AES for database. It's open sourced (BSD license), and maintained/managed by Zetetic LLC. In a world of mobile, SQLCipher is widely used in Nokia/QT, Apple's iOS.

SQLCipher for Android project is aiming to support the standard integrated encryption for SQLite database in Android environment. By creating the standard SQLite's API for SQLCipher, developers can use the encrypted database with the same coding as per usual.

Reference: https://guardianproject.info/code/sqlcipher/

How to Use

Application developers can use SQLCipher by following 3 steps below.

1. Locate sqlcipher.jar, libdatabase_sqlcipher.so, libsqlcipher_android.so and libstlport_shared.so in application's lib directory.
2. Regarding all source files, change all android.database.sqlite.* which is specified by import, to info.guardianproject.database.sqlite.*. In addition, android.database.Cursor can be used as it is.
3. Initialize database in onCreate(), and set password when opening database.

Easy code example

```
SQLiteDatabase.loadLibs(this);                 // First, Initialize library by using context.
SQLiteOpenHelper.getWritableDatabase(passwoed):  // Parameter is password(Suppose that it's string type and It's got
 in a secure way.)
```

SQLCipher for Android was version 1.1.0 at the time of writing, and now version 2.0.0 is under developing, and RC4 is disclosed now. In terms of the past usage in Android and stability of API, it's necessary to be verified later, but currently still there's a room to consider as encryption solution of SQLite, which can be used in Android.

Library Structure

The following files which are included as SDK, are necessary, to use SQLCipher.

- assets/icudt46l.zip 2,252KB
  It's necessary when icudt46l.dat doesn't exist below /system/usr/icu/ and its earlier version. When icudt46l.dat cannot be found, this zip is unzipped and to be used.

- libs/armeabi/libdatabase_sqlcipher.so 44KB
- libs/armeabi/libsqlcipher_android.so 1,117KB
- libs/armeabi/libstlport_shared.so 555KB
  Native Library. It's read out when SQLCipher's initial load (When calling SQLiteDatabase#loadLibs()).

- libs/commons-codec.jar 46KB
- libs/guava-r09.jar 1,116KB
- libs/sqlcipher.jar 102KB
  Java library which calls Native library. sqlcipher.jar is main. Others are referred from sqlcipher.jar.

Toal: about 5.12MB
However, when icudt46l.zip is unzipped, it amounts to around 7MB.

4.6. Handling Files

According to Android security designing idea, files are used only for making information persistence and temporary save (cache), and it should be private in principle. Exchanging information between applications should not be direct access to files, but it should be exchanged by inter-application linkage system, like Content Provider or Service. By using this, inter-application access control can be achieved.

Since enough access control cannot be performed on external memory device like SD card etc, so it should be limited to use only when it's necessary by all means in terms of function, like when handling huge size files or transferring information to another location (PC etc.). Basically, files that include sensitive information should not be saved in external memory device. In case sensitive information needs to be saved in a file of external device at any rate, counter-measures like encryption are necessary, but it's not referred here.

4.6.1. Sample Code

As mentioned above, files should be private in principle. However, sometimes files should be read out/written by other applications directly for some reasons. File types which are categorized from the security point of view and comparison are shown in Table 4.6-1. These are categorized into 4 types of files based on the file storage location or access permission to other application. Sample code for each file category is shown below and explanation for each of them are also added there.

Table 4.6-1 File category and comparison from security point of view

File category	Access permission to other application	Storage location	Overview
Private file	NA	In application directory	● Can read and write only in an application ● Sensitive information can be handled. ● File should be this type in principle.
Read out public file	Read out	In application directory	● Other applications and users can read. ● Information that can be disclosed to outside of application is handled.
Read write public file	Read out Write in	In application directory	● Other applications and users can read and write. ● It should not be used from both security and application designing points of view.
External memory device (Read write public)	Read out Write in	External memory device like SD card	● No access control ● Other applications and users can always read/write/delete files. ● Usage should be minimum requirement. ● Comparatively huge size of files can be handled.

4.6.1.1. Using Private Files

This is the case to use files that can be read /written only in the same application, and it is a very safe way to use files. In principle, whether the information stored in the file is public or not, keep files private as much as possible, and when exchanging the necessary information with other applications, it should be done using another Android system (Content Provider, Service.)

Points:

1. Files must be created in application directory.
2. The access privilege of file must be set private mode in order not to be used by other applications.
3. Sensitive information can be stored.
4. Regarding the information to be stored in files, handle file data carefully and securely.

PrivateFileActivity.java

```
package org.jssec.android.file.privatefile;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.privatefile.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateFileActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "private_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file);

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set private mode in order not to be used by other
applications.
```



```
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);

            // *** POINT 3 *** Sensitive information can be stored.
            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Not sensotive information (File Activity)¥n").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }

        finish();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * Delete file process
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(this.getFilesDir() + "/" + FILE_NAME);
```

```
        file.delete();

        mFileView.setText(R.string.file_view);
    }
}
```

PrivateUserActivity.java

```
package org.jssec.android.file.privatefile;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.privatefile.R;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateUserActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "private_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClass(this, PrivateFileActivity.class);

        startActivity(intent);
    }

    /**
     *  Call file Activity process
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     *  Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
```



```
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * Rewrite file process
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set private mode in order not to be used by other
applications.
            fos = openFileOutput(FILE_NAME, MODE_APPEND);

            // *** POINT 3 *** Sensitive information can be stored.
            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Sensitive information (User Activity)¥n").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }

        callFileActivity();
    }

}
```


4.6.1.2. Using Public Read Only Files

This is the case to use files to disclose the contents to unspecified large number of applications. If you implement by following the below points, it's also comparatively safe file usage method.

Points:

1. Files must be created in application directory.
2. The access privilege of file must be set to read only to other applications.
3. Sensitive information must not be stored.
4. Regarding the information to be stored in files, handle file data carefully and securely.

PublicFileActivity.java

```
package org.jssec.android.file.publicfile.readonly;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.publicfile.readonly.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicFileActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "public_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file);

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set to read only to other applications.
            fos = openFileOutput(FILE_NAME, MODE_WORLD_READABLE);

            // *** POINT 3 *** Sensitive information must not be stored.
            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
```

```
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Not sensitive information (Public File Activity)¥n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
        }

        finish();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * Delete file process
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(this.getFilesDir() + "/" + FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
```



```
}
```

PublicUserActivity.java

```
package org.jssec.android.file.publicuser.readonly;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.publicuser.readonly.R;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicUserActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_PACKAGE = "org.jssec.android.file.publicfile.readonly";
    private static final String TARGET_CLASS = "org.jssec.android.file.publicfile.readonly.PublicFileActivity";

    private static final String FILE_NAME = "public_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity does not exist)");
        }
    }

    /**
     * Call file Activity process
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }
```

```
    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 4 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * Rewrite file process
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        boolean exception = false;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            // Fail to write in. FileNotFoundException occurs.
            fos = new FileOutputStream(file, true);

            fos.write(new String("Not sensitive information (Public User Activity)¥n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(e.getMessage());
            exception = true;
        } catch (IOException e) {
            mFileView.setText(e.getMessage());
            exception = true;
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    exception = true;
```



```
                }
            }
        }

        if (exception == false)
            callFileActivity();
    }

    private final String getFilesPath(String filename) {
        String path = "";

        try {
            Context ctx = createPackageContext(TARGET_PACKAGE,
                    Context.CONTEXT_RESTRICTED);
            File file = new File(ctx.getFilesDir(), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
        }
        return path;
    }
}
```

4.6.1.3. Using Public Read/Write Files

This is the usage of the file which permits read-write access to unspecified large number of application.

Unspecified large number of application can read and write, means that needless to say. Malware can also read and write, so the credibility and safety of data will be never guaranteed. In addition, even in case of not malicious intention, data format in file or timing to write in cannot be controlled. So this type of file is almost not practical in terms of functionality.

As above, it's impossible to use read-write files safely from both security and application designing points of view, so using read-write files should be avoided.

Point:

1. Must not create files that be allowed to read/write access from other applications.

4.6.1.4. Using Eternal Memory (Read Write Public) Files

This is the case when storing files in an external memory like SD card. It's supposed to be used when storing comparatively huge information (placing file which was downloaded from Web), or when bring out the information to outside (backup etc.)

"External memory file (Read Write public)" has the equal characteristics with "Read Write public file" to unspecified large number of applications. In addition, it has the equal characteristics with "Read Write public file" to applications which declares to use android.permission.WRITE_EXTERNAL_STORAGE Permission. So, the usage of "External memory file (Read Write public) file" should be minimized as less as possible.

A Backup file is most probably created in an external memory device as Android application's customary practice. However, as mentioned as above, files in an external memory have the risk that is tampered/ deleted by other applications including malware. Hence, in applications which output backup, some contrivances to minimize risks in terms of application spec or designing like displaying a caution "Copy Backup files to the safety location like PC etc, a.s.a.p.", are necessary.

Points:

1. Sensitive information must not be stored.
2. Files must be stored in the unique directory per application.
3. Regarding the information to be stored in files, handle file data carefully and securely.
4. Writing file by the requesting application should be prohibited as the specification.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.file.externalfile"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />
    <!-- declare android.permission.WRITE_EXTERNAL_STORAGE permission to write to the external strage -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ExternalFileActivity"
            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>
</manifest>
```

ExternalFileActivity.java

```
package org.jssec.android.file.externalfile;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.file.externalfile.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExternalFileActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_TYPE = "external";

    private static final String FILE_NAME = "external_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file);

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Sensitive information must not be stored.
            // *** POINT 2 *** Files must be stored in the unique directory per application.
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fos = new FileOutputStream(file, false);

            // *** POINT 3 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Non-Sensitive Information(ExternalFileActivity)¥n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                }
            }
```



```
        }

        finish();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 3 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * Delete file process
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
}
```

ExternalFileUser.java

```
package org.jssec.android.file.externaluser;

import java.io.File;
import java.io.FileInputStream;
```

```
import java.io.FileNotFoundException;
import java.io.IOException;

import org.jssec.android.file.externaluser.R;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExternalUserActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_PACKAGE = "org.jssec.android.file.externalfile";
    private static final String TARGET_CLASS = "org.jssec.android.file.externalfile.ExternalFileActivity";
    private static final String TARGET_TYPE = "external";

    private static final String FILE_NAME = "external_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity does not exist)");
        }
    }

    /**
     * Call file Activity process
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
```



```
        try {
            File file = new File(getFilesPath(FILE_NAME));
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 3 *** Regarding the information to be stored in files, handle file data carefully and securel
y.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                }
            }
        }
    }

    /**
     * Rewrite file process
     *
     * @param view
     */
    public void onWriteFileClick(View view) {

        // *** POINT 4 *** Writing file by the requesting application should be prohibited as the specification.
        // Application should be designed supposing malicious application may overwrite or delete file.

        final AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(
                this);
        alertDialogBuilder.setTitle("POINT 4");
        alertDialogBuilder.setMessage("Do not write in calling appllication.");
        alertDialogBuilder.setPositiveButton("OK",
                new DialogInterface.OnClickListener() {

                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        callFileActivity();
                    }
                });

        alertDialogBuilder.create().show();

    }

    private final String getFilesPath(String filename) {
        String path = "";

        try {
            Context ctx = createPackageContext(TARGET_PACKAGE,
                    Context.CONTEXT_IGNORE_SECURITY);
```

```
            File file = new File(ctx.getExternalFilesDir(TARGET_TYPE), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
        }
        return path;
    }
}
```


4.6.2. Rule Book

Handling files follow the rules below.

1.	File Must Be Created as a Private File in Principle	(Required)
2.	Must Not Create Files that Be Allowed to Read/Write Access from Other Applications	(Required)
3.	Using Files Stored in External Device (e.g. SD Card) Should Be Requisite Minimum	(Required)
4.	Application Should Be Designed Considering the Scope of File	(Required)

4.6.2.1. File Must Be Created as a Private File in Principle (Required)

As mentioned in "4.6 Handling Files" and "4.6.1.3 Using Public Read/Write File," regardless of the contents of the information to be stored, files should be set private, in principle. From Android security designing point of view, exchanging information and its access control should be done in Android system like Content Provider and Service, etc, and in case there's a reason that is impossible, it should be considered to be substituted by file access permission as alternative method.

Please refer to sample code of each file type and following rule items.

4.6.2.2. Must Not Create Files that Be Allowed to Read/Write Access from Other Applications (Required)

As mentioned in "4.6.1.3 Using Public Read/Write File," when permitting other applications to read/write files, information stored in files cannot be controlled. So, sharing information by using read/write public files should not be considered from both security and function/designing points of view.

4.6.2.3. Using Files Stored in External Device (e.g. SD Card) Should Be Requisite Minimum(Required)

As mentioned in "4.6.1.4 Using Eternal Memory (Read Write Public) File," storing files in external memory device like SD card, leads to holding the potential problems from security and functional points of view. On the other hand, SD card can handle files which have longer scope, compared with application directory, and this is the only one storage that can be always used to bring out the data to outside of application. So, there may be many cases that cannot help using it, depends on application's spec.

When storing files in external memory device, considering unspecified large number of applications and users can read/write/delete files, so it's necessary that application is designed considering the points as per below as well as the points mentioned in sample code. In addition, regarding encryption technology like encryption and electrical signature, it's planned that articles are published in future edition of this Guidebook.

- Sensitive information should not be saved in a file of external memory device, in principle.
- In case sensitive information is saved in a file of external memory device, it should be encrypted.

- In case saving in a file of external memory device information that will be trouble if it's tampered by other application or users, it should be saved with electrical signature.
- When reading in files in external memory device, use data after verifying the safety of data to read in.
- Application should be designed supposing that files in external memory device can be always deleted.

Please refer to "4.6.2.4 Application Should Be Designed Considering the Scope of File (Required)."

4.6.2.4. Application Should Be Designed Considering the Scope of File (Required)

Data saved in application directory is deleted by the following user operations. It's consistent with the application's scope, and it's distinctive that it's shorter than the scope of application.

- Uninstalling application.
- Delete data and cache of each application (Setting > Apps > select target application.)

Files that were saved in external memory device like SD card, it's distinctive that the scope of the file is longer than the scope of the application. In addition, the following situations are also necessary to be considered.

- File deletion by user
- Pick off/replace/unmount SD card
- File deletion by Malware

As mentioned above, since scope of files are different depends on the file saving location, not only from the viewpoint to protect sensitive information, but also form view point to achieve the right behavior as application, it's necessary to select the file save location.

4.6.3. Advanced Topics

4.6.3.1. File Sharing Through File Descriptor

There is a method to share files through file descriptor, not letting other applications access to public files. This method can be used in Content Provider and in Service. Opponent application can read/write files through file descriptors which are got by opening private files in Content Provider or in Service.

Comparison between the file sharing method of direct access by other applications and the file sharing method via file descriptor, is as per below Table 4.6-2. Variation of access permission and range of applications that are permitted to access, can be considered as merits. Especially, from security point of view, this is a great merit that, applicaions that are permitted to accesss can be controlled in detail.

Table 4.6-2 Comparison of inter-application file sharing method

File sharing method	Variation or access permission setting	Range of applications that are permitted to access
File sharing that permits other applications to access files directly	Read in Write in Read in + Write in	Give all applications access permissions equally
File sharing through file descriptor	Read in Write in Only add Read in + Write in Read in + Only add	Can control whether to give access permission or not, to application which try to access individually and temporarily, to Content Provider or Service

This is common in both of above file sharing methods, when giving write permission for files to other applications, integrity of file contents are difficult to be guaranteed. When several applications write in in parallel, there's a risk that data structure of file contents are destroyed, and application doesn't work normally. So, in sharing files with other applications, giving only read only permission is preferable.

Herein below an implementation example of file sharing by Content Provider and its sample code, are published.

Point

1. The source application is In house application, so sensitive information can be saved.
2. Even if it's a result from In house only Content Provider application, verify the safety of the result data.

InhouseProvider.java

```
package org.jssec.android.file.inhouseprovider;

import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.ParcelFileDescriptor;

public class InhouseProvider extends ContentProvider {

    private static final String FILENAME = "sensitive.txt";

    // In-house signature permission
    private static final String MY_PERMISSION = "org.jssec.android.file.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;

    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of  debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of  keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public boolean onCreate() {
        File dir = getContext().getFilesDir();
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(new File(dir, FILENAME));
            // *** POINT 1 *** The source application is In house application, so sensitive information can be saved.
            fos.write(new String("Sensitive information").getBytes());

        } catch (IOException e) {
        } finally {
            try {
                fos.close();
            } catch (IOException e) {
            }
        }

        return true;
    }

    @Override
    public ParcelFileDescriptor openFile(Uri uri, String mode)
            throws FileNotFoundException {
```



```
        // Verify that in-house-defined signature permission is defined by in-house application.
        if (!SigPerm
                .test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException(
                    "In-house-defined signature permission is not defined by in-house application.");
        }

        File dir = getContext().getFilesDir();
        File file = new File(dir, FILENAME);

        // Always return read-only, since this is sample
        int modeBits = ParcelFileDescriptor.MODE_READ_ONLY;
        return ParcelFileDescriptor.open(file, modeBits);
    }

    @Override
    public String getType(Uri uri) {
        return "";
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        return 0;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
}
```

InhouseUserActivity.java

```
package org.jssec.android.file.inhouseprovideruser;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.pm.PackageManager;
```

```
import android.content.pm.ProviderInfo;
import android.net.Uri;
import android.os.Bundle;
import android.os.ParcelFileDescriptor;
import android.view.View;
import android.widget.TextView;

public class InhouseUserActivity extends Activity {

    // Content Provider information of destination (requested provider)
    private static final String AUTHORITY = "org.jssec.android.file.inhouseprovider";

    // In-house signature permission
    private static final String MY_PERMISSION = "org.jssec.android.file.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;

    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of  debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of  keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Get package name of destination (requested) content provider.
    private static String providerPkgname(Context context, String authority) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
        ProviderInfo pi = pm.resolveContentProvider(authority, 0);
        if (pi != null)
            pkgname = pi.packageName;
        return pkgname;
    }

    public void onReadFileClick(View view) {

        logLine("[ReadFile]");

        // Verify that in-house-defined signature permission is defined by in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  In-house-defined signature permission is not defined by in-house application.");
            return;
        }

        // Verify that the certificate of destination (requested) content provider application is in-house certificat
e.
        String pkgname = providerPkgname(this, AUTHORITY);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  Destination (Requested) Content Provider is not in-house application.");
            return;
        }

        // Only the information which can be disclosed to in-house only content provider application, can be included
```



```
in a request.
        ParcelFileDescriptor pfd = null;
        try {
            pfd = getContentResolver().openFileDescriptor(
                    Uri.parse("content://" + AUTHORITY), "r");
        } catch (FileNotFoundException e) {

        }

        if (pfd != null) {
            FileInputStream fis = new FileInputStream(pfd.getFileDescriptor());

            if (fis != null) {
                try {
                    byte[] buf = new byte[(int) fis.getChannel().size()];
                    fis.read(buf);
                    // *** POINT 2 *** Handle received result data carefully and securely,
                    // even though the data came from in-house applications.
                    // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely
."
                    logLine(new String(buf));
                } catch (IOException e) {
                } finally {
                    try {
                        fis.close();
                    } catch (IOException e) {
                    }
                }
            }
            try {
                pfd.close();
            } catch (IOException e) {
            }

        } else {
            logLine("  null file descriptor");
        }
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView) findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

4.6.3.2. Access Permission Setting for the Directory

Herein above, security considerations are explained, focusing on files. It's also necessary to consider

the security for directory which is a file container. Herein below, security considerations of access permission setting for directory are explained.

In Android, there are some methods to get/create subdirectory in application directory. The major ones are as per below Table 4.6-3.

Table 4.6-3 Methods to get/create subdirectory in application directory

	Specify access permission to other application	Deletion by user
Context#getFilesDir()	Impossible (Only execution permission)	"Setting" > "Apps" > select target application > "Clear data"
Context#getCacheDir()	Impossible (Only execution permission)	"Setting" > "Apps" > select target application > "Clear cache" It can be deleted by "Clear data," too
Context#getDir(String name, int MODE)	Following can be set to mode MODE_PRIVATE MODE_WORLD_READABLE MODE_WORLD_WRITABLE	"Setting" > "Apps" > select target application > "Clear data"

Here especially what needs to pay attention is access permission setting by Context#getDir(). As explained in file creation, basically directory also should be set private from the security designing point of view. When sharing information depends on access permission setting, there may be an unexpected side effect, so other methods should be taken as information sharing.

MODE_WORLD_READABLE

This is a flag to give all applications read-only permission to directory. So all application can get file list and individual file attribute information in the directory. Since secret files cannot be placed under this directory, it's necessary to pay enough attention when using this flag.

MODE_WORLD_WRITABLE

This flag gives other applications write permission to directory. All applications can create/move[8]/rename/delete files in the directory. These operations has no relation with access permission setting (Read/Write/Execute) of file itself, so it's necessary to pay attention that operations can be done only with write permission to directory. Normally this flag should not be used since file may be deleted/replaced freely by other applications.

Regarding Table 4.6-3 "Deletion by User," refer to "4.6.2.4 Application Should Be Designed Considering the Scope of File (Required)."

[8] Files cannot be moved over mount point (e.g. from internal storage to external storage). Therefore, moving the protected files from internal storage to external storage cannot be happened.

4.6.3.3. Access Permission Setting for Shared Preference and Database File

Shared Preference and database also consist of files. Regarding access permission setting, what are explained for files are applied here. i.e., both Shared Preference and database, should be created as private files same like files, and sharing contents should be achieved by the Android's inter-application linkage system.

Herein below, the usage example of Shared Preference is shown. Shared Preference is crated as private file by MODE_PRIVATE.

Example of setting access restriction to Shared Preference file.

```
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;

       Ommision of a passage

       // Get Shared Preference . (If there's no Shared Preference, it's to be created.)
       // Point:Basically, specify MODE_PRIVATE mode.
       SharedPreferences preference = getSharedPreferences(
              PREFERENCE_FILE_NAME, MODE_PRIVATE);

       // Example of writing preference which value is charcter string
       Editor editor = preference.edit();
       editor.putString("prep_key", "prep_value");// key:"prep_key", value:"prep_value"
       editor.commit();
```

Please refer to "4.5 Using SQLite" for database.

4.7. Using Browsable Intent

Android application can be designed to launch from browser corresponding with a webpage link. This functionality is called 'Browsable Intent.' By specifying URI scheme in Manifest file, an application responds the transition to the link (user tap etc) which has its URI scheme, and the application is launched with the link as a parameter.

In addition, the method to launch the corresponding application from browser by using URI scheme is supported not only in Android but also in iOS and other platforms, and this is generally used for the linkage between Web application and external application, etc. For example, following URI scheme is defined in Twitter application or Facebook application, and the corresponding applications are launched from the browser both in Android and in iOS.

Table 4.7-1

URI scheme	Corresponding application
fb://	Facebook
twitter://	Twitter

It seems very convenient function considering the linkage and convenience, but there are some risks that this function is abused by a malicious third party. What can be supposed are as follows, they abuse application functions by preparing a malicious Web site with a link in which URL has incorrect parameter, or they get information which is included in URL by cheating a smartphone owner into installing the Malware which responds the same URI scheme.

There are some points to be aware when using 'Browsable Intent' against these risks.

4.7.1. Sample Code

Sample codes of an application which uses 'Browsable Intent' are shown below.

Points:

1. (Webpage side) Sensitive information must not be included.
2. Handle the URL parameter carefully and securely.

Starter.html

```
<html>
    <body>
        <!-- *** POINT 1 *** Sensitive information must not be included -->
        <!-- Character strings to be passed as URL parameter, should be UTF-8 and URI encoded. -->
        <a href="secure://jssec?user=user_id"> Login </a>
    </body>
</html>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.browsableintent"
```



```
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".BrowsableIntentActivity"
            android:label="@string/title_activity_browsable_intent" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                // Accept implicit Intent
                <category android:name="android.intent.category.DEFAULT" />
                // Accept Browsable intent
                <category android:name="android.intent.category.BROWSABLE" />
                // Accept URI 'secure://jssec'
                <data android:scheme="secure" android:host="jssec"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

BrowsableIntentActivity.java

```
package org.jssec.android.browsableintent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class BrowsableIntentActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_browsable_intent);

        Intent intent = getIntent();
        Uri uri = intent.getData();
        if (uri != null) {
            // Get UserID which is passed by URI parameter
            // *** POINT 2 *** Handle the URL parameter carefully and securely.
            // Omitted, since this is a sample. Please refer to "3.2 Handling Input Data Carefully and Securely."
            String userID = "User ID = " +  uri.getQueryParameter("user");
            TextView tv = (TextView)findViewById(R.id.text_userid);
            tv.setText(userID);
        }
    }
```

```
}
```


4.7.2. Rule Book

Follow rules listed below when using "Browsable Intent".

1.	(Webpage side) Sensitive Information Must Not Be Included in Parameter of Corresponding Link (Required)
2.	Handle the URL Parameter Carefully and Securely (Required)

4.7.2.1. (Webpage side) Sensitive Information Must Not Be Included in Parameter of Corresponding Link (Required)

When tapping the link in browser, an intent which has a URL value in its data (It can be retrieve by Intent#getData) is issued, and an application which has a corresponding Intent Filter is launched from Android system.

At this moment, when there are several applications which Intent Filter is set to receive the same URI scheme, application selection dialogue is shown in the same way as normal launch by implicit Intent, and an application which user selected is launched. In case that a Malware is listed in the selection of application selection dialogue, there is a risk that user may launch the Malware by mistake and parameters in URL are sent to Malware.

As per above, it is necessary to avoid from include sensitive information directly in URL parameter as it is for creating general Webpage link since all parameters which are included in Webpage link URL can be given to Malware.

Example that User ID and Password are included in URL

```
insecure://sample/login?userID=12345&password=abcdef
```

In addition, there is a risk that user may launch a Malware and input password to it when it is defined in specs that password input is executed in an application after being launched by 'Browsable Intent', even if the URL parameter includes only non-sensitive information like User ID. So it should be considered that specs like a whole Login process is completed within application side. It must be kept in mind when designing an application and a service that launching application by 'Browsable Intent' is equivalent to launching by implicit Intent and there is no guarantee that a valid application is launched.

4.7.2.2. Handle the URL Parameter Carefully and Securely (Required)

URL parameters which are sent to an application are not always from a legitimate Web page, since a link which is matched with URI scheme can be made by not only developers but anyone. In addition, there is no method to verify whether the URL parameter is sent from a valid Web page or not.

So it is necessary to verify safety of a URL parameter before using it, e.g. check if an unexpected value is included or not.

4.8. Outputting Log to LogCat

There's a logging mechanism called LogCat in Android, and not only system log information but also application log information are also output to LogCat. Log information in LogCat can be read out from other application in the same device[9], so the application which outputs sensitive information to Logcat, is considered that it has the vulnerability of the information leakage. The sensitive information should not be output to LogCat.

From a security point of view, in release version application, it's preferable that any log should not be output. However, even in case of release version application, log is output for some reasons in some cases. In this chapter, we introduce some ways to output messages to LogCat in a safe manner even in a release version application. Along with this explanation, please refer to "4.8.3.1 Two Ways of Thinking for the Log Outputting in Release version application."

4.8.1. Sample Code

Herein after, the method to control the Log output to LogCat by ProGuard in release version application. ProGuard is one of the optimization tools which automatically delete the unnecessary code like unused methods, etc.

There are five types of log output methods, Log.e(), Log.w(), Log.i(), Log.d(), Log.v(), in android.util.Log class. Regarding log information, intentionally output log information (hereinafter referred to as the Operation log information) should be distinguished from logging which is inappropriate for a release version application such as debug log (hereinafter referred to as the Development log information). It's recommended to use Log.e()/w()/i() for outputting operation log information, and to use Log.d()/v() for outputting development log. Refer to "4.8.3.2 Selection Standards of Log Level and Log Output Method" for the details of proper usage of five types of log output methods, in addition, also refer to "4.8.3.3 DEBUG Log and VERBOSE Log Are Not Always Deleted Automatically."

Here's an example of how to use LogCat in a safe manner. This example includes Log.d() and Log.v() for outputting debug log. If the application is for release, these two methods would be deleted automatically. In this sample code, ProGuard is used to automatically delete code blocks where Log.d()/v() is called.

[9] The log information output to LogCat can be read by applications that declare using READ_LOGS permission. However, in Android 4.1 and later, log information that is output by other application cannot be read. But smartphone user can read every log information ouput to logcat through ADB.

Points:

1. Sensitive information must not be output by Log.e()/w()/i(), System.out/err.
2. Sensitive information should be output by Log.d()/v() in case of need.
3. The return value of Log.d()/v()should not be used (with the purpose of substitution or comparison).
4. When you build an application for release, you should bring the mechanism that automatically deletes inappropriate logging method like Log.d() or Log.v() in your code..
5. An APK file for the (public) release must be created in release build configurations.

ProGuardActivity.java

```
package org.jssec.android.log.proguard;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ProGuardActivity extends Activity {

    final static String LOG_TAG = "ProGuardActivity";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_proguard);

        // *** POINT 1 *** Sensitive information must not be output by Log.e()/w()/i(), System.out/err.
        Log.e(LOG_TAG, "Not sensitive information (ERROR)");
        Log.w(LOG_TAG, "Not sensitive information (WARN)");
        Log.i(LOG_TAG, "Not sensitive information (INFO)");

        // *** POINT 2 *** Sensitive information should be output by Log.d()/v() in case of need.
        // *** POINT 3 *** The return value of Log.d()/v()should not be used (with the purpose of substitution or comp
arison).
        Log.d(LOG_TAG, "sensitive information (DEBUG)");
        Log.v(LOG_TAG, "sensitive information (VERBOSE)");
    }
}
```

A part of project.properties

```
# ProGuard
proguard.config=proguard-project.txt
```

proguard-project.txt

```
# prevent from changing class name and method name etc.
-dontobfuscate

# *** POINT 4 *** In release build, the build configurations in which Log.d()/v() are deleted automatically should be
 constructed.
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

*** Point 5 *** An APK file for the (public) release must be created in release build configurations.

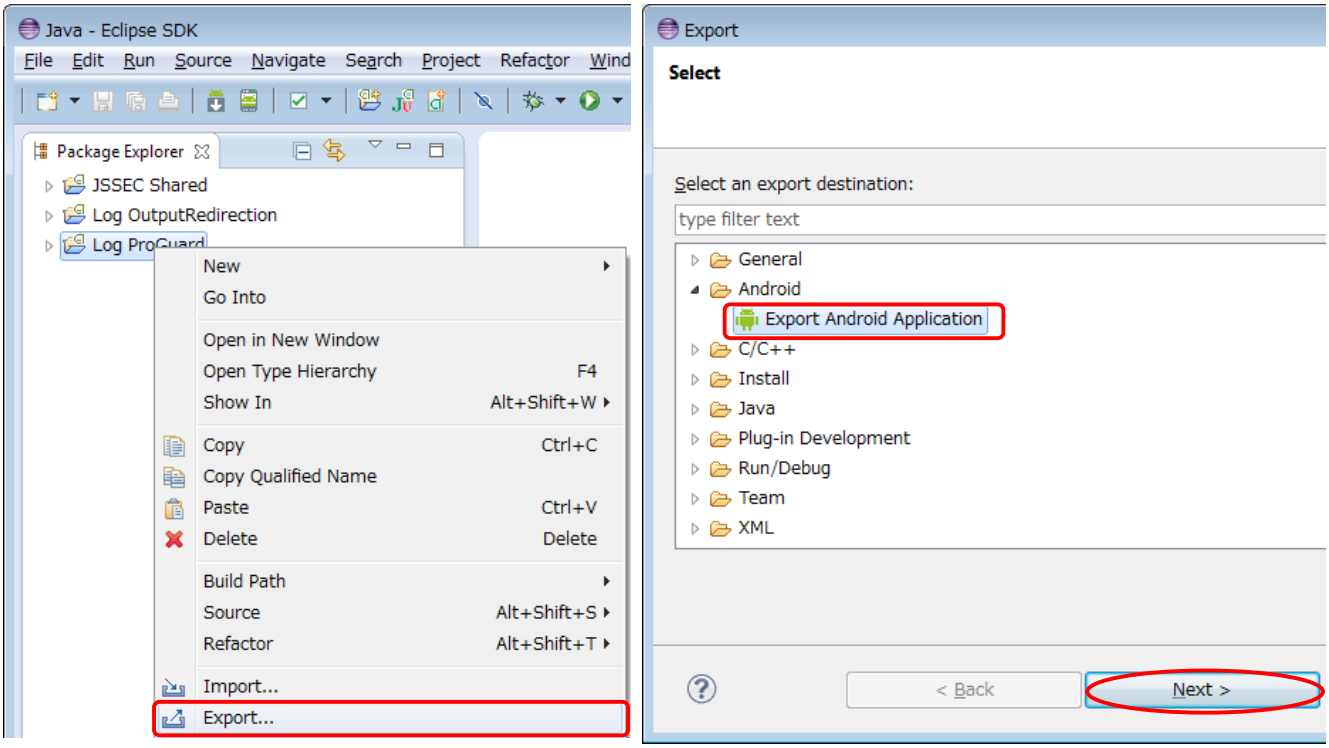
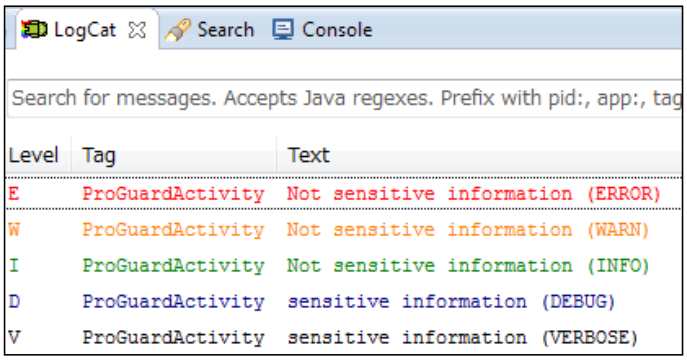


Figure 4.8-1 How to create release version application in Eclipse (By Export)

The difference of LogCat output between development version application (debug build) and release version application (release build) are shown in below Figure 4.8-2.

Development version application (Debug build)



Release version application (Release build)

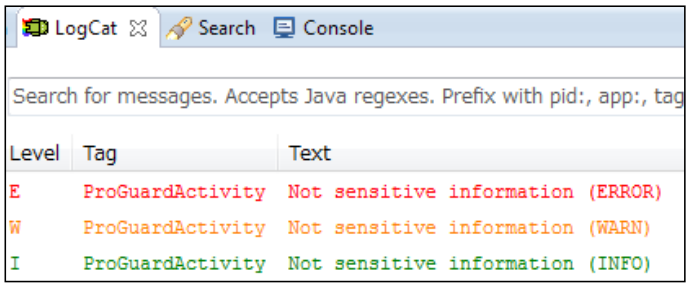


Figure 4.8-2 Difference of LogCat output
between development version application and release version application

4.8.2. Rule Book

When you output log messages, follow the rules below.

1.	Sensitive Information Must Not Be Included in Operation Log Information (Required)
2.	Construct the Build System to Auto-delete Codes which Output Development Log Information When Build for the Release (Recommended)
3.	Use Log.d()/v() Method When Outputting Throwable Object (Recommended)
4.	Use Only Methods of the android.util.Log Class for the Log Output (Recommended)

4.8.2.1. Sensitive Information Must Not Be Included in Operation Log Information (Required)

Log which was output to LogCat can be read out from other applications, so sensitive information like user's login information should not be output by release version application. It's necessary not to write code which outputs sensitive information to log during development, or it's necessary to delete all of such codes before release.

To follow this rule, first, not to include sensitive information in operation log information. In addition, it's recommended to construct the system to delete code which outputs sensitive information when build for release. Please refer to "4.8.2.2 Construct the Build System to Auto-delete Codes which Output Development Log Information When Build for the Release (Recommended)."

4.8.2.2. Construct the Build System to Auto-delete Codes which Output Development Log Information When Build for the Release (Recommended)

When application development, sometimes it's preferable if sensitive information is output to log for checking the process contents and for debugging, for example the interim operation result in the process of complicated logic, information of program's internal state, communication data structure of communication protocol. It doesn't matter to output the sensitive information as debug log during developing, in this case, the corresponding log output code should be deleted before release, as mentioned in "4.8.2.1 Sensitive Information Must Not Be Included in Operation Log Information (Required)."

To delete surely the code which outputs development log information when release builds, the system which executes code deletion automatically by using some tools, should be constructed. ProGuard, which was described in "4.8.1 Sample Code," can work for this method. As described below, there are some noteworthy points on deleting code by ProGuard. Here it's supposed to apply the system to applications which output development log information by either of Log.d()/v(), based on "4.8.3.2 Selection Standards of Log Level and Log Output Method."

ProGuard deletes unnecessary code like unused methods, automatically. By specifying Log.d()/v()as parameter of -assumenosideeffects option, call for Log.d(),Log.v() are granted as unnecessary code , and those are to be deleted.

By specifying -assumenosideeffects to Log.d()/v(), make it auto-deletion target

```
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

In case using this auto deletion system, pay attention that Log.v()/d() code is not deleted when using returned value of Log.v(), Log.d(), so returned value of Log.v(), Log.d(), should not be used. For example, Log.v() is not deleted in the next examination code.

Examination code which Log.v() that is specifeied to be deleted is not deketed

```
int i = android.util.Log.v("tag", "message");
System.out.println(String.format("Log.v() returned %d. ", i));  //Use the returned value of Log.v() for examination
```

If you'd like to reuse source code, you should keep the consistency of the project environment including ProGuard settings. For example, source code that presupposes Log.d() and Log.v() are deleted automatically by above ProGuard setting. If using this source code in another project which ProGuard is not set, Log.d() and Log.v() are not to be deleted, so there's a risk that the sensitive information may be leaked. When reusing source code, the consistency of project environment including ProGuard setting should be secured.

4.8.2.3. Use Log.d()/v() Method When Outputting Throwable Object (Recommended)

As mentioned in "4.8.1 Sample Code" and "4.8.3.2 Selection Standards of Log Level and Log Output Method," sensitive information should not be output to log through Log.e()/w()/i(). On the other hand, in order that a developer wants to output the details of program abnormality to log, when exception occurs, stack trace is output to LogCat by Log.e(..., Throwable tr)/w(..., Throwable tr)/i(..., Throwable tr), in some cases. However, sensitive information may sometimes be included in the stack trace because it shows detail internal structure of the program. For example, when SQLiteException is output as it is, what type of SQL statement is issued is clarified, so it may give the clue for SQL injection attack. Therefore, it's recommended that use only Log.d()/Log.v() methods, when outputting throwable object.

4.8.2.4. Use Only Methods of the android.util.Log Class for the Log Output (Recommended)

You may output log by System.out/err to verify the application's behavior whether it works as expected or not, during development. Of course, log can be output to LogCat by print()/println() method of System.out/err, but it's strongly recommended to use only methods of android.util.Log class, by the following reasons.

When outputting log, generally, use the most appropriate output method properly based on the urgency of the information, and control the output. For example, categories like serious error, caution, simple application's information notice, etc are to be used. However, in this case, information which needs to be output at the time of release (operation log information) and information which may include the sensitive information (development log information) are output by the same method. So, it may happen that when delete code which outputs sensitive information,

it's in danger that some deletion are dropped by oversight.

Along with this, when using android.util.Log and System.out/err for log output, compared with using only android.util.Log, what needs to be considered will increase, so it's in danger that some mistakes may occur, like some deletion are dropped by oversight.

To decrease risk of above mentioned mistakes occurrence, it's recommended to use only methods of android.util.Log class.

4.8.3. Advanced Topics

4.8.3.1. Two Ways of Thinking for the Log Outputting in Release version application

There are two ways of thinking for log output in release version application. One is any log should never be output, and another is necessary information for later analysis should be output as log. It's favorable that any log should never be output in release version application from the security point of view, but sometimes, log is output even in release version application for various reasons. Each way of thinking is described as per below.

The former is "Any log should never be output," this is because outputting log in release version application is not so much valuable, and there is a risk to leak sensitive information. This comes from there's no method for developers to collect log information of the release version application in Android application operation environment, which is different from many Web application operation environments. Based on this thinking, the logging codes are used only in development phase, and all the logging codes are deleted on building release version application.

The latter is "necessary information should be output as log for the later analysis," as a final option to analyze application bugs in customer support, in case of any questions or doubt to your customer support. Based on this idea, as introduced above, it is necessary to prepare the system that prevent human errors and bring it in your project because if you don't have the system you have to keep in mind to avoid logging the sensitive information in release version application.

For more details about logging method, refer to the following document.

> Code Style Guidebook for Contributors / Log Sparingly
> http://source.android.com/source/code-style.html#log-sparingly

4.8.3.2. Selection Standards of Log Level and Log Output Method

There are five levels of log level (ERROR, WARN, INFO, DEBUG, VERBOSE) are defined in android.util.Log class in Android. You should select the most appropriate method when using the android.util.Log class to output log messages according to Table 4.8-1 which shows the selection standards of logging levels and methods.

Table 4.8-1 Selection standards of log levels and log output method

Log level	Method	Log information to be output	Cautions for application release
ERROR	`Log.e()`	Log information which is output when application is in a fatal state.	Log information as per left may be referred by users, so it could be output both in development version application and in release version application. Therefore, sensitive information should not be output in these levels.
WARN	`Log.w()`	Log information which is output when application faces the unexpected serious situation.	
INFO	`Log.i()`	Other than above, log information which is output to notify any	

		remarkable changes or results in application state.	
DEBUG	`Log.d()`	Program's internal state information which needs to be output temporarily for analyzing the cause of specific bug when developing application.	Log information as per left is only for application developers. Therefore, this type of information should not be output in case of release version application.
VERBOSE	`Log.v()`	Log information which is not applied to any of above. Log information which application developer outputs for many purposes, is applied this. For example, in case of outputting server communication data to dump.	

For more details about logging method, refer to the following document.

> Code Style Guidebook for Contributors / Log Sparingly
> http://source.android.com/source/code-style.html#log-sparingly

4.8.3.3. DEBUG Log and VERBOSE Log Are Not Always Deleted Automatically

The following is quoted from the developer reference of android.util.Log class[10].

The order in terms of verbosity, from least to most is ERROR, WARN, INFOFEBUG, VERBOSE. Verbose should never be compiled into an application except during development. Debug logs are compiled in but stripped at runtime. Error, warning and info logs are always kept.

After reading the above texts, some developers might have misunderstood the Log class behavior as per below.

- Log.v() call is not compiled when release build, VERBOSE log is never output.
- Log.v() call is compiled, but DEBUG log is never output when execution.

However, logging methods never behave in above ways, and all messages are output regardless of whether it is compiled with debug mode or release mode. If you read the document carefully, you will be able to realize that the gist of the document is not about the behavior of logging methods but basic policies for logging.

In this chapter, we introduced the sample code to get the expected result as described above by

[10] http://developer.android.com/reference/android/util/Log.html

using ProGuard.

4.8.3.4. BuildConfig.DEBUG Should Be Used in ADT 21 or Later

Recent ADT plugin for Eclipse, the following BuildConfig.java file is automatically generated. The following DEBUG consistent (BuildConfig.DEBUG) is automatically set as "false" in release build, and as "true" in debug build, by ADT plugin.

BuildConfig.java

```
/** Automatically generated file. DO NOT MODIFY */
package com.example.buildconfig;

public final class BuildConfig {
    public final static boolean DEBUG = true;
}
```

By using BuildConfig.DEBUG as per below, log output is restrained when release build.

```
if (BuildConfig.DEBUG) android.util.Log.d(TAG, "Log output information");
```

Unfortunately, there are some bugs in ADT20 and earlier, and DEBUG consistent became true even in release build in some cases. However, these bugs are fixed in ADT 21, and it's necessary to use BuildConfig.DEBUG with ADT 21 or later.

4.8.3.5. Remove Sensitive Information from Assembly

If you build the following code with ProGuard for the purpose of deleting Log.d() method, it is necessary to remember that ProGuard keeps the statement that construct the string for logging message (the first line of the code) even though it remove the statement of calling Log.d() method (the second line of the code).

```
    String debug_info = String.format("%s:%s", "Sensitive information1", "Sensitive information2");
    if (BuildConfig.DEBUG) android.util.Log.d(TAG, debug_info);
```

The following disassembly shows the result of release build of the code above with ProGuard. Actually, there's no Log.d() call process, but you can see that character string consistence definition like "Sensitive information1" and calling process of String#format() method, are not deleted and still remaining there.

```
    const-string v1, "%s:%s"
    const/4 v2, 0x2
    new-array v2, v2, [Ljava/lang/Object;
    const/4 v3, 0x0
    const-string v4, "Sensitive information 1"
    aput-object v4, v2, v3
    const/4 v3, 0x1
    const-string v4, "Sensitive information 2"
    aput-object v4, v2, v3
    invoke-static {v1, v2}, Ljava/lang/String;->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;
```



```
    move-result-object v0
```

Actually, it's not easy to find the particular part that disassembled APK file and assembled log output information as above. However, in some application which handles the very confidential information, this type of process should not be remained in APK file in some cases.

You should implement your application like below to avoid such a consequence of remaining the sensitive information in bytecode. In release build, the following codes are deleted completely by the compiler optimization. However, you have to use BuildConfig.DEBUG with ADT 21 or later (Please refer to "4.8.3.4 BuildConfig.DEBUG Should Be Used in ADT 21").

```
    if (BuildConfig.DEBUG) {
        String debug_info = String.format("%s:%s", " Snsitive information 1", "Sensitive information 2");
        if (BuildConfig.DEBUG) android.util.Log.d(TAG, debug_info);
    }
```

Besides, ProGuard cannot remove the log message of the following code ("result:" + value).

```
    Log.d(TAG, "result:" + value);
```

In this case, you can solve the problem in the following manner.

```
    if (BuildConfig.DEBUG) Log.d(TAG, "result:" + value);
```

4.8.3.6. The Contents of Intent Is Output to LogCat

When using Activity, it's necessary to pay attention, since ActivityManager outputs the content of Intent to LogCat. Refer to "4.1.3.5 Log Output When using Activities."

4.8.3.7. Restrain Log which Is Output to System.out/err

System.out/err method outputs all messages to LogCat. Android could send some messages to System.out/err even if developers did not use these methods in their code, for example, in the following cases, Android sends stack trace to System.err method.

- When using Exception#printStackTrace()
- When it's output to System.err implicitly
  (When the exception is not caught by application, it's given to Exception#printStackTrace() by the system.)

You should handle errors and exceptions appropriately since the stack trace includes the unique information of the application.

We introduce a way of changing default output destination of System.out/err. The following code redirects the output of System.out/err method to nowhere when you build a release version

application. However, you should consider whether this redirection does not cause a malfunction of application or system because the code temporarily overwrites the default behavior of System.out/err method. Furthermore, this redirection is effective only to your application and is worthless to system processes.

OutputRedirectApplication.java

```
package org.jssec.android.log.outputredirection;

import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;

import android.app.Application;

public class OutputRedirectApplication extends Application {

    // PrintStream which is not output anywhere
    private final PrintStream emptyStream = new PrintStream(new OutputStream() {
        public void write(int oneByte) throws IOException {
            // do nothing
        }
    });

    @Override
    public void onCreate() {
        // Redirect System.out/err to PrintStream which doesn't output anywhere, when release build.

        // Save original stream of System.out/err
        PrintStream savedOut = System.out;
        PrintStream savedErr = System.err;

        // Once, redirect System.out/err to PrintStream which doesn't output anywhere
        System.setOut(emptyStream);
        System.setErr(emptyStream);

        // Restore the original stream only when debugging. (In release build, the following 1 line is deleted byProGu
ard.)
        resetStreams(savedOut, savedErr);
    }

    // All of the following methods are deleted byProGuard when release.
    private void resetStreams(PrintStream savedOut, PrintStream savedErr) {
        System.setOut(savedOut);
        System.setErr(savedErr);
    }
}
```

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.log.outputredirection"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="8" />

    <application
```



```
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:name=".OutputRedirectApplication" >
        <activity
            android:name=".LogActivity"
            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>
</manifest>
```

project.properties

```
# ProGuard
proguard.config=proguard-project.txt
```

proguard-project.txt

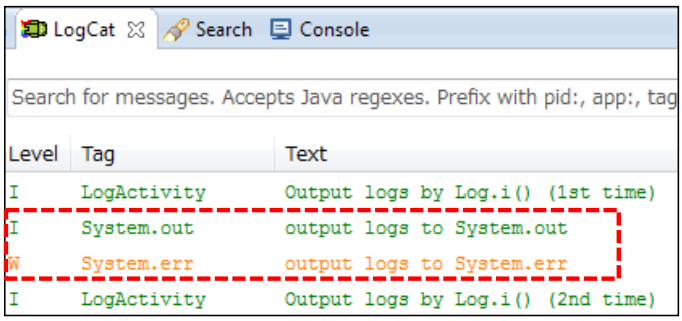
```
# Prevent from changing class name and method name, etc
-dontobfuscate

# In release build, delete call from Log.d()/v() automatically.
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}

# In release build, delete resetStreams() automatically.
-assumenosideeffects class org.jssec.android.log.outputredirection.OutputRedirectApplication {
    private void resetStreams(...);
}
```

The difference of LogCat output between development version application (debug build) and release version application (release build) are shown as per below Figure 4.8-3.

Development version application (Debug build)



Release version application (Release build)

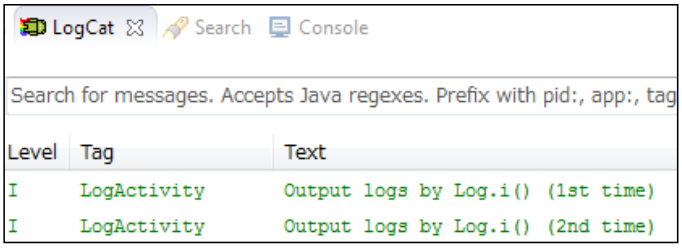


Figure 4.8-3 Difference of System.out/err in LogCat output, between development application and release application

WebView enables your application to integrate HTML/JavaScript content.

4.9.1. Sample Code

We need to take proper action, depending on what we'd like to show through WebView although we can easily show web site and html file by it. And also we need to consider risk from WebView's remarkable function; such as JavaScript-Java object bind.

Especially what we need to pay attention is JavaScript. (Please note that JavaScript is disabled as default. And we can enable it by WebSettings#setJavaScriptEnabled()). With enabling JavaScript, there is potential risk that malicious third party can get device information and operate your device.

The following is principle for application with WebView[11]:

(1) You can enable JavaScript if the application uses contents which are managed in house.
(2) You should NOT enable JavaScript other than the above case.

Figure 4.9-1 shows flow chart to choose sample code according to content characteristic.

[11]Strictly speaking, you can enable JavaScript if we can say the content is safe. If the contents are managed in house, the contents should be guaranteed of security. And the company can secure them. In other words, we need to have business representation's decision to enable JavaScript for other company's contents. The contents which are developed by trusted partner might have security guarantee. But there is still potential risk. Therefore the decision is needed by responsible person.

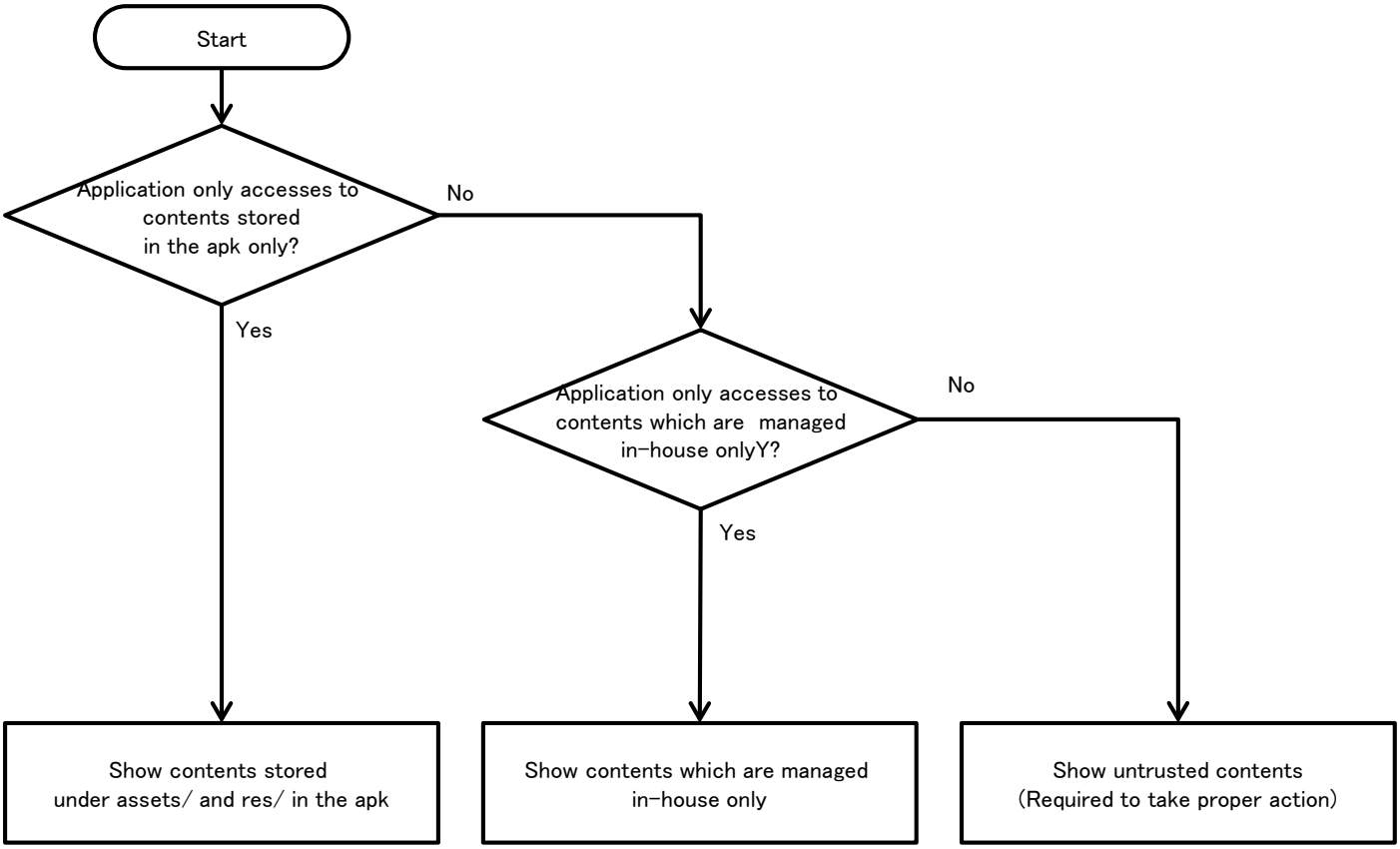


Figure 4.9-1 Flow Figure to select Sample code of WebView

4.9.1.1. Show Only Contents Stored under assets/res Directory in the APK

You can enable JavaScript if your application shows only contents stored under assets/ and res/ directory in apk.

The following sample code shows how to use WebView to show contents stored under assets/ and res/.

Points:

1. Disable to access files (except files under assets/ and res/ in apk).
2. You may enable JavaScript.

WebViewAssetsActivity.java

```
package org.jssec.webview.assets;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
    /**
     * Show contents in assets
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView webView = (WebView) findViewById(R.id.webView);
        WebSettings webSettings = webView.getSettings();

        // *** POINT 1 *** Disable to access files (except files under assets/ and res/ in this apk)
        webSettings.setAllowFileAccess(false);

        // *** POINT 2 *** Enable JavaScript (Optional)
        webSettings.setJavaScriptEnabled(true);

        // Show contents which were stored under assets/ in this apk
        webView.loadUrl("file:///android_asset/sample/index.html");
    }
}
```


4.9.1.2. Show Only Contents which Are Managed In-house

You can enable JavaScript to show only contents which are managed in-house only if your web service and your Android application can take proper actions to secure both of them.

- Web service side actions:
  As Figure 4.9-2 shows, your web service can only refer to contents which are managed in-house. In addition, the web service is needed to take appropriate security action. Because there is potential risk if contents which your web service refers to may have risk; such as malicious attack code injection, data manipulation, etc.
  Please refer to "4.9.2.1 Enable JavaScript Only If Contents Are Managed In-house (Required)."

- Android application side actions:
  Using HTTPS, the application should establish network connection to your managed web service only if the certification is trusted.

The following sample code is an activity to show contents which are managed in-house.

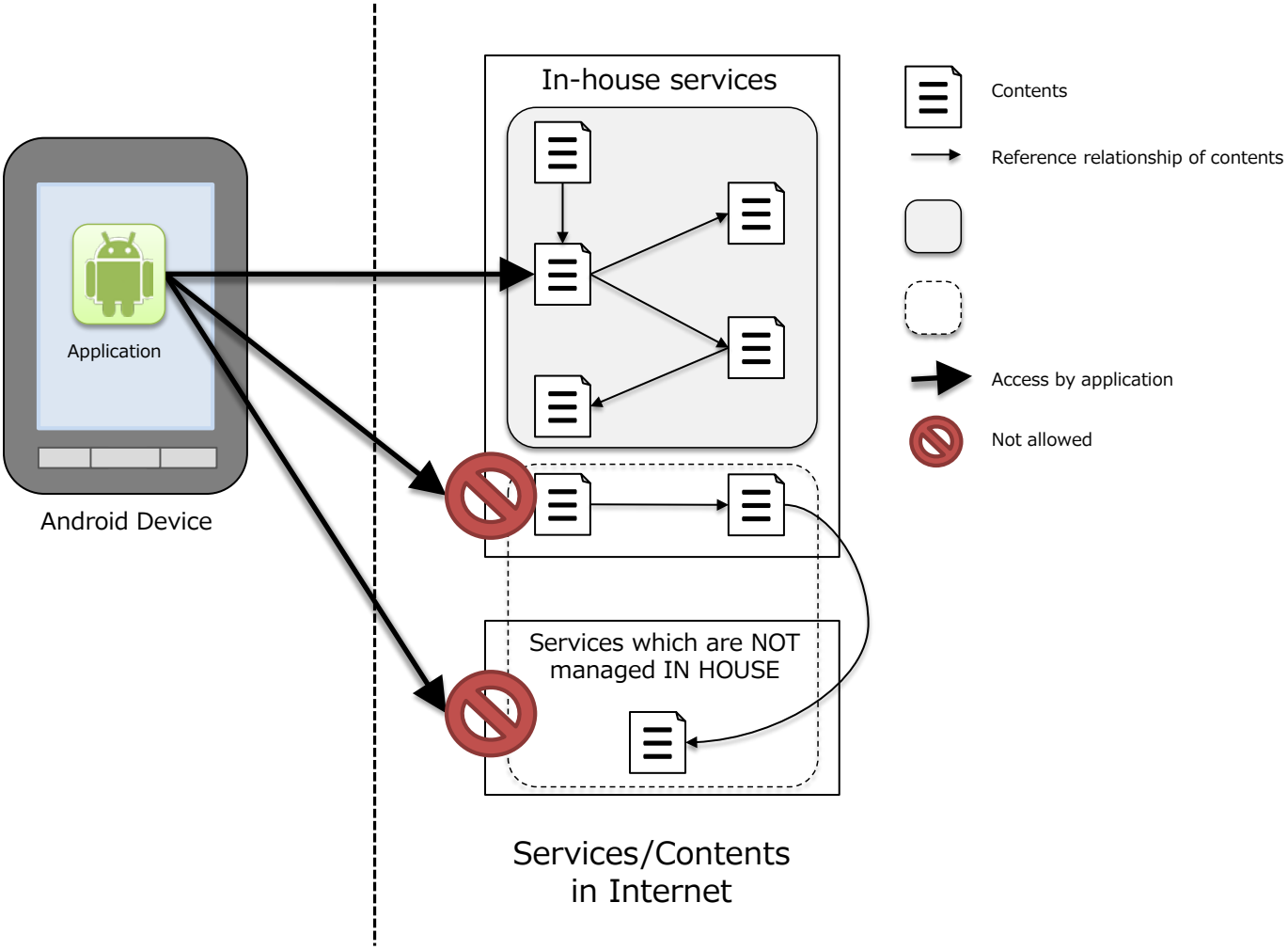


Figure 4.9-2 Accessible contents and Non-accessible contents from application

Points:

1. Handle SSL error from WebView appropriately.
2. (Optional) Enable JavaScript of WebView.
3. Restrict URLs to HTTPS protocol only.
4. Restrict URLs to in-house.

WebViewTrustedContentsActivity.java

```
package org.jssec.webview.trustedcontents;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.net.http.SslCertificate;
import android.net.http.SslError;
import android.os.Bundle;
import android.webkit.SslErrorHandler;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class WebViewTrustedContentsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        WebView webView = (WebView) findViewById(R.id.webView);

        webView.setWebViewClient(new WebViewClient() {
            @Override
            public void onReceivedSslError(WebView view,
                    SslErrorHandler handler, SslError error) {
                // *** POINT 1 *** Handle SSL error from WebView appropriately
                // Show SSL error dialog.
                AlertDialog dialog = createSslErrorDialog(error);
                dialog.show();

                // *** POINT 1 *** Handle SSL error from WebView appropriately
                // Abort connection in case of SSL error
                // Since, there may be some defects in a certificate like expiration of validity,
                // or it may be man-in-the-middle attack.
                handler.cancel();
            }
        });

        // *** POINT 2 *** Enable JavaScript (optional)
        // in case to show contents which are managed in house.
        webView.getSettings().setJavaScriptEnabled(true);

        // *** POINT 3 *** Restrict URLs to HTTPS protocol only
        // *** POINT 4 *** Restrict URLs to in-house
        webView.loadUrl("https://url.to.your.contents/");
    }

    private AlertDialog createSslErrorDialog(SslError error) {
        // Error message to show in this dialog
        String errorMsg = createErrorMessage(error);
        // Handler for OK button
        DialogInterface.OnClickListener onClickOk = new DialogInterface.OnClickListener() {
            @Override
```



```
            public void onClick(DialogInterface dialog, int which) {
                setResult(RESULT_OK);
            }
        };
        // Create a dialog
        AlertDialog dialog = new AlertDialog.Builder(
                WebViewTrustedContentsActivity.this).setTitle("SSL connection error")
                .setMessage(errorMsg).setPositiveButton("OK", onClickOk)
                .create();
        return dialog;
    }

    private String createErrorMessage(SslError error) {
        SslCertificate cert = error.getCertificate();
        StringBuilder result = new StringBuilder()
        .append("The site's certification is NOT valid. Connection was disconnected.¥n¥nError:¥n");
        switch (error.getPrimaryError()) {
        case SslError.SSL_EXPIRED:
            result.append("The certificate is no longer valid.¥n¥nThe expiration date is ")
            .append(cert.getValidNotAfter());
            return result.toString();
        case SslError.SSL_IDMISMATCH:
            result.append("Host name doesn't match. ¥n¥nCN=")
            .append(cert.getIssuedTo().getCName());
            return result.toString();
        case SslError.SSL_NOTYETVALID:
            result.append("The certificate isn't valid yet.¥n¥nIt will be valid from ")
            .append(cert.getValidNotBefore());
            return result.toString();
        case SslError.SSL_UNTRUSTED:
            result.append("Certificate Authority which issued the certificate is not reliable.¥n¥nCertificate Authori
ty¥n")
            .append(cert.getIssuedBy().getDName());
            return result.toString();
        default:
            result.append("Unknown error occured. ");
            return result.toString();

        }
    }

}
```

4.9.1.3. Show Contents which Are Not Managed In-house

Don't enable JavaScript if your application shows contents which are not managed in house because there is potential risk to access to malicious content.

The following sample code is an activity to show contents which are not managed in-house.

This sample code shows contents specified by URL which user inputs through address bar. Please note that JavaScript is disabled and connection is aborted when SSL error occurs. The error handling is the same as "4.9.1.2 Show Only Contents which Are Managed In-house" for the details of HTTPS communication. Please refer to "5.4 Communicating via HTTPS" for the details also.

Points:

1. Handle SSL error from WebView appropriately.
2. Disable JavaScript of WebView.

WebViewUntrustActivity.java

```
package org.jssec.webview.untrust;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.net.http.SslCertificate;
import android.net.http.SslError;
import android.os.Bundle;
import android.view.View;
import android.webkit.SslErrorHandler;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;

public class WebViewUntrustActivity extends Activity {
    /*
     * Show contents which are NOT managed in-house (Sample program works as a simple browser)
     */

    private EditText textUrl;
    private Button buttonGo;
    private WebView webView;

    // Activity definition to handle any URL request
    private class WebViewUnlimitedClient extends WebViewClient {

        @Override
        public boolean shouldOverrideUrlLoading(WebView webView, String url) {
            webView.loadUrl(url);
            textUrl.setText(url);
            return true;
        }

        // Start reading Web page
        @Override
```



```
        public void onPageStarted(WebView webview, String url, Bitmap favicon) {
            buttonGo.setEnabled(false);
            textUrl.setText(url);
        }

        // Show SSL error dialog
        // And abort connection.
        @Override
        public void onReceivedSslError(WebView webview,
                SslErrorHandler handler, SslError error) {

            // *** POINT 1 *** Handle SSL error from WebView appropriately
            AlertDialog errorDialog = createSslErrorDialog(error);
            errorDialog.show();
            handler.cancel();
            textUrl.setText(webview.getUrl());
            buttonGo.setEnabled(true);
        }

        // After loading Web page, show the URL in EditText.
        @Override
        public void onPageFinished(WebView webview, String url) {
            textUrl.setText(url);
            buttonGo.setEnabled(true);
        }
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        webView = (WebView) findViewById(R.id.webview);
        webView.setWebViewClient(new WebViewUnlimitedClient());

        // *** POINT 2 *** Disable JavaScript of WebView
        // Explicitly disable JavaScript even though it is disabled by default.
        webView.getSettings().setJavaScriptEnabled(false);

        webView.loadUrl(getString(R.string.texturl));
        textUrl = (EditText) findViewById(R.id.texturl);
        buttonGo = (Button) findViewById(R.id.go);
    }

    public void onClickButtonGo(View v) {
        webView.loadUrl(textUrl.getText().toString());
    }

    private AlertDialog createSslErrorDialog(SslError error) {
        // Error message to show in this dialog
        String errorMsg = createErrorMessage(error);
        // Handler for OK button
        DialogInterface.OnClickListener onClickOk = new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                setResult(RESULT_OK);
            }
        };
        // Create a dialog
        AlertDialog dialog = new AlertDialog.Builder(
```

```
                WebViewUntrustActivity.this).setTitle("SSL connection error")
                .setMessage(errorMsg).setPositiveButton("OK", onClickOk)
                .create();
        return dialog;
    }

    private String createErrorMessage(SslError error) {
        SslCertificate cert = error.getCertificate();
        StringBuilder result = new StringBuilder()
        .append("The site's certification is NOT valid. Connection was disconnected.¥n¥nError:¥n");
        switch (error.getPrimaryError()) {
        case SslError.SSL_EXPIRED:
            result.append("The certificate is no longer valid.¥n¥nThe expiration date is ")
            .append(cert.getValidNotAfter());
            return result.toString();
        case SslError.SSL_IDMISMATCH:
            result.append("Host name doesn't match. ¥n¥nCN=")
            .append(cert.getIssuedTo().getCName());
            return result.toString();
        case SslError.SSL_NOTYETVALID:
            result.append("The certificate isn't valid yet.¥n¥nIt will be valid from ")
            .append(cert.getValidNotBefore());
            return result.toString();
        case SslError.SSL_UNTRUSTED:
            result.append("Certificate Authority which issued the certificate is not reliable.¥n¥nCertificate Authori
ty¥n")
            .append(cert.getIssuedBy().getDName());
            return result.toString();
        default:
            result.append("Unknown error occured. ");
            return result.toString();
        }
    }
}
```


4.9.2. Rule Book

Comply with following rule when you need to use WebView.

1.	Enable JavaScript Only If Contents Are Managed In-house	(Required)
2.	Use HTTPS to Communicate to Servers which Are Managed In-house	(Required)
3.	Disable JavaScript to Show URLs Which Are Received through Intent, etc.	(Required)
4.	Handle SSL Error Properly	(Required)

4.9.2.1. Enable JavaScript Only If Contents Are Managed In-house (Required)

What we have to pay attention on WebView is whether we enable the JavaScript or not. As principle, we can only enable the JavaScript only IF the application will access to services which are managed in-house. And you must not enable the JavaScript if there is possibility to access services which are not managed in-house.

Services managed In-house

In case that application accesses contents which are developed IN HOUSE and are distributed through servers which are managed IN HOUSE, we can say that the contents are ONLY modified by your company. In addition, it is also needed that each content refers to only contents stored in the servers which have proper security.

In this scenario, we can enable JavaScript on the WebView. Please refer to "4.9.1.2 Show Only Contents which Are Managed In-house" also.

And you can also enable JavaScript if your application shows only contents stored under assets/ and res/ directory in the apk. Please refer to "4.9.1.1 Show Only Contents Stored under assets/res Directory" also.

Services unmanaged in-house

You must NOT think you can secure safety on contents which are NOT managed IN HOUSE. Therefore you have to disable JavaScript. Please refer to "4.9.1.3 Show Contents which Are Not Managed In-house."

In addition, you have to disable JavaScript if the contents are stored in external storage devices; such as microSD because other application can modify the contents.

4.9.2.2. Use HTTPS to Communicate to Servers which Are Managed In-house (Required)

You have to use HTTPS to communicate to servers which are managed in-house because there is potential risk of spoofing the services by malicious third party.

Please refer to both "4.9.2.4 Handle SSL Error Properly (Required)," and "5.4 Communicating via HTTPS" .

4.9.2.3. Disable JavaScript to Show URLs Which Are Received through Intent, etc. (Required)

Don't enable JavaScript if your application needs to show URLs which are passed from other application as Intent, etc. Because there is potential risk to show malicious web page with malicious JavaScript.

Sample code in the section "4.9.1.2 Show Only Contents which Are Managed In-house," uses fixed value URL to show contents which are managed in-house, to secure safety.

If you need to show URL which is received from Intent, etc, you have to confirm that URL is in managed URL in-house. In short, the application has to check URL with white list which is regular expression, etc. In addition, it should be HTTPS.

4.9.2.4. Handle SSL Error Properly (Required)

You have to terminate the network communication and inform error notice to user when SSL error happens on HTTPS communication.

SSL error shows invalid server certification risk or MTIM (man-in-the-middle attack) risk. Please note that WebView has NO error notice mechanism regarding SSL error. Therefore your application has to show the error notice to inform the risk to the user. Please refer to sample code in the section of "4.9.1.2 Show Only Contents which Are Managed In-house," and "4.9.1.3 Show Contents which Are Not Managed In-house".

In addition, your application MUST terminate the communication with the error notice.
In other words, you MUST NOT do following.

- Ignore the error to keep the transaction with the service.
- Retry HTTP communication instead of HTTPS.

Please refer to the detail described in "5.4 Communicating via HTTPS".

WebView's default behavior is to terminate the communication in case of SSL error. Therefore what we need to add is to show SSL error notice. And then we can handle SSL error properly.

5. How to use Security Functions

There are various security functions prepared in Android, like encryption, digital signature and permission etc. If these security functions are not used correctly, security functions don't work efficiently and loophole will be prepared. This chapter will explain how to use the security functions properly.

5.1. Creating Password Input Screens

5.1.1. Sample Code

When creating password input screen, some points to be considered in terms of security, are described here. Only what is related to password input is mentioned, here. Regarding how to save password, another articles is planned to be published is future edition.

Figure 5.1-1

Points:

1. The input password should be mask displayed (Display with *)
2. Provide the option to display the password in a plain text.
3. Alert a user that displaying password in a plain text has a risk.

Points: When handling the last Input password, pay attention the following points along with the above points.

4. In the case there is the last input password in an initial display, display the fixed digit numbers of black dot as dummy in order not that the digits number of last password is guessed.
5. When the dummy password is displayed and the "Show password" button is pressed, clear the last input password and provide the state for new password input.
6. When last input password is displayed with dummy, in case user tries to input password, clear the last input password and treat new user input as a new password.

password_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <!-- Label for password item -->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/password" />

    <!-- Label for password item -->
    <!-- *** POINT 1 *** The input password must be masked (Display with black dot) -->
    <EditText
        android:id="@+id/password_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/hint_password"
        android:password="true" />

    <!-- *** POINT 2 *** Provide the option to display the password in a plain text -->
    <CheckBox
        android:id="@+id/password_display_check"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/display_password" />

    <!-- *** POINT 3 *** Alert a user that displaying password in a plain text has a risk. -->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/alert_password" />

    <!-- Cancel/OK button -->
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:gravity="center"
        android:orientation="horizontal" >

        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```



```
            android:onClick="onClickCancelButton"
            android:text="@android:string/cancel" />

        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="onClickOkButton"
            android:text="@android:string/ok" />
    </LinearLayout>
</LinearLayout>
```

Implementation for 3 methods which are located at the bottom of PasswordActivity.java, should be adjusted depends on the purposes.

- private String getPreviousPassword()
- private void onClickCancelButton(View view)
- private void onClickOkButton(View view)

PasswordActivity.java

```
package org.jssec.android.password.passwordinputui;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.text.InputType;
import android.text.TextWatcher;
import android.view.View;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.EditText;
import android.widget.Toast;

public class PasswordActivity extends Activity {

    // Key to save the state
    private static final String KEY_DUMMY_PASSWORD = "KEY_DUMMY_PASSWORD";

    // View inside Activity
    private EditText mPasswordEdit;
    private CheckBox mPasswordDisplayCheck;

    // Flag to show whether password is dummy display or not
    private boolean mIsDummyPassword;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.password_activity);

        // Get View
        mPasswordEdit = (EditText) findViewById(R.id.password_edit);
        mPasswordDisplayCheck = (CheckBox) findViewById(R.id.password_display_check);

        // Whether last Input password exist or not.
```

```
        if (getPreviousPassword() != null) {
            // *** POINT 4 *** In the case there is the last input password in an initial display,
            // display the fixed digit numbers of black dot as dummy in order not that the digits number of last passw
ord is guessed.

            // Display should be dummy password.
            mPasswordEdit.setText("**********");
            // To clear the dummy password when inputting password, set text change listener.
            mPasswordEdit.addTextChangedListener(new PasswordEditTextWatcher());
            // Set dummy password flag
            mIsDummyPassword = true;
        }

        // Set a listner to change check state of password display option.
        mPasswordDisplayCheck
                .setOnCheckedChangeListener(new OnPasswordDisplayCheckedChangeListener());
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // Unnecessary when specifying not to regenerate Activity by the change in screen aspect ratio.
        // Save Activity state
        outState.putBoolean(KEY_DUMMY_PASSWORD, mIsDummyPassword);
    }

    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        // Unnecessary when specifying not to regenerate Activity by the change in screen aspect ratio.
        // Restore Activity state
        mIsDummyPassword = savedInstanceState.getBoolean(KEY_DUMMY_PASSWORD);
    }

    /**
     * Process in case password is input
     */
    private class PasswordEditTextWatcher implements TextWatcher {

        public void beforeTextChanged(CharSequence s, int start, int count,
                int after) {
            // Not used
        }

        public void onTextChanged(CharSequence s, int start, int before,
                int count) {
            // *** POINT 6 *** When last Input password is displayed as dummy, in the case an user tries to input pass
word,
            // Clear the last Input password, and treat new user input as new password.
            if (mIsDummyPassword) {
                // Set dummy password flag
                mIsDummyPassword = false;
                // Trim space
                CharSequence work = s.subSequence(start, start + count);
                mPasswordEdit.setText(work);
                // Cursor position goes back the beginning, so bring it at the end.
                mPasswordEdit.setSelection(work.length());
            }
```



```
        }

        public void afterTextChanged(Editable s) {
            // Not used
        }

    }

    /**
     * Process when check of password display option is changed.
     */
    private class OnPasswordDisplayCheckedChangeListener implements
            OnCheckedChangeListener {

        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
            // *** POINT 5 ***  When the dummy password is displayed and the "Show password" button is pressed,
            // clear the last input password and provide the state for new password input.
            if (mIsDummyPassword && isChecked) {
                // Set dummy password flag
                mIsDummyPassword = false;
                // Set password empty
                mPasswordEdit.setText(null);
            }

            // Cursor position goes back the beginning, so memorize the current cursor position.
            int pos = mPasswordEdit.getSelectionStart();

            // *** POINT 2 *** Provide the option to display the password in a plain text
            // Create InputType
            int type = InputType.TYPE_CLASS_TEXT;
            if (isChecked) {
                // Plain display when check is ON.
                type |= InputType.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD;
            } else {
                // Masked display when check is OFF.
                type |= InputType.TYPE_TEXT_VARIATION_PASSWORD;
            }

            // Set InputType to password EditText
            mPasswordEdit.setInputType(type);

            // Set cursor position
            mPasswordEdit.setSelection(pos);
        }

    }

    // Implement the following method depends on application

    /**
     * Get the last Input password
     *
     * @return Last Input password
     */
    private String getPreviousPassword() {
        // When need to restore the saved password, return password character string
        // For the case password is not saved, return null
        return "hirake5ma";
    }
```

```
    /**
     * Process when cancel button is clicked
     *
     * @param view
     */
    public void onClickCancelButton(View view) {
        // Close Activity
        finish();
    }

    /**
     * Process when OK button is clicked
     *
     * @param view
     */
    public void onClickOkButton(View view) {
        // Execute necessary processes like saving password or using for authentication

        String password = null;

        if (mIsDummyPassword) {
            // When dummy password is displayed till the final moment, grant last iInput password as fixed password.
            password = getPreviousPassword();
        } else {
            // In case of not dummy password display, grant the user input password as fixed password.
            password = mPasswordEdit.getText().toString();
        }

        // Display password by Toast
        Toast.makeText(this, "password is ¥"" + password + "¥"",
                Toast.LENGTH_SHORT).show();

        // Close Activity
        finish();
    }
}
```


5.1.2. Rule Book

Follow the below rules when creating password input screen.

1.	Provide the Mask Display Feature, If the Password Is Entered	(Required)
2.	Provide the Option to Display Password in a Plain Text	(Required)
3.	Mask the Password when Activity Is Launched	(Required)
4.	When Displaying the Last Input Password, Dummy Password Must Be Displayed	(Required)

5.1.2.1. Provide the Mask Display Feature, If the Password Is Entered (Required)

Smartphone is often used in crowded places like in a train or in a bus, and the risk that password is peeked by someone. So the function to mask display password is necessary as an application spec.

There are 2 methods to mask display EditText which password is input, one is to specify by layout XML statically and another is to switch in a program dynamically. The former one can be achieved by setting android:password="true" to android:password attribute by EditText tab in layout XML. The latter one can be achieved by adding InputType.TYPE_TEXT_VARIATION_PASSWORD in input type of EditText, by setInputType() method of EditText class.

Sample code of each of them is shown below.

Method to specify in layout XML.

password_activity.xml

```
<!—Password input item -->
<!—Set true for the android:password attribute -->
<EditText
    android:id="@+id/password_edit"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/hint_password"
    android:password="true" />
```

Method to specify in Activity.

PasswordActivity.java

```
    // Set password display type
    // Set TYPE_TEXT_VARIATION_PASSWORD for InputType.
    EditText passwordEdit = (EditText) findViewById(R.id.password_edit);
    int type = InputType.TYPE_CLASS_TEXT
            | InputType.TYPE_TEXT_VARIATION_PASSWORD;
    passwordEdit.setInputType(type);
```

5.1.2.2. Provide the Option to Display Password in a Plain Text (Required)

Password input in Smartphone is done by touch panel input, so compared with keyboard input in PC, miss input may be easily happened. Because of the inconvenience of inputting, user may use the simple password, and it makes more dangerous. In addition, when there's a policy like account is locked due the several times of password input failure, it's necessary to avoid from miss input as much as possible. As a solution of these problems, by preparing an option to display password in a plain text, user can use the safe password.

However, when displaying password in a plain text, it may be sniffed, so when using this option. It's necessary to call user cautions for sniffing from behind. In addition, in case option to display in a plain text is implemented, it's also necessary to prepare the system to auto cancel the plain text display like setting the time of plain display. The restrictions for password plain text display are published in another article in future edition. So, the restrictions for password plain text display are not included in sample code.

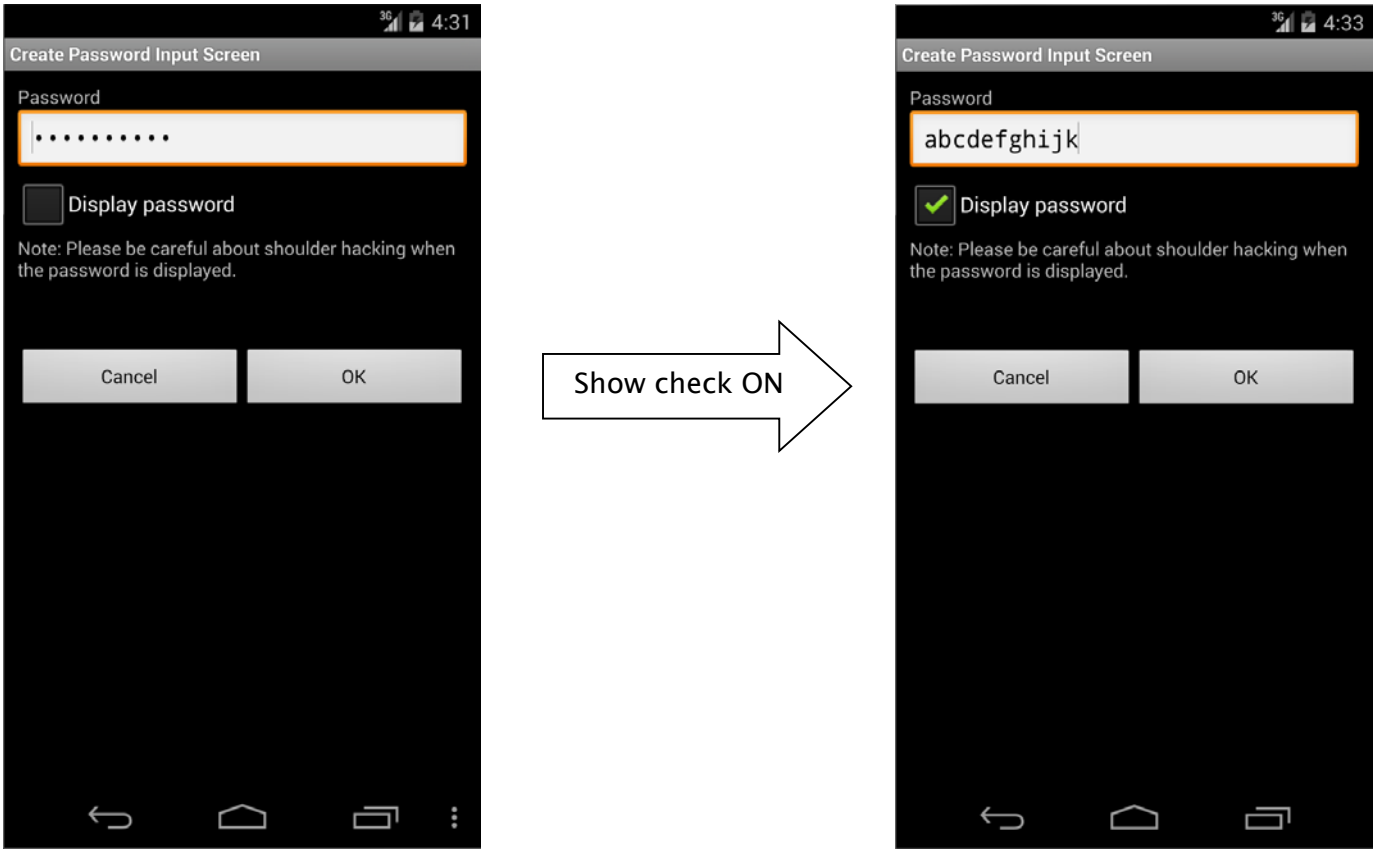


Figure 5.1-2

By specifying InputType of EditText, mask display and plain text display can be switched.

PasswordActivity.java

```
    /**
     * Process when check of password display option is changed.
     */
    private class OnPasswordDisplayCheckedChangeListener implements
            OnCheckedChangeListener {
```



```
    public void onCheckedChanged(CompoundButton buttonView,
            boolean isChecked) {
        // *** POINT 5 ***  When the dummy password is displayed and the "Show password" button is pressed,
        // Clear the last input password and provide the state for new password input.
        if (mIsDummyPassword && isChecked) {
            // Set dummy password flag
            mIsDummyPassword = false;
            // Set password empty
            mPasswordEdit.setText(null);
        }

        // Cursor position goes back the beginning, so memorize the current cursor position.
        int pos = mPasswordEdit.getSelectionStart();

        // *** POINT 2 *** Provide the option to display the password in a plain text
        // Create InputType
        int type = InputType.TYPE_CLASS_TEXT;
        if (isChecked) {
            // Plain display when check is ON.
            type |= InputType.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD;
        } else {
            // Masked display when check is OFF.
            type |= InputType.TYPE_TEXT_VARIATION_PASSWORD;
        }

        // Set InputType to password EditText
        mPasswordEdit.setInputType(type);

        // Set cursor position
        mPasswordEdit.setSelection(pos);
    }

}
```

5.1.2.3. Mask the Password when Activity Is Launched (Required)

To prevent it from a password peeping out, the default value of password display option, should be set OFF, when Activity is launched. The default value should be always defined as safer side, basically.

5.1.2.4. When Displaying the Last Input Password, Dummy Password Must Be Displayed(Required)

When specifying the last input password, not to give the third party any hints for password, it should be displayed as dummy with the fixed digits number of mask characters (* etc). In addition, in the case pressing "Show password" when dummy display, clear password and switch to plain text display mode. It can help to suppress the risk that the last input password is sniffed low, even if the device is passed to a third person like when it's stolen. FYI, In case of dummy display and when a user tries to input password, dummy display should be cancelled, it necessary to turn the normal input state.

When displaying the last Input password, display dummy password.

PasswordActivity.java

```
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.password_activity);

        // Get View
        mPasswordEdit = (EditText) findViewById(R.id.password_edit);
        mPasswordDisplayCheck = (CheckBox) findViewById(R.id.password_display_check);

        // Whether last Input password exist or not.
        if (getPreviousPassword() != null) {
            // *** POINT 4 *** In the case there is the last input password in an initial display,
                 // display the fixed digit numbers of black dot as dummy in order not that the digits number of last
password is guessed.

            // Display should be dummy password.
            mPasswordEdit.setText("**********");
            // To clear the dummy password when inputting password, set text change listener.
            mPasswordEdit.addTextChangedListener(new PasswordEditTextWatcher());
            // Set dummy password flag
            mIsDummyPassword = true;
        }

        - Abbreviation -

    }

    /**
     * Get the last input password.
     *
     * @return the last input password
     */
    private String getPreviousPassword() {
        // To restore the saved password, return the password character string.
        // For the case password is not saved, return null.
        return "hirake5ma";
    }
```

In the case of dummy display, when password display option is turned ON, clear the displayed contents.

PasswordActivity.java

```
    /**
     * Process when check of password display option is changed.
     */
    private class OnPasswordDisplayCheckedChangeListener implements
            OnCheckedChangeListener {

        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
            // *** POINT 5 ***  When the dummy password is displayed and the "Show password" button is pressed,
                 // Clear the last input password and provide the state for new password input.
            if (mIsDummyPassword && isChecked) {
                // Set dummy password flag
                mIsDummyPassword = false;
                // Set password empty
                mPasswordEdit.setText(null);
            }
```



```
            - Abbreviation -

        }

    }
```

In case of dummy display, when user tries to input password, clear dummy display.

PasswordActivity.java

```
    // Key to save the state
    private static final String KEY_DUMMY_PASSWORD = "KEY_DUMMY_PASSWORD";

    - Abbreviation -

    // Flag to show whether password is dummy display or not.
    private boolean mIsDummyPassword;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        - Abbreviation -

        // Whether last Input password exist or not.
        if (getPreviousPassword() != null) {
            // *** POINT 4 *** In the case there is the last input password in an initial display,
            // display the fixed digit numbers of black dot as dummy in order not that the digits number of last passw
ord is guessed.

            // Display should be dummy password.
            mPasswordEdit.setText("**********");
            // To clear the dummy password when inputting password, set text change listener.
            mPasswordEdit.addTextChangedListener(new PasswordEditTextWatcher());
            // Set dummy password flag
            mIsDummyPassword = true;
        }

        - Abbreviation -

    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // Unnecessary when specifying not to regenerate Activity by the change in screen aspect ratio.
        // Save Activity state
        outState.putBoolean(KEY_DUMMY_PASSWORD, mIsDummyPassword);
    }

    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        // Unnecessary when specifying not to regenerate Activity by the change in screen aspect ratio.
        // Restore Activity state
        mIsDummyPassword = savedInstanceState.getBoolean(KEY_DUMMY_PASSWORD);
    }
```

```
    /**
     * Process when inputting password.
     */
    private class PasswordEditTextWatcher implements TextWatcher {

        public void beforeTextChanged(CharSequence s, int start, int count,
                int after) {
            // Not used
        }

        public void onTextChanged(CharSequence s, int start, int before,
                int count) {
            // *** POINT 6 *** When last Input password is displayed as dummy, in the case an user tries to input pass
word,
                 // Clear the last Input password, and treat new user input as new password.
            if (mIsDummyPassword) {
                // Set dummy password flag
                mIsDummyPassword = false;
                // Trim space
                CharSequence work = s.subSequence(start, start + count);
                mPasswordEdit.setText(work);
                // Cursor position goes back the beginning, so bring it at the end.
                mPasswordEdit.setSelection(work.length());
            }
        }

        public void afterTextChanged(Editable s) {
            // Not used
        }

    }
```


5.1.3. Advanced Topics

5.1.3.1. Login Process

The representative example of where password input is required is login process. Here are some Points that need cautions in Login process.

Error message when login fail

In login process, need to input 2 information which is ID(account) and password. When login failure, there are 2 cases. One is ID doesn't exist. Another is ID exists but password is incorrect. If either of these 2 cases is distinguished and displayed in a login failure message, attackers can guess whether the specified ID exists or not. To stop this kind of guess, these 2 cases should not be specified in login failure message, and this message should be displayed as per below.

Message example: Login ID or password is incorrect.

Auto Login function

There is a function to perform auto login by omitting login ID/password input in the next time and later, after successful login process has been completed once. Auto login function can omit the complicated input. So the convenience will increase, but on the other hand, when a Smartphone is stolen, the risk which is maliciously being used by the third party, will follow.

Only the use when damages caused by the malicious third party is somehow acceptable, or only in the case enough security measures can be taken, auto login function can be used. For example, in the case of online banking application, when the device is operated by the third party, financial damage may be caused. So in this case, security measures are necessary along with auto login function. There are some possible counter-measures, like [Require re-inputting password just before financial process like payment process occurs], [When setting auto login, call a user for enough attentions and prompt user to secure device lock], etc. When using auto login, it's necessary to investigate carefully considering the convenience and risks along with the assumed counter measures.

5.1.3.2. Changing Password

When changing the password which was once set, following input items should be prepared on the screen.

- Current password
- New password
- New password(for input validation)

When auto login function is introduced, there are possibilities that third party can use an application. In that case, to avoid from changing password unexpectedly, it's necessary to require the current password input. In addition, to decrease the risk of getting into unserviceable state due to miss

inputting new password, it's necessary to require new password input 2 times.

5.1.3.3. Regarding "Make passwords visible" Setting

There is a setting in Android's setting menu, called "Make passwords visible." In case of Android 4.4, it's shown as below.

```
Setting > Security > Make passwords visible
```

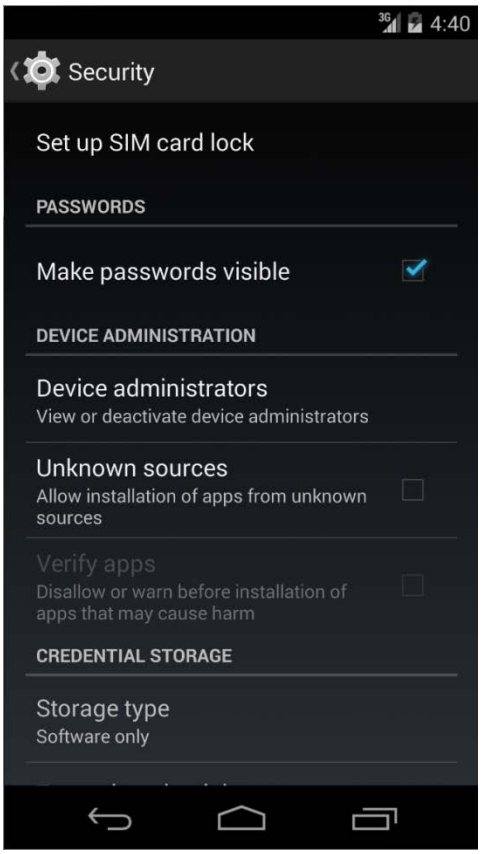


Figure 5.1-3

When turning ON "Make passwords visible" setting, the last input character is displayed in a plain text. After the certain time (about 2 seconds) passed, or after inputting the next character, the characters which was displayed in a plain text is masked. When turning OFF, it's masked right after inputting. This setting affects overall system, and it's applied to all applications which use password display function of EditText.

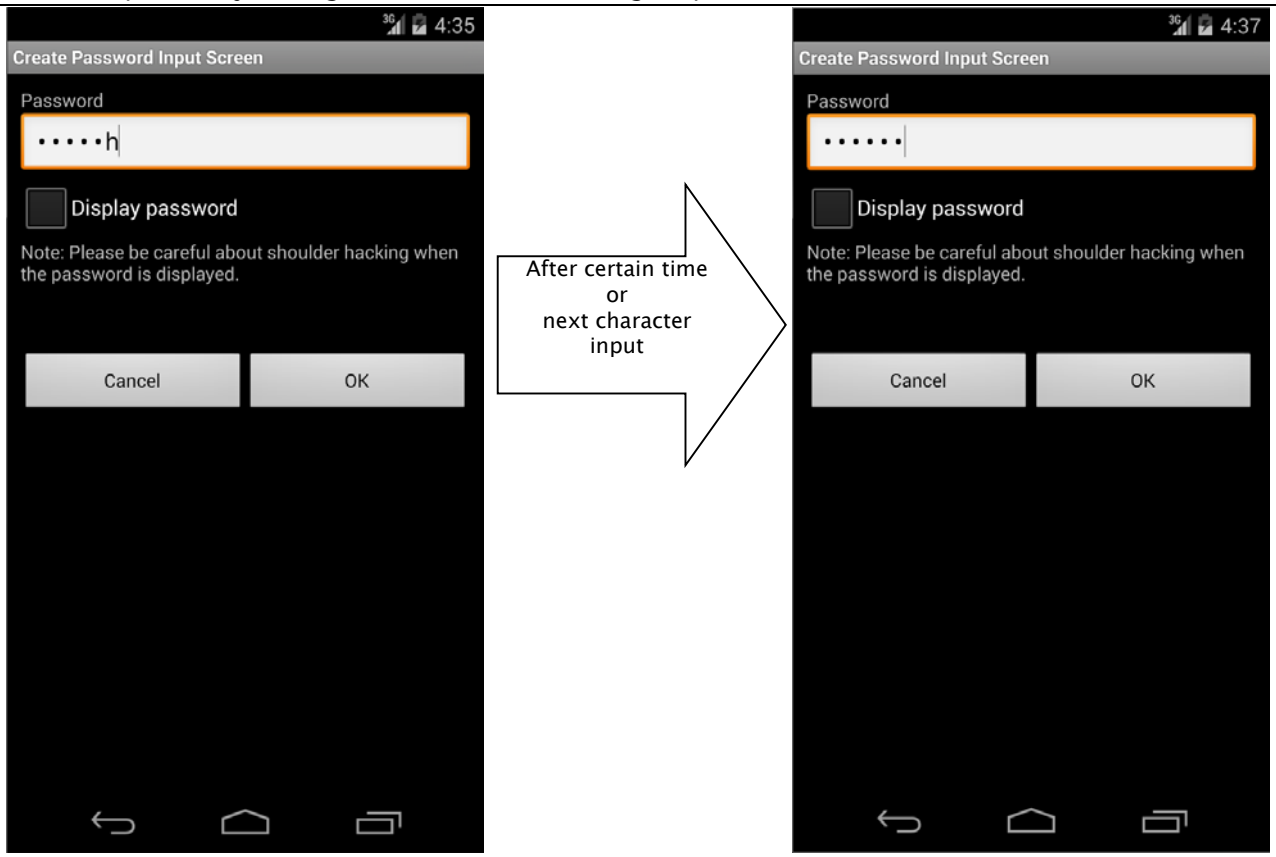

4:35
Create Password Input Screen
Password
•••••h
Display password
Note: Please be careful about shoulder hacking when the password is displayed.
Cancel
OK
After certain time
or
next character
input
4:37
Create Password Input Screen
Password
••••••
Display password
Note: Please be careful about shoulder hacking when the password is displayed.
Cancel
OK

Figure 5.1-4

5.2. Permission and Protection Level

There are four types of Protection Level within permission and they consist of normal, dangerous, signature, and signatureOrSystem. Depending on the Protection Level, permission is referred to as normal permission, dangerous permission, signature permission, or signatureOrSystem permission. In the following sections, such names are used.

5.2.1. Sample Code

5.2.1.1. How to Use System Permissions of Android OS

Android OS has a security mechanism called "permission" that protects its user's assets such as contacts and a GPS feature from a malware. When an application seeks access to such information and/or features, which are protected under Android OS, the application needs to explicitly declare a permission in order to access them. When an application, which has declared a permission that needs user's consent to be used, is installed, the following confirmation screen appears.

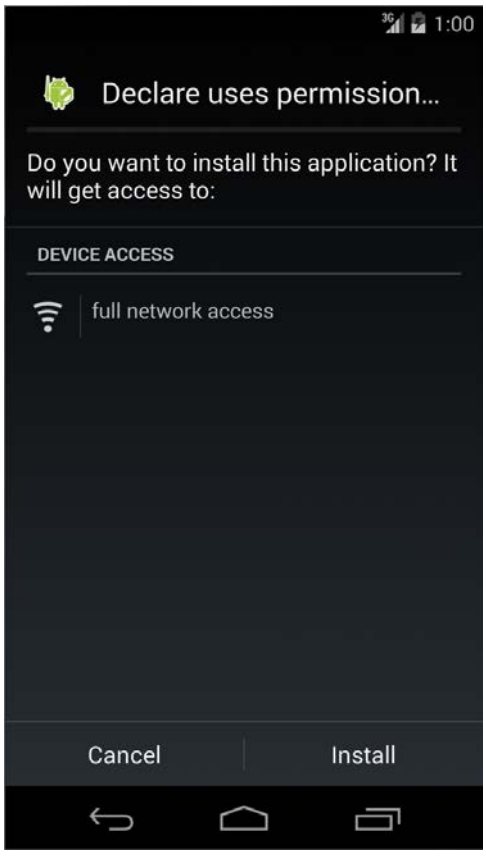


Figure 5.2-1

From this confirmation screen, a user is able to know which types of features and/or information an application is trying to access. If the behavior of an application is trying to access features and/or information that are clearly unnecessary, then there is a high possibility that the application is a malware. Hence, as your application is not suspected to be a malware, declarations of permission to use needs to be minimized.

Points:

1. Declare a permission used in an application with uses-permission.
2. Do not declare any unnecessary permissions with uses-permission.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.usespermission"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 1 *** Declare a permission used in an application with uses-permission -->
    <!-- Permission to access Internet -->
    <uses-permission android:name="android.permission.INTERNET"/>

    <!-- *** POINT 2 *** Do not declare any unnecessary permissions with uses-permission -->
    <!-- If declaring to use Permission that is unnecessary for application behaviors, it gives users a sense of dist
rust. -->

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".MainActivity"
            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>

</manifest>
```

5.2.1.2. How to Communicate Between In-house Applications with In-house-defined Signature Permission

Besides system permissions defined by Android OS, an application can define its own permissions as well. If using an in-house-defined permission (it is an in-house-defined signature permission to be more precise), you can build a mechanism where only communications between in-house applications is permitted. By providing the composite function based on inter-application communication between multiple in-house applications, the applications get more attractive and your business could get more profitable by selling them as series. It is a case of using in-house-defined signature permission.

The sample application "In-house-defined Signature Permission (UserApp)" launchs the sample application "In-house-defined Signature Permission (ProtectedApp)" with Context.startActivity() method. Both applications need to be signed with the same developer key. If keys for signing them are different, the UserApp sends no Intent to the ProtectedApp, and the ProtectedApp processes no Intent received from the UserApp. Furthermore, it prevents malwares from circumventing your own signature permission using the matter related to the installation order as explained in the Advanced section.

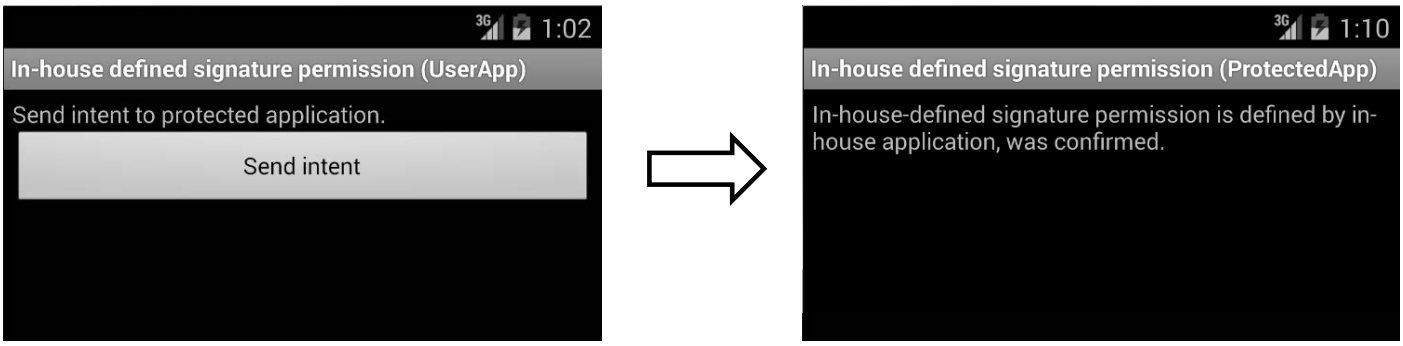


Figure 5.2-2

Points: Application Providing Component

1. Define a permission with protectionLevel="signature".
2. For a component, enforce the permission with its permission attribute.
3. If the component is an activity, you must define no intent-filter.
4. At run time, verify if the signature permission is defined by itself on the program code.
5. When exporting an APK from Eclipse, sign the APK with the same developer key that applications using the component use.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.protectedapp"
    android:versionCode="1"
    android:versionName="1.0" >
```



```
    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 1 *** Define a permission with protectionLevel="signature" -->
    <permission
        android:name="org.jssec.android.permission.protectedapp.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- *** POINT 2 *** For a component, enforce the permission with its permission attribute -->
        <activity
            android:name=".ProtectedActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:permission="org.jssec.android.permission.protectedapp.MY_PERMISSION" >

            <!-- *** POINT 3 *** If the component is an activity, you must define no intent-filter -->
        </activity>
    </application>
</manifest>


ProtectedActivity.java
package org.jssec.android.permission.protectedapp;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.TextView;

public class ProtectedActivity extends Activity {

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.protectedapp.MY_PERMISSION";

    // Hash value of in-house certificate
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" of debug.keystore
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" of keystore
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private TextView mMessageView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        mMessageView = (TextView) findViewById(R.id.messageView);

        // *** POINT 4 *** At run time, verify if the signature permission is defined by itself on the program code
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            mMessageView.setText("In-house defined signature permission is not defined by in-house application.");
            return;
        }

        // *** POINT 4 *** Continue processing only when the certificate matches
        mMessageView.setText("In-house-defined signature permission is defined by in-house application, was confirmed
.");
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // Get the package name of the application which declares a permission named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // Fail if the permission named sigPermName is not a Signature Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // Return the certificate hash value of the application which declares a permission named sigPermName.
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
```



```
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 5 *** When exporting an APK from Eclipse, sign the APK with the same developer key that applications using the component have used.

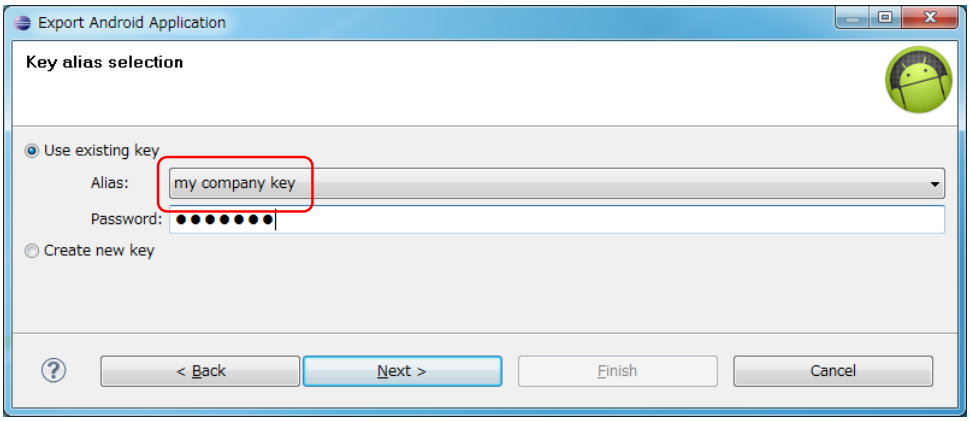


Figure 5.2-3

Points:Application Using Component

6. The same signature permission that the application uses must not be defined.
7. Declare the in-house permission with uses-permission tag.
8. Verify if the in-house signature permission is defined by the application that provides the component on the program code.
9. Verify if the destination application is an in-house application.
10. Use an explicit intent when the destination component is an activity.
11. When exporting an APK from Eclipse, sign the APK with the same developer key that the destination application uses.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.userapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- *** POINT 6 *** The same signature permission that the application uses must not be defined -->

    <!-- *** POINT 7 *** Declare the in-house permission with uses-permission tag -->
    <uses-permission
        android:name="org.jssec.android.permission.protectedapp.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".UserActivity"
            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>
```



```
</manifest>
```

UserActivity.java

```
package org.jssec.android.permission.userapp;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class UserActivity extends Activity {

    // Requested (Destination) application's Activity information
    private static final String TARGET_PACKAGE  = "org.jssec.android.permission.protectedapp";
    private static final String TARGET_ACTIVITY = "org.jssec.android.permission.protectedapp.ProtectedActivity";

    // In-house Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.protectedapp.MY_PERMISSION";

    // Hash value of in-house certificate
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" of debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" of keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onSendButtonClicked(View view) {

        // *** POINT 8 *** Verify if the in-house signature permission is defined by the application that provides the
 component on the program code.

        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "In-house-defined signature permission is not defined by In house application.", Toa
st.LENGTH_LONG).show();
            return;
        }

        // *** POINT 9 *** Verify if the destination application is an in-house application.
```

```
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this, "Requested (Destination) application is not in-house application.", Toast.LENGTH_LON
G).show();
            return;
        }

        // *** POINT 10 *** Use an explicit intent when the destination component is an activity.
        try {
            Intent intent = new Intent();
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);
        } catch(Exception e) {
            Toast.makeText(this,
                    String.format("Exception occurs:%s", e.getMessage()),
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
```



```
import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 11 *** When exporting an APK from Eclipse, sign the APK with the same developer key that the destination application uses.

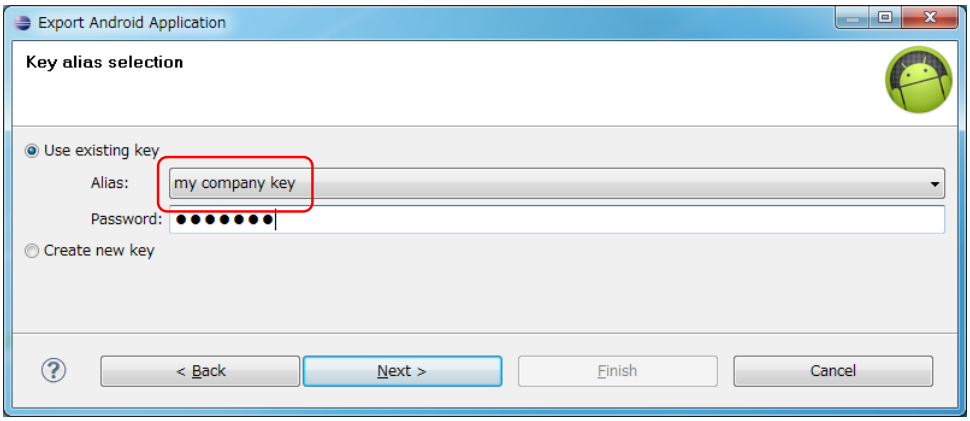


Figure 5.2-4

5.2.1.3. How to verify the hash value of an application's certificate

We will provide an explanation on how to verify the hash value of an application's certificate that appears at different points in this Guidebook. Strictly speaking, the hash value means "the SHA256 hash value of the public key certificate for the developer key used to sign the APK."

How to verify it with Keytool

Using a program called keytool that is bundled with JDK, you can get the hash value (also known as certificate fingerprint) of a public key certificate for the developer key. There are various hash methods such as MD5, SHA1, and SHA256 due to the differences in hash algorithm. However, considering the security strength of the encryption bit length, this Guidebook recommends the use of SHA256. Unfortunately, the keytool bundled to JDK6 that is used in Android SDK does not support SHA256 for calculating hash values. Therefore, it is necessary to use the keytool that is bundled to JDK7.

Example of outputting the content of a debugging certicate of an Android through a keytool

```
> keytool -list -v -keystore < keystore file > -storepass < password >

Type of keystore: JKS
Keystore provider: SUN

One entry is included in a keystore

Other name: androiddebugkey
Date of creation: 2012/01/11
Entry type: PrivateKeyEntry
Length of certificate chain: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4f0cef98
Start date of validity period: Wed Jan 11 11:10:32 JST 2012 End date: Fri Jan 03 11:10:32 JST 2042
Certificate fingerprint:
        MD5:  9E:89:53:18:06:B2:E3:AC:B4:24:CD:6A:56:BF:1E:A1
        SHA1: A8:1E:5D:E5:68:24:FD:F6:F1:ED:2F:C3:6E:0F:09:A3:07:F8:5C:0C
        SHA256: FB:75:E9:B9:2E:9E:6B:4D:AB:3F:94:B2:EC:A1:F0:33:09:74:D8:7A:CF:42:58:22:A2:56:85:1B:0F:85:C6:35
        Signatrue algorithm name: SHA1withRSA
        Version: 3


*******************************************
*******************************************
```

How to Verify it with JSSEC Certificate Hash Value Checker

Without installing JDK7, you can easily verify the certificate hash value by using JSSEC Certificate Hash Value Checker.

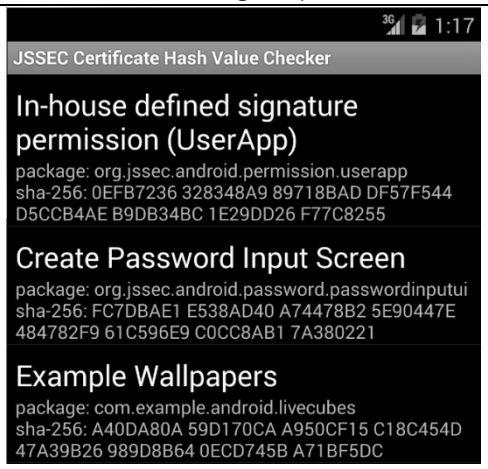


Figure 5.2-5

This is an Android application that displays a list of certificate hash values of applications which are installed in the device. In the Figure above, the 64-character hexadecimal notation string that is shown on the right of "sha-256" is the certificate hash value. The sample code folder, "JSSEC CertHash Checker" that comes with this Guidebook is the set of source codes. If you would like, you can compile the codes and use it.

5.2.2. Rule Book

Be sure to follow the rules below when using in-house permission.

1.	System Dangerous Permissions of Android OS Must Only Be Used for Protecting User Assets (Required)
2.	Your Own Dangerous Permission Must Not Be Used (Required)
3.	Your Own Signature Permission Must Only Be Defined on the Provider-side Application (Required)
4.	Verify If the In-house-defined Signature Permission Is Defined by an In-house Application (Required)
5.	Your Own Normal Permission Should Not Be Used (Recommended)
6.	The String for Your Own Permission Name Should Be of an Extent of the Package Name of Application (Recommended)

5.2.2.1. System Dangerous Permissions of Android OS Must Only Be Used for Protecting User Assets (Required)

Since the use of your own dangerous permission is not recommended (please refer to "5.2.2.2 Your Own Dangerous Permission Must Not Be Used (Required)", we will proceed on the premise of using system dangerous permission of Android OS.

Unlike the other three types of permissions, dangerous permission has a feature that requires the user's consent to the grant of the permission to the application. When installing an application on a device that has declared a dangerous permission to use, the following screen will be displayed. Subsequently, the user is able to know what level of permission (dangerous permission and normal permission) the application is trying to use. When the user taps "install," the application will be granted the permission and then it will be installed.

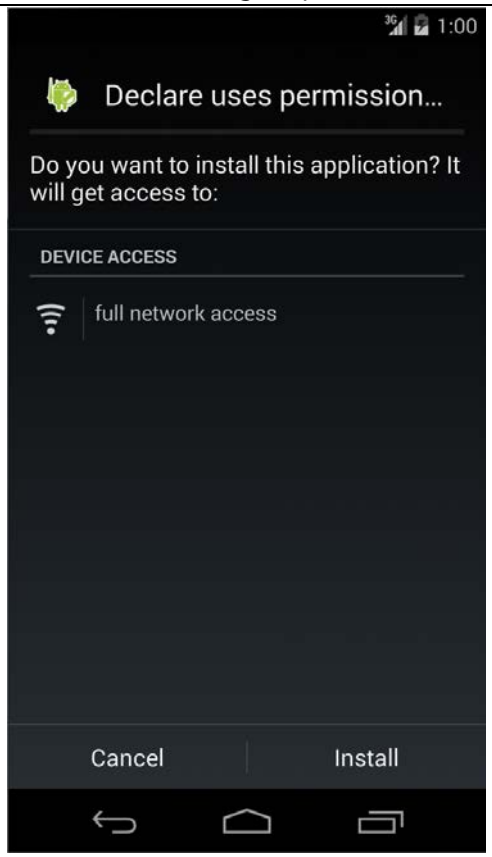


Figure 5.2-6

An application can handle user assets and assets that the developer wants to protect. We must be aware that dangerous permission can protect only user assets because the user is just who the granting of permission is entrusted to. On the other hand, assets that the developer wants to protect cannot be protected by the method above.

For example, suppose that an application has a Component that communicates only with an In-house application, it doesn't permit the access to the Component from any applications of the other companies, and it is implemented that it's protected by dangerous permission. When a user grants permission to an application of another company based on the user's judgment, in-house assets that need to be protected may be exploited by the application granted. In order to provide protection for in-house assets in such cases, we recommend the usage of in-house-defined signature permission.

5.2.2.2. Your Own Dangerous Permission Must Not Be Used (Required)

Even when in-house-defined Dangerous Permission is used, the screen prompt "Asking for the Allowance of Permission from User" is not displayed in some cases. This means that at times the feature that asks for permission based on the judgment of a user, which is the characteristic of Dangerous Permission, does not function. Accordingly, the Guidebook will make the rule "In-house -defined dangerous permission must not be used."

In order to explain it, we assume two types of applications. The first type of application defines a in-house dangerous permission, and it is an application that makes a Component, which is protected by this permission, public. We call this ProtectedApp. The other is another application which we call AttackerApp and it tries to exploit the Component of ProtectedApp. Also we assume that the

AttackerApp not only declares the permission to use it, but also defines the same permission.

AttackerApp can use the Component of a ProtectedApp without the consent of a user in the following cases:

1. When the user installs the AttackerApp, the installation will be completed without the screen prompt that asks for the user to grant the application the dangerous permission.
2. Similarly, when the user installs the ProtectedApp, the installation will be completed without any special warnings.
3. When the user launches the AttackerApp afterwards, the AttackerApp can access the Component of the ProtectedApp without being detected by the user, which can potentially lead to damage.

The cause of this case is explained in the following. When the user tries to install the AttackerApp first, the permission that has been declared for usage with uses-permission is not defined on the particular device yet. Finding no error, Android OS will continue the installation. Since the user consent for dangerous permission is required only at the time of installation, an application that has already been installed will be handled as if it has been granted permission. Accordingly, if the Component of an application which is installed later is protected with the dangerous permission of the same name, the application which was installed beforehand without the user permission will be able to exploit the Component.

Furthermore, since the existence of system dangerous permissions defined by Android OS is guaranteed when an application is installed, the user verification prompt will be displayed every time an application with uses-permission is installed. This problem arises only in the case of self-defined dangerous permission.

At the time of this writing, no viable method to protect the access to the Component in such cases has been developed yet. Therefore, your own dangerous permission must not be used.

5.2.2.3. Your Own Signature Permission Must Only Be Defined on the Provider-side Application (Required)

As demonstrated in, "5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission," the security can be assured by checking the signature permission at the time of executing inter-communications between In-house applications. When using this mechanism, the definition of the permission whose Protection Level is signature must be written in AndroidManifest.xml of the provider-side application that has the Component, but the user-side application must not define the signature permission.

The reason for this is as follows.

We assume that there are multiple user-side applications that have been installed prior to the provider-side application and every user-side application not only has required the signature permission that the provider-side application has defined, but also has defined the same permission. Under these circumstances, all user-side applications will be able to access the provider-side application just after the provider-side application is installed. Subsequently, when the user-side application that was installed first is uninstalled, the definition of the permission also will be deleted and then the permission will turn out to be undefined. As a result, the remaining user-side

applications will be unable to access to the provider-side application.

In this manner, when the user-side application defines a self-defined permission, it can unexpectedly turn out the permission to be undefined. Therefore, only the provider-side application providing the Component that needs to be protected should define the permission, and defining the permission on the user-side must be avoided.

By doing as mentioned just above, the self-defined permission will be applied by Android OS at the time of the installation of the provider-side application, and the permission will turn out to be undefined at the time of the uninstallation of the application. Therefore, since the existance of the permission's definition always corresponds to that of the provider-side application, it is possible to provide an appropriate Component and protect it. Please be aware that this argument stands because regarding in-house-defined signature permission the user-side application is granted the permission regardless of the installation order of applications in inter-communication[12] (post Android 2.2).

5.2.2.4. Verify If the In-house-defined Signature Permission Is Defined by an In-house Application (Required)

Actuality, you cannot say to be secure enough only by declaring a signature permission through AnroidManifest.xml and protecting the Component with the permission. For the details of this issue, please refer to, "5.2.3.1 Characteristics of Android OS that Avoids Self-defined Signature Permission and Its Counter-measures" in the Advanced Topics section.

The following are the steps for using in-house-defined signature permission securely and correctly.

First, write as the followings in AndroidManifest.xml:

1. Define an in-house signature permission in the AndroidManifest.xml of the provider-side application. (definition of permission)
   Example: <permission android:name="xxx" android:protectionLevel="signature" />
2. Enforce the permission with the permission attribute of the Component to be protected in the AndroidManifest.xml of the provider-side application. (enforcement of permission)
   Example: <activity android:permission="xxx" ... >...</activity>
3. Declare the in-house-defined signature permission with the uses-permission tag in the AndroidManifest.xml of every user-side application to access the Component to be protected. (declaration of using permission)
   Example: <uses-permission android:name="xxx" />

Next, implement the followings in the source code.

4. Before processing a request to the Component, first verify that the in-house-defined signature permission has been defined by an in-house application. If not, ignore the request. (protection

[12] If utilizing the protection provided by signature permission for devices Android 2.1 or earlier (outside of the scope of this Guidebook), or If using normal/dangerous permission, the permission will not be granted the user-side application if the user-side application is installed before the provider-side application, the permission remains undefined. Therefore, the Component cannot be accessed even after the provider-side application has been installed.

in the provider-side component)

5. Before accessing the Component, first verify that the in-house-defined signature permission has been defined by an in-house application. If not, do not access the Component (protection in the user-side component).

Lastly, execute the following with the Export function of Eclipse.

6. Sign APKs of all inter-communicating applications with the same developer key.

Here, for specific points on how to implement "Verify that the in-house-defined signature permission has been defined by an In house application", please refer to "5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission".

5.2.2.5. Your Own Normal Permission Should Not Be Used (Recommended)

An application can use a normal permission just by declaring it with uses-permission in AndroidManifest.xml. Therefore, you cannot use a normal permission for the purpose of protecting a Component from a malware installed.

Furthermore, in the case of inter-application communication with self-defined normal permission, whether an application can be granted the permission depends on the order of installation. For example, when you install an application (user-side) that has declared to use a normal permission prior to another application (provider-side) that possesses a Component which has defined the permission, the user-side application will not be able to access the Component protected with the permission even if the provider-side application is installed later.

As a way to prevent the loss of inter-application communication due to the order of installation, you may think of defining the permission in every application in the communication. By this way, even if an user-side application has been installed prior to the provider-side application, all user-side applications will be able to access the provider-side application. However, it will create a situation that the permission is undefined when the user-side application installed first is uninstalled. As a result, even if there are other user-side applications, they will not be able to gain access to the provider-side application.

As stated above, there is a concern of damaging the availability of an application, thus your own normal permission should not be used.

5.2.2.6. The String for Your Own Permission Name Should Be of an Extent of the Package Name of Application (Recommended)

When multiple applications define permissions under the same name, the Protection Level that has been defined by an application installed first will be applied. Protection by signature permission will not be available in the case that the application installed first defines a normal permission and the application installed later defines a signature permission under the same name. Even in the absence of malicious intent, a conflict of permission names among multiple applications could cause behavior s of any applications as an unintended Protection Level. To prevent such accidents, it is recommended that a permission name extends (starts with) the package name of the application

definding the permission as below.

```
(package name).permission.(identifying string)
```

For example, the following name would be preferred when defining a permission of READ access for the package of org.jssec.android.sample.

```
org.jssec.android.sample.permission.READ
```


5.2.3. Advanced Topics

5.2.3.1. Characteristics of Android OS that Avoids Self-defined Signature Permission and Its Counter-measures

Self-defined signature permission is a permission that actualizes inter-application communication between the applications signed with the same developer key. Since a developer key is a private key and must not be public, there is a tendency to use signature permission for protection only in cases where in-house applications communicate with each other.

First, we will describe the basic usage of self-defined signature permission that is explained in the Developer Guide (http://developer.android.com/guide/topics/security/security.html) of Android. However, as it will be explained later, there are problems with regard to the avoidance of permission. Consequently, counter-measures that are described in this Guidebook are necessary.

The followings are the basic usage of self-defined Signature Permission.

1. Define an self-defined signature permission in the AndroidManifest.xml of the provider-side application. (definition of permission)
   Example: <permission android:name="xxx" android:protectionLevel="signature" />
2. Enforce the permission with the permission attribute of the Component to be protected in the AndroidManifest.xml of the provider-side application. (enforcement of permission)
   Example: <activity android:permission="xxx" ... >...</activity>
3. Declare the self-defined signature permission with the uses-permission tag in the AndroidManifest.xml of every user-side application to access the Component to be protected. (declaration of using permission)
   Example: <uses-permission android:name="xxx" />
4. Sign APKs of all inter-communicating applications with the same developer key.

Actually, if the following conditions are fullfilled, this approach will create a loophole to avoid signature permission from being performed.

For the sake of explanation, we call an application that is protected by self-defined signature permission as ProtectedApp, and AttackerApp for an application that has been signed by a different developer key from the ProtectedApp. What a loophole to avoid signature permission from being performed means is, despite the mismatch of the signature for AttackerApp, it is possible to gain access to the Component of ProtectedApp.

Condition 1. An AttackerApp also defines a normal permission (strictly speaking, signature permission is also acceptable) under the same name as the signature permission which has been defined by the ProtectedApp.
Example: <permission android:name=" xxx" android:protectionLevel="normal" />
Condition 2. The AttackerApp declares the self-defined normal permission with uses-permission.
Example: <uses-permission android:name="xxx" />
Condition 3. The AttackerApp has installed on the device prior to the ProtectedApp.

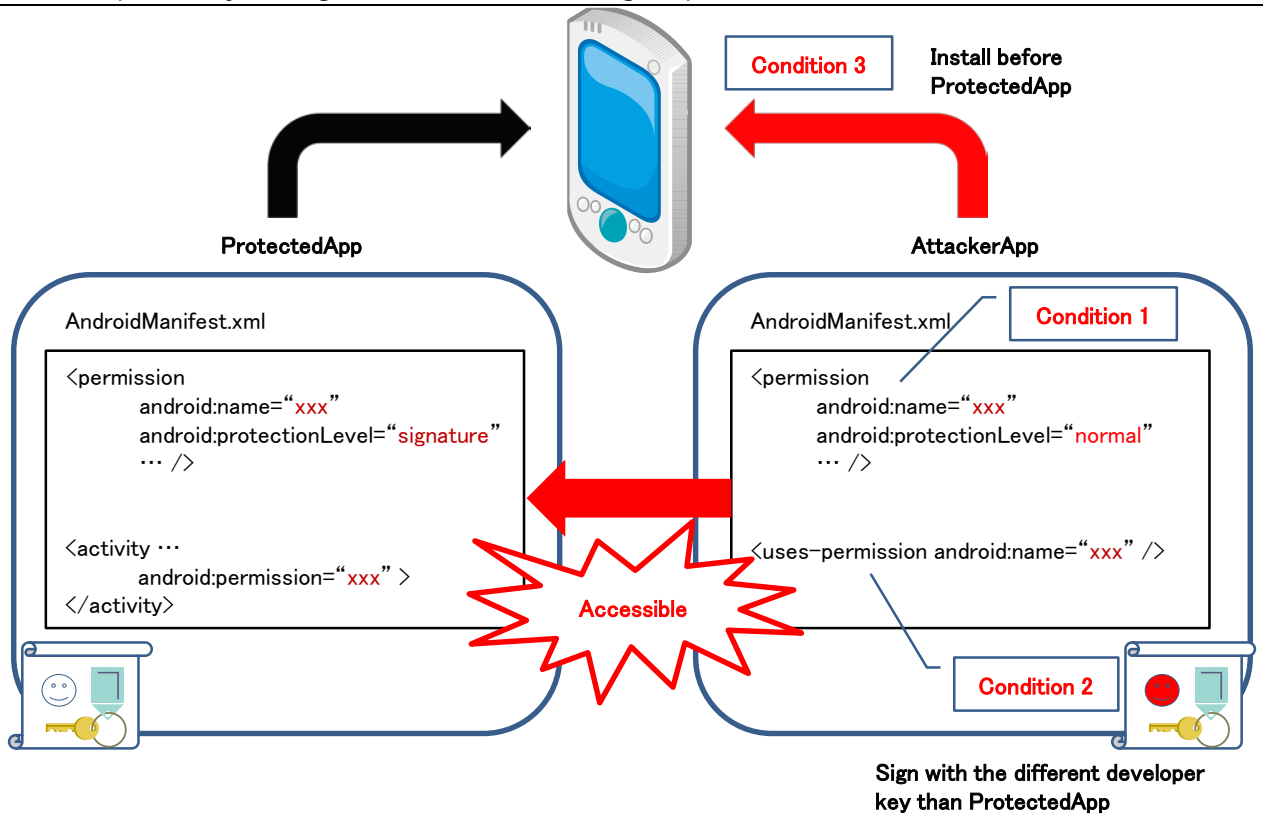


Figure 5.2-7

The permission name that is necessary to meet Condition 1 and Condition 2 can easily be known by an attacker taking AndroidManifest.xml out from an APK file. The attacker also could satisfy Condition 3 with a certain amount of effort (e.g. deceiving a user).

There is a risk of self-defined signature permission to evade protection if only the basic usage is adopted, and a counter-measure to prevent such loopholes is needed. Specificcally, you could find how to solve the above-mentioned issues by using the method described in "5.2.2.4 Verify If the In-house-defined Signature Permission Is Defined by an In-house Application".

5.2.3.2. Falsification of AndroidManifest.xml by a User

We have already touched on the case that a Protection Level of self-defined permission could be changed as not intended. To prevent malfunctioning due to such cases, it has been needed to implement some sort of counter-measures on the source-code side of Java. From the viewpoint of AndroidManifest.xml falsification, we will talk about the counter-measures to be taken on the source-code side. We will demonstrate a simple case of installation that can detect falsifications. However, please note that these counter-measures are little effective against professional hackers who falsify with criminal intent.

This section is about the falsification of an application and users with malicious intent. Although this is originally outside of the scope of a Guidebook, from the fact that this is related to Permission and the tools for such falsification are provided in public as Android applications, we decided to mention it as "Simple counter-measures against amateur hackers".

It must be remembered that applications that can be installed from Google Play are applications that

can be falsified without root privilege. The reason is that applications that can rebuild and sign APK files with altered AndroidManifest.xml are distributed. By using these applications, anyone can delete any permission from applications they have installed.

As an example, there seems to be cases of rebuilding APKs with different signatures altering AndroidManifest.xml with INTERNET permission removed to render advertising modules attached in applications as useless. There are some users who praise these types of tools due to the fact that no personal information is leaked anywhere. As these ads which are attached in applications stop functioning, such actions cause monetary damage for developers who are counting on ad revenue. And it is believed that most of the users don't have any compunction.

In the following code, we show an instance of implementation that an application that has declared INTERNET permission with uses-permission verifies if INTERNET permission is described in the AndroidManifest.xml of itself at run time.

```
public class CheckPermissionActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Acquire Permission defined in AndroidManifest.xml
        List<String> list = getDefinedPermissionList();

        // Detect falsification
        if( checkPermissions(list) ){
            // OK
            Log.d("dbg", "OK.");
        }else{
            Log.d("dbg", "manifest file is stale.");
            finish();
        }
    }

    /**
     * Acquire Permission through list that was defined in AndroidManifest.xml
     * @return
     */
    private List<String> getDefinedPermissionList(){
        List<String> list = new ArrayList<String>();
        list.add("android.permission.INTERNET");
        return list;
    }

    /**
     * Verify that Permission has not been changed Permission
     * @param permissionList
     * @return
     */
    private boolean checkPermissions(List<String> permissionList){
        try {
            PackageInfo packageInfo = getPackageManager().getPackageInfo(
                    getPackageName(), PackageManager.GET_PERMISSIONS);
            String[] permissionArray = packageInfo.requestedPermissions;
            if (permissionArray != null) {
```

```
                for (String permission : permissionArray) {
                    if(! permissionList.remove(permission)){
                        // Unintended Permission has been added
                        return false;
                    }
                }
            }

            if(permissionList.size() == 0){
                // OK
                return true;
            }

        } catch (NameNotFoundException e) {
        }

        return false;
    }
}
```

5.2.3.3. Detection of APK Falsification

We explained about detecting the falsification of permissions by a user in "5.2.3.2 Falsification of AndroidManifest.xml by a User". However, the falsification of applications is not limited to permission only, and there are many other cases where applications are appropriated without any changes in the source code. For example, it is a case where they distribute other developers' applications (falsified) in the market as if they were their own applications just by replacing resources to their own. Here, we will show a more generic method to detect the falsification of an APK file.

In order to falsify an APK, it is needed to decode the APK file into folders and files, modify their contents, and then rebuild them into a new APK file. Since the falsifier does not have the key of the original developer, he would have to sign the new APK file with his own key. As the falsification of an APK inevitably brings with a change in signature (certificate), it is possible to detect whether an APK has been falsified at run time by comparing the certificate in the APK and the developer's certificate embedded in the source code as below.

The following is a sample code. Also, a professional hacker will be able to easily circumvent the detection of falsification if this implementation example is used as it is. Please apply this sample code to your application by being aware that this is a simple implementation example.

Points:

1. Verify that an application's certificate belongs to the developer before major processing is started.

SignatureCheckActivity.java

```
package org.jssec.android.permission.signcheckactivity;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.Utils;

import android.app.Activity;
```



```
import android.content.Context;
import android.os.Bundle;
import android.widget.Toast;

public class SignatureCheckActivity extends Activity {
    // Self signed certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" of debug.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" of keystore
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 1 *** Verify that an application's certificate belongs to the developer before major processing i
s started
        if (!PkgCert.test(this, this.getPackageName(), myCertHash(this))) {
            Toast.makeText(this, "Self-sign match  NG", Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        Toast.makeText(this, "Self-sign match  OK", Toast.LENGTH_LONG).show();
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
```

```
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

5.2.3.4. Permission Re-delegation Problem

An application must declare to use permission when accessing contacts or GPS with its information and features that are protected by Android OS. When the permission required is granted, the permission is delegated to the application and the application would be able to access the information and features protected with the permission.

Depending on how the program is designed, the application to which has been delegated (granted) the permission is able to acquire data that is protected with the permission. Furthermore, the application can offer another application the protected data without enforcing the same permission. This is nothing less than permission-less application to access data that is protected by permission. This is virtually the same thing as re-delegating the permission, and this is referred to the Permission Re-delegation Problem. Accordingly, the specification of the permission mechanism of Android only is able to manage permission of direct access from an application to protected data.

A specific example is shown in Figure 5.2-8. The application in the center shows that an application which has declared android.permission.READ_CONTACTS to use it reads contacts and then stores them into its own database. The Permission Re-delegation Problem occurs when information that has been stored is offered to another application without any restriction via Content Provider.

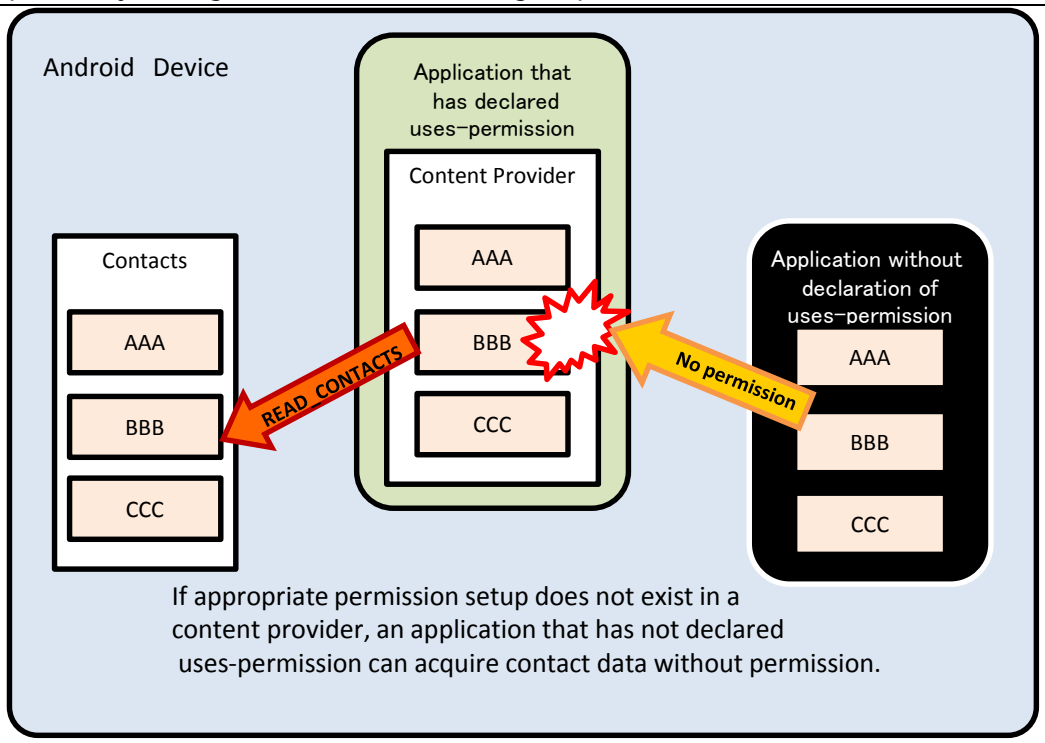


Figure 5.2-8 An Application without Permission Acquires Contacts

As a similar example, an application that has declared android.permission.CALL_PHONE to use it receives a phone number (maybe input by a user) from another application that has not declared the same permission. If that number is being called without the verification of a user, then also there is the Permission Re-delegation Problem.

There are cases where the secondary provision of another application with nearly-intact information asset or functional asset acquired with the permission is needed. In those cases, the provider-side application must demand the same permission for the provision in order to maintain the original level of protection. Also, in the case of only providing a portion of information asset as well as functional asset in a secondary fashion, an appropriate amount of protection is necessary in accordance with the degree of damage that is incurred when a portion of that information or functional asset is exploited. We can use protective measures such as demanding permission as similar to the former, verifying user consent, and setting up restrictions for target applications by using "4.1.1.1 Creating/Using Private Activities," or "4.1.1.4 Creating/Using In-house Activities" etc.

Such Permission Re-delegation Problem is not only limited to the issue of the Android permission. For an Android application, it is generic that the application aquires necessary information/functions from different applications, networks, and storage media. And in many cases, some permissions as well as restrictions are needed to access them. For example, if the provider source is an Android application, it is the permission, if it is a network, then it is the log-in, and if it is a storage media, there will be access restrictions. Therefore, such measures need to be implemented for an application after carefully considering as information/functions are not used in the contrary manner of the user's intention. This is especially important at the time of providing aquired information/functions to another application in a secondary manner or transfering to networks or storage media. Depending on the necessity, you have to enforce permission or restrict usage like the Android permission. Asking for the user's consent is part of the solution.

In the following code, we demonstrate a case where an application that acquires a list from the contact database by using READ_CONTACTS permission enforces the same READ_CONTACTS permission on the information destination source.

Point

1. Enforce the same permission that the provider does.

AndroidManifest.xml

```
<?xml version="1.0" encofing="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.transferpermission"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.READ_CONTACTS"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".TransferPermissionActivity"
            android:label="@string/title_activity_transfer_permission" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <provider
            android:name=".TransferPermissionContentProvider"
            <!-- *** Point1 *** Enforce the same permission that the provider does. -->
            android:authorities="org.jssec.android.permission.transferpermission"
            android:enabled="true"
            android:exported="true"
            android:readPermission="android.permission.READ_CONTACTS" >
        </provider>
    </application>

</manifest>
```

When an application enforces multiple permissions, the above method will not solve it. By using Context#checkCallingPermission() or PackageManager#checkPermission() from the source code, verify whether the invoker application has declared all permissions with uses-permission in the Manifest.

In the case of an Activity

```
public void onCreate(Bundle savedInstanceState) {
...(Snip)
      PackageManager mgr = getPackageManager();
      if (mgr.checkPermission("android.permission.READ_CONTACTS") == PackageManager.PERMISSION_GRANTED
       && mgr.checkPermission("android.permission.WRITE_CONTACTS") == PackageManager.PERMISSION_GRANTED) {
                // Processing during the time when an invoker is correctly declaring to use
                return;
```



```
        }
        finish();
}
```

5.3. Add In-house Accounts to Account Manager

Account Manager is the Android OS's system which centrally manages account information (account name, password) which is necessary for applications to access to online service and authentication token. A user needs to register the account information to Account Manager in advance, and when an application tries to access to online service, Account Manager will automatically provide application authentication token after getting user's permission. The advantage of Account Manager is that an application doesn't need to handle the extremely sensitive information, password.

The structure of account management function which uses Account Manager is as per below Figure 5.3-1. "Requesting application" is the application which accesses the online service, by getting authentication token, and this is above mentioned application. On the other hand, "Authenticator application" is function extension of Account Manager, and by providing Account Manager of an object called Authenticator, as a result Account Manager can manage centrally the account information and authentication token of the online service.Requesting application and Authenticator application don't need to be the separate ones, so these can be implemented as a single application.

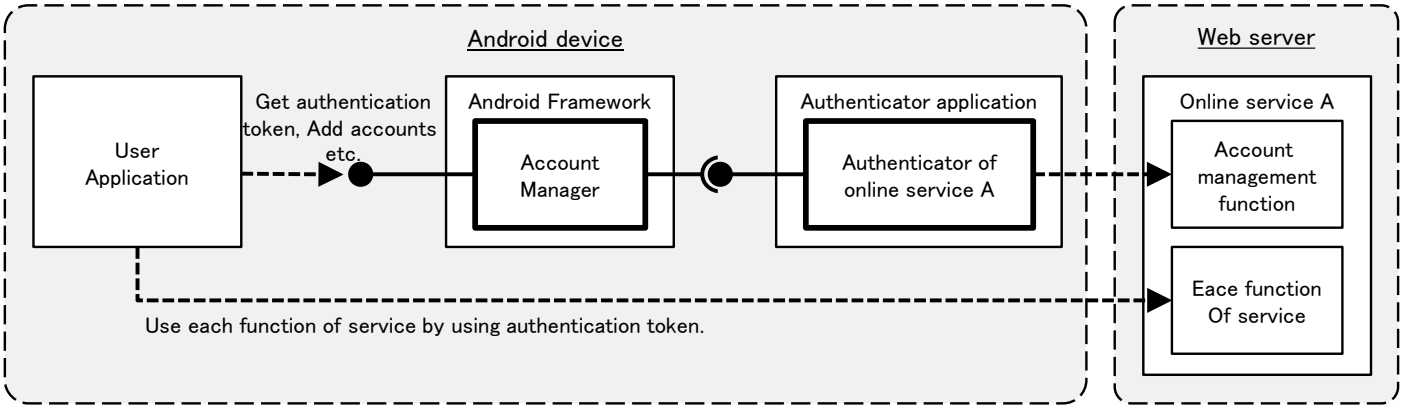


Figure 5.3-1Configuration of account management function which uses Account Manager

Originally, the developer's signature key of user application (requesting application) and Authenticator application can be the different ones. However, only in Android 4.0.x devices, there's an Android Framework bug, and when the signature key of user application and Authenticator application are different, exception occurs in user application, and in-house account cannot be used. The following sample code does not implement any workarounds against this defect. Please refer to "5.3.3.2 Exception Occurs When Signature Keys of User Application and Authenticator Application Are Different, in Android 4.0.x" for details.

5.3.1. Sample Code

"5.3.1.1 Creating In-house account" is prepared as a sample of Authenticator application, and "5.3.1.2 Using In-house Account" is prepared as a sample of requesting application. In sample code set which is distributed in JSSEC's Web site, each of them is corresponded to AccountManager Authenticator and AccountManager User.

5.3.1.1. Creating In-house accounts

Here is the sample code of Authenticator application which enables Account Manager to use the in-house account. There is no Activity which can be launched from home screen in this application. Please pay attention that it's called indirectly via Account Manager from another sample code "5.3.1.2 Using In-house Account."

Points:

1. The service that provides an authenticator must be private.
2. The login screen activity must be implemented in an authenticator application.
3. The login screen activity must be made as a public activity.
4. The explicit intent which the class name of the login screen activity is specified must be set to KEY_INTENT.
5. Sensitive information (like account information or authentication token) must not be output to the log.
6. Password should not be saved in Account Manager.
7. HTTPS should be used for communication between an authenticator and the online services.

Service which gives Account Manager IBinder of Authenticator is defined in AndroidManifest.xml. Specify resource XML file which Authenticator is written, by meta-data.

AccountManager Authenticator/AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.accountmanager.authenticator"
    android:versionCode="1"
    android:versionName="1.0" xmlns:tools="http://schemas.android.com/tools">

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <!-- Necessary Permission to implement Authenticator -->
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Service which gives IBinder of Authenticator to AccountManager -->
        <!-- *** POINT 1 *** The service that provides an authenticator must be private. -->
        <service
            android:name=".AuthenticationService"
            android:exported="false" >
            <!-- intent-filter and meta-data are usual pattern. -->
            <intent-filter>
                <action android:name="android.accounts.AccountAuthenticator" />
            </intent-filter>
            <meta-data
                android:name="android.accounts.AccountAuthenticator"
                android:resource="@xml/authenticator" />
```

```
        </service>

        <!-- Activity for for login screen which is displayed when adding an account -->
        <!-- *** POINT 2 *** The login screen activity must be implemented in an authenticator application. -->
        <!-- *** POINT 3 *** The login screen activity must be made as a public activity. -->
        <activity
            android:name=".LoginActivity"
            android:exported="true"
            android:label="@string/login_activity_title"
            android:theme="@android:style/Theme.Dialog"
            tools:ignore="ExportedActivity" />

</application>

</manifest>
```

Define Authenticator by XML file. Specify account type etc of in-house account.

res/xml/authenticator.xml

```
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="org.jssec.android.accountmanager"
    android:icon="@drawable/ic_launcher"
    android:label="@string/label"
    android:smallIcon="@drawable/ic_launcher" />
```

Service which gives Authenticator's Instance to AccountManager. Easy implementation which returns Instance of JssecAuthenticator class that is Authenticator implemented in this sample by onBind(), is enough.

AuthenticationService.java

```
package org.jssec.android.accountmanager.authenticator;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class AuthenticationService extends Service {

    private JssecAuthenticator mAuthenticator;

    @Override
    public void onCreate() {
        mAuthenticator = new JssecAuthenticator(this);
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mAuthenticator.getIBinder();
    }
}
```


JssecAuthenticator is the Authenticator which is implemented in this sample. It inherits AbstractAccountAuthenticator, and all abstract methods are implemented. These methods are called by Account Manager. At addAccount() and at getAuthToken(), the intent for launching LoginActivity to get authentication token from online service are returned to Account Manager.

JssecAuthenticator.java

```
package org.jssec.android.accountmanager.authenticator;

import android.accounts.AbstractAccountAuthenticator;
import android.accounts.Account;
import android.accounts.AccountAuthenticatorResponse;
import android.accounts.AccountManager;
import android.accounts.NetworkErrorException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;

public class JssecAuthenticator extends AbstractAccountAuthenticator {

    public static final String JSSEC_ACCOUNT_TYPE = "org.jssec.android.accountmanager";
    public static final String JSSEC_AUTHTOKEN_TYPE = "webservice";
    public static final String JSSEC_AUTHTOKEN_LABEL = "JSSEC Web Service";
    public static final String RE_AUTH_NAME = "reauth_name";

    protected final Context mContext;

    public JssecAuthenticator(Context context) {
        super(context);
        mContext = context;
    }

    @Override
    public Bundle addAccount(AccountAuthenticatorResponse response, String accountType,
            String authTokenType, String[] requiredFeatures, Bundle options)
            throws NetworkErrorException {

        AccountManager am = AccountManager.get(mContext);
        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
        Bundle bundle = new Bundle();
        if (accounts.length > 0) {
            // In this sample code, when an account already exists, consider it as an error.
            bundle.putString(AccountManager.KEY_ERROR_CODE, String.valueOf(-1));
            bundle.putString(AccountManager.KEY_ERROR_MESSAGE,
                    mContext.getString(R.string.error_account_exists));
        } else {
            // *** POINT 2 *** The login screen activity must be implemented in an authenticator application.
            // *** POINT 4 *** The explicit intent which the class name of the login screen activity is specified must
 be set to KEY_INTENT.
            Intent intent = new Intent(mContext, LoginActivity.class);
            bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        }
        return bundle;
    }

    @Override
    public Bundle getAuthToken(AccountAuthenticatorResponse response, Account account,
            String authTokenType, Bundle options) throws NetworkErrorException {
```

```
        Bundle bundle = new Bundle();
        if (accountExist(account)) {
            // *** POINT 4 *** KEY_INTENT must be given an explicit intent that is specified the class name of the log
in screen activity.
            Intent intent = new Intent(mContext, LoginActivity.class);
            intent.putExtra(RE_AUTH_NAME, account.name);
            bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        } else {
            // When the specified account doesn't exist, consider it as an error.
            bundle.putString(AccountManager.KEY_ERROR_CODE, String.valueOf(-2));
            bundle.putString(AccountManager.KEY_ERROR_MESSAGE,
                    mContext.getString(R.string.error_account_not_exists));
        }
        return bundle;
    }

    @Override
    public String getAuthTokenLabel(String authTokenType) {
        return JSSEC_AUTHTOKEN_LABEL;
    }

    @Override
    public Bundle confirmCredentials(AccountAuthenticatorResponse response, Account account,
            Bundle options) throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle editProperties(AccountAuthenticatorResponse response, String accountType) {
        return null;
    }

    @Override
    public Bundle updateCredentials(AccountAuthenticatorResponse response, Account account,
            String authTokenType, Bundle options) throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle hasFeatures(AccountAuthenticatorResponse response, Account account,
            String[] features) throws NetworkErrorException {
        Bundle result = new Bundle();
        result.putBoolean(AccountManager.KEY_BOOLEAN_RESULT, false);
        return result;
    }

    private boolean accountExist(Account account) {
        AccountManager am = AccountManager.get(mContext);
        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
        for (Account ac : accounts) {
            if (ac.equals(account)) {
                return true;
            }
        }
        return false;
    }
}
```


This is Login activity which sends an account name and password to online service, and perform login authentication, and as a result, get an authentication token. It's displayed when adding a new account or when getting authentication token again. It's supposed that the actual access to online service is implemented in WebService class.

LoginActivity.java

```
package org.jssec.android.accountmanager.authenticator;

import org.jssec.android.accountmanager.webservice.WebService;

import android.accounts.Account;
import android.accounts.AccountAuthenticatorActivity;
import android.accounts.AccountManager;
import android.content.Intent;
import android.os.Bundle;
import android.text.InputType;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.widget.EditText;

public class LoginActivity extends AccountAuthenticatorActivity {
    private static final String TAG = AccountAuthenticatorActivity.class.getSimpleName();
    private String mReAuthName = null;
    private EditText mNameEdit = null;
    private EditText mPassEdit = null;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        // Display alert icon
        requestWindowFeature(Window.FEATURE_LEFT_ICON);
        setContentView(R.layout.login_activity);
        getWindow().setFeatureDrawableResource(Window.FEATURE_LEFT_ICON,
                android.R.drawable.ic_dialog_alert);

        // Find a widget in advance
        mNameEdit = (EditText) findViewById(R.id.username_edit);
        mPassEdit = (EditText) findViewById(R.id.password_edit);

        // *** POINT 3 *** The login screen activity must be made as a public activity, and suppose the attack access
from other application.
        // Regarding external input, only RE_AUTH_NAME which is String type of Intent#extras, are handled.
        // This external input String is passed toextEdit#setText(), WebService#login(),new Account(),
        // as a parameter,it's verified that there's no problem if any character string is passed.
        mReAuthName = getIntent().getStringExtra(JssecAuthenticator.RE_AUTH_NAME);
        if (mReAuthName != null) {
            // Since LoginActivity is called with the specified user name, user name should not be editable.
            mNameEdit.setText(mReAuthName);
            mNameEdit.setInputType(InputType.TYPE_NULL);
            mNameEdit.setFocusable(false);
            mNameEdit.setEnabled(false);
        }
    }

    // It's executed when login button is pressed.
```

```
    public void handleLogin(View view) {
        String name = mNameEdit.getText().toString();
        String pass = mPassEdit.getText().toString();

        if (TextUtils.isEmpty(name) || TextUtils.isEmpty(pass)) {
            // Process when the inputed value is incorrect
            setResult(RESULT_CANCELED);
            finish();
        }

        // Login to online service based on the inpputted account information.
        WebService web = new WebService();
        String authToken = web.login(name, pass);
        if (TextUtils.isEmpty(authToken)) {
            // Process when authentication failed
            setResult(RESULT_CANCELED);
            finish();
        }

        // Process when login was successful, is as per below.

        // *** POINT 5 *** Sensitive information (like account information or authentication token) must not be outpu
t to the log.
        Log.i(TAG, "WebService login succeeded");


        if (mReAuthName == null) {
            // Register accounts which logged in successfully, to aAccountManager
            // *** POINT 6 *** Password should not be saved in Account Manager.
            AccountManager am = AccountManager.get(this);
            Account account = new Account(name, JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            am.addAccountExplicitly(account, null, null);
            am.setAuthToken(account, JssecAuthenticator.JSSEC_AUTHTOKEN_TYPE, authToken);
            Intent intent = new Intent();
            intent.putExtra(AccountManager.KEY_ACCOUNT_NAME, name);
            intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE,
                    JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            setAccountAuthenticatorResult(intent.getExtras());
            setResult(RESULT_OK, intent);
        } else {
            // Return authentication token
            Bundle bundle = new Bundle();
            bundle.putString(AccountManager.KEY_ACCOUNT_NAME, name);
            bundle.putString(AccountManager.KEY_ACCOUNT_TYPE,
                    JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            bundle.putString(AccountManager.KEY_AUTHTOKEN, authToken);
            setAccountAuthenticatorResult(bundle);
            setResult(RESULT_OK);
        }
        finish();
    }
}
```

Actually, WebService class is dummy implementation here, and this is the sample implementation which supposes authentication is always successful, and fixed character string is returned as an authentication token.

WebService.java

```
package org.jssec.android.accountmanager.webservice;

public class WebService {

    /**
     * Suppose to access to account managemnet function of online service.
     *
     * @param username Account name character string
     * @param password password character string
     * @return Return authentication token
     */
    public String login(String username, String password) {
        // *** POINT 7 *** HTTPS should be used for communication between an authenticator and the online services.
        // Actually, communication process with servers is implemented here, but Omit here, since this is a sample.
        return getAuthToken(username, password);
    }

    private String getAuthToken(String username, String password) {
        // In fact, get the value which uniqueness and impossibility of speculation are guaranteed by the server,
        // but the fixed value is returned without communication here, since this is sample.
        return "c2f981bda5f34f90c0419e171f60f45c";
    }
}
```

5.3.1.2. Using In-house Accounts

Here is the sample code of an application which adds an in-house account and gets an authentication token. When another sample application "5.3.1.1 Creating In-house account" is installed in a device, in-house account can be added or authentication token can be got. "Access request" screen is displayed only when the signature keys of both applications are different.

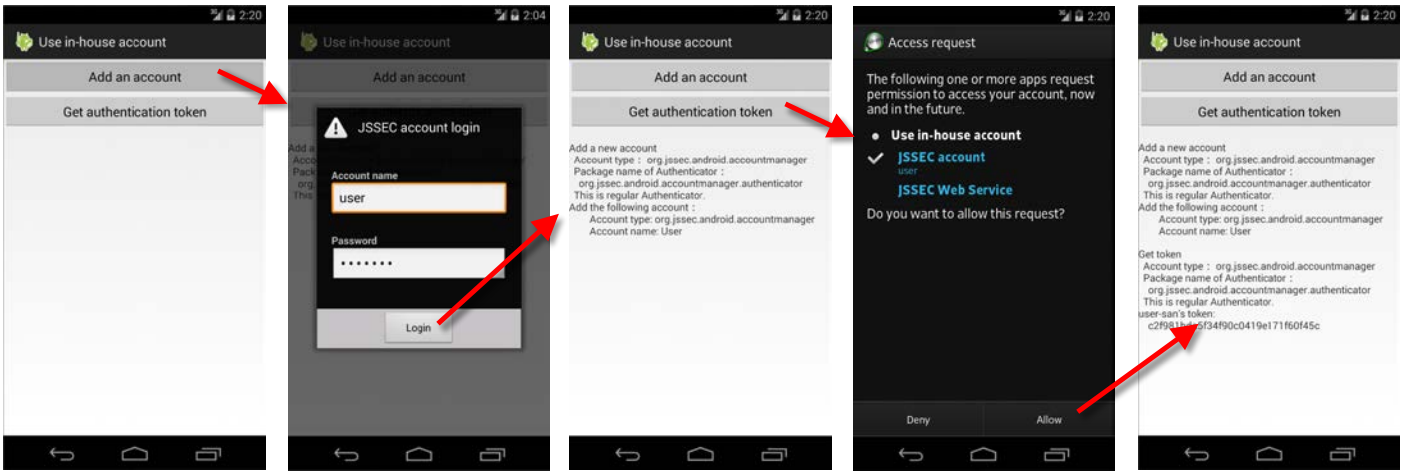


Figure 5.3-2 Behavior screen of sample application AccountManager User

Point:

1. Execute the account process after verifying if the authenticator is regular one.

AndroidManifest.xml of AccountManager user application. Declare to use necessary Permission. Refer to "5.3.3.1 Usage of Account Manager and Permission" for the necessary Permission.

AccountManager User/AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.accountmanager.user"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".UserActivity"
            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>

</manifest>
```

Activity of user application. When tapping the button on the screen, either addcount() or getAuthToken() is to be executed. Authenticator which corresponds to the specific account type may be fake in some cases, so pay attention that the account process is started after verifying that the Authenticator is regular one.

UserActivity.java

```
package org.jssec.android.accountmanager.user;

import java.io.IOException;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.Utils;

import android.accounts.Account;
import android.accounts.AccountManager;
import android.accounts.AccountManagerCallback;
import android.accounts.AccountManagerFuture;
import android.accounts.AuthenticatorDescription;
import android.accounts.AuthenticatorException;
import android.accounts.OperationCanceledException;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class UserActivity extends Activity {

    // Information of the Authenticator to be used
    private static final String JSSEC_ACCOUNT_TYPE = "org.jssec.android.accountmanager";
    private static final String JSSEC_TOKEN_TYPE = "webservice";
    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);

        mLogView = (TextView)findViewById(R.id.logview);
    }

    public void addAccount(View view) {
        logLine();
        logLine("Add a new account");

        // *** POINT 1 *** Execute the account process after verifying if the authenticator is regular one.
        if (!checkAuthenticator()) return;

        AccountManager am = AccountManager.get(this);
        am.addAccount(JSSEC_ACCOUNT_TYPE, JSSEC_TOKEN_TYPE, null, null, this,
```

```
            new AccountManagerCallback<Bundle>() {
                @Override
                public void run(AccountManagerFuture<Bundle> future) {
                    try {
                        Bundle result = future.getResult();
                        String type = result.getString(AccountManager.KEY_ACCOUNT_TYPE);
                        String name = result.getString(AccountManager.KEY_ACCOUNT_NAME);
                        if (type != null && name != null) {
                            logLine("Add the following accounts:");
                            logLine("  Account type: %s", type);
                            logLine("  Account name: %s", name);
                        } else {
                            String code = result.getString(AccountManager.KEY_ERROR_CODE);
                            String msg = result.getString(AccountManager.KEY_ERROR_MESSAGE);
                            logLine("The account cannot be added");
                            logLine("  Error code %s: %s", code, msg);
                        }
                    } catch (OperationCanceledException e) {
                    } catch (AuthenticatorException e) {
                    } catch (IOException e) {
                    }
                }
            },
            null);
    }

    public void getAuthToken(View view) {
        logLine();
        logLine("Get token");

        // *** POINT 1 *** After checking that the Authenticator is the regular one, execute account process.
        if (!checkAuthenticator()) return;

        AccountManager am = AccountManager.get(this);
        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
        if (accounts.length > 0) {
            Account account = accounts[0];
            am.getAuthToken(account, JSSEC_TOKEN_TYPE, null, this,
                    new AccountManagerCallback<Bundle>() {
                        @Override
                        public void run(AccountManagerFuture<Bundle> future) {
                            try {
                                Bundle result = future.getResult();
                                String name = result.getString(AccountManager.KEY_ACCOUNT_NAME);
                                String authtoken = result.getString(AccountManager.KEY_AUTHTOKEN);
                                logLine("%s-san's token:", name);
                                if (authtoken != null) {
                                    logLine("    %s", authtoken);
                                } else {
                                    logLine("    Couldn't get");
                                }
                            } catch (OperationCanceledException e) {
                                logLine("  Exception: %s",e.getClass().getName());
                            } catch (AuthenticatorException e) {
                                logLine("  Exception: %s",e.getClass().getName());
                            } catch (IOException e) {
                                logLine("  Exception: %s",e.getClass().getName());
                            }
                        }
                    }, null);
```



```
        } else {
            logLine("Account is not registered.");
        }
    }

    // *** POINT 1 *** Verify that Authenticator is regular one.
    private boolean checkAuthenticator() {
        AccountManager am = AccountManager.get(this);
        String pkgname = null;
        for (AuthenticatorDescription ad : am.getAuthenticatorTypes()) {
            if (JSSEC_ACCOUNT_TYPE.equals(ad.type)) {
                pkgname = ad.packageName;
                break;
            }
        }

        if (pkgname == null) {
            logLine("Authenticator cannot be found.");
            return false;
        }

        logLine("  Account type: %s", JSSEC_ACCOUNT_TYPE);
        logLine("  Package name of Authenticator: ");
        logLine("    %s", pkgname);

        if (!PkgCert.test(this, pkgname, getTrustedCertificateHash(this))) {
            logLine("  It's not regular Authenticator(certificate is not matched.)");
            return false;
        }

        logLine("  This is regular Authenticator.");
        return true;
    }

    // Certificate hash value of regular Authenticator application
    // Certificate hash value can be checked in sample applciation JSSEC CertHash Checker
    private String getTrustedCertificateHash(Context context) {
        if (Utils.isDebuggable(context)) {
            // Certificate hash value of  debug.keystore "androiddebugkey"
            return "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
        } else {
            // Certificate hash value of  keystore "my company key"
            return "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
        }
    }

    private void log(String str) {
        mLogView.append(str);
    }

    private void logLine(String line) {
        log(line + "¥n");
    }

    private void logLine(String fmt, Object... args) {
        logLine(String.format(fmt, args));
    }

    private void logLine() {
        log("¥n");
```

```
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // Will not handle multiple signatures.
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```


5.3.2. Rule Book

Follow the rules below when implementing Authenticator application.

1.	Service that Provides Authenticator Must Be Private (Required)
2.	Login Screen Activity Must Be Implemented by Authenticator Application (Required)
3.	The Login Screen Activity Must Be Made as a Public Activity and Suppose Attack Accesses by Other Applications (Required)
4.	Provide KEY_INTENT with Explicit Intent with the Specified Class Name of Login Screen Activity (Required)
5.	Sensitive Information (like Account Information and Authentication Token) Must Not Be Output to the Log (Required)
6.	Password Should Not Be Saved in Account Manager (Recommended)
7.	HTTPS Should Be Used for Communication Between an Authenticator and the Online Service (Required)

Follow the rules below when implementing user application.

8.	Account Process Should Be Executed after verifying if the Authenticator is the regular one (Required)

5.3.2.1. Service that Provides Authenticator Must Be Private (Required)

It's presupposed that the Service which provides with Authenticator is used by Account Manager, and it should not be accessed by other applications. So, by making it Private Service, it can exclude accesses by other applications. In addition, Account Manager runs with system privilege, so Account Manager can access even if it's private Service.

5.3.2.2. Login Screen Activity Must Be Implemented by Authenticator Application (Required)

Login screen which is displayed when adding new account and getting authentication token again, should be implemented by Authenticator application. Own Login screen should not be prepared in user application side. As mentioned at the beginning of this article, [The advantage of AccountManager is that the extremely sensitive information/password is not necessarily to be handled by application.], If login screen is prepared in user application side, password is handled by user application, and its design becomes what is beyond the policy of Account Manager.

By preparing login screen by Authenticator application, who can operate login screen is limited only the device's user. It means that there's no way to attack the account for malicious applications by attempting to login directly, or by creating an account.

5.3.2.3. The Login Screen Activity Must Be Made as a Public Activity and Suppose Attack Accesses by Other Applications (Required)

Login screen Activity is the system launched by the user application's priviledge. In order that the login screen Activity is displayed even when the signature keys of user application and Authenticator application are different, login screen Activity should be implemented as Public Activity.

What login screen Activity is public Activity means, that there's a chance that it may be launched by malicious applications. Never trust on any input data. Hence, it's necessary to take the counter-measures mentioned in "3.2 Handling Input Data Carefully and Securely"

5.3.2.4. Provide KEY_INTENT with Explicit Intent with the Specified Class Name of Login Screen Activity (Required)

When Authenticator needs to open login screen Activity, Intent which launches login screen Activity is to be given in the Bundle that is returned to Account Manager, by KEY_INTENT. The Intent to be given, should be the explicit Intent which specifies class name of login screen Activity. In the case of specifying implicit Intent which specifies Action name, there's a possibility that not login screen Activity which Authenticator application prepared by itself, but an Activity which other application prepared, is launched. When a malicious application prepares a login screen which looks like the regular login screen, there's a risk that a user may input password in the fake login screen.

5.3.2.5. Sensitive Information (like Account Information and Authentication Token) Must Not Be Output to the Log (Required)

Applications which access to online service sometimes face a trouble like it cannot access to online service successfully. The causes of unsuccessful access are various, like lack in network environment arrangement, mistakes in implementing communication protocol, lack of Permission, authentication error, etc. A common implementation is that a program outputs the detailed information to log, so that developer can analyze the cause of a problem later.

Sensitive information like password or authentication token should not be output to log. Log information can be read from other applications, so it may become the cause of information leakage. Also, account names should not be output to log, if it could be lead the damage of leakage.

5.3.2.6. Password Should Not Be Saved in Account Manager (Recommended)

Two of authentication information, password and authentication token, can be saved in an account to be register to AccountManager. This information is to be saved in /data/system/accounts.db, in a plain text (i.e. without encryption). To read in the contents of accounts.db, either root privilege or system privilege is required, and it cannot be read from the marketed Android devices. In the case there is any vulnerability in Android OS, which root privilege or system privilege may be taken over by attackers, authentication information which is saved in accounts.db will be on the edge of the risk.

The Authentication application which is introduced in this article, is designed to save authentication token in AccountManager without saving user password. When accessing to online service continuously in a certain period, generally the expiration period of authentication token is extended, so the design that password is not saved is enough in most cases.

In general, valid date of authentication token is shorter than password, and it's characteristic that it can be disabled anytime. In case, authentication token is leaked, it can be disabled, so authentication token is comparatively safer, compared with password. In the case authentication token is disabled, user can input the password again to get a new authentication token.

If disabling password when it's leaked, user cannot use online service any more. In this case, it requires call center support etc, and it will take huge cost. Hence, it's better to avoid from the design to save password in AccountManager. In case, the design to save password cannot be avoided, high level of reverse engineering counter-measures like encrypting password and obfuscating the key of that encryption, should be taken.

5.3.2.7. HTTPS Should Be Used for Communication Between an Authenticator and the Online Service (Required)

Password or authentication token is so called authentication information, and if it's taken over by the third party, the third party can masquerade as the valid user. Since Authenticator sends/receives these types of authentication information with online service, reliable encrypted communication method like an HTTPS should be used.

5.3.2.8. Account Process Should Be Executed after verifying if the Authenticator is the regular one (Required)

In the case there are several Authenticators which the same account type is defined in a device, Authenticator which was installed earlier becomes valid. So, when the own Authenticator was installed later, it's not to be used.

If the Authenticator which was installed earlier, is the malware's masquerade, account information inputted by user may be taken over by malware. User application should verify the account type which performs account operation, whether the regular Authenticator is allocated to it or not, before executing account operation.

Whether the Authenticator which is allocated to one account type is regular one or not, can be verified by checking whether the certificate hash value of the package of Authenticator matches with pre-confirmed valid certificate hash value. If the certificate hash values are found to be not matched, a measure to prompt user to uninstall the package which includes the unexpected Authenticator allocated to that account type, is preferable.

5.3.3. Advanced Topics

5.3.3.1. Usage of Account Manager and Permission

To use each method of AccountManager class, it's necessary to declare to use the appropriate Permission respectively, in application's AndroidManifest.xml. Table 5.3-1 shows correspondence of Permission and methods.

Table 5.3-1 Function of Account Manager and Permission

	Functions that Account Manager provides	
Permission	Method	Explanation
AUTHENTICATE_ACCOUNTS (Only Packages which are signed by the same key of Authenticator, can use.)	getPassword()	To get password
	getUserData()	To get user information
	addAccountExplicitly()	To add accounts to DB
	peekAuthToken()	To get cached token
	setAuthToken()	To register authentication token
	setPassword()	To change password
	setUserData()	To set user information
GET_ACCOUNTS	getAccounts()	To get a list of all accounts
	getAccountsByType()	To get a list of accounts which account types are same
	getAccountsByTypeAndFeatures()	To get a list of accounts which have the specified function
	addOnAccountsUpdatedListener()	To register event listener
	hasFeatures()	Whether it has the specified function or not
MANAGE_ACCOUNTS	getAuthTokenByFeatures()	To get authentication token of the accounts which have the specified function
	addAccount()	To request a user to add accounts
	removeAccount()	To remove an account
	clearPassword()	Initialize password
	updateCredentials()	Request a user to change password
	editProperties()	Change Authenticator setting
	VerifyCredentials()	Request a user to input password again
USE_CREDENTIALS	getAuthToken()	To get authentication token
	blockingGetAuthToken()	To get authentication token
MANAGE_ACCOUNTS	invalidateAuthToken()	To delete cached token

or USE_CREDENTIALS		

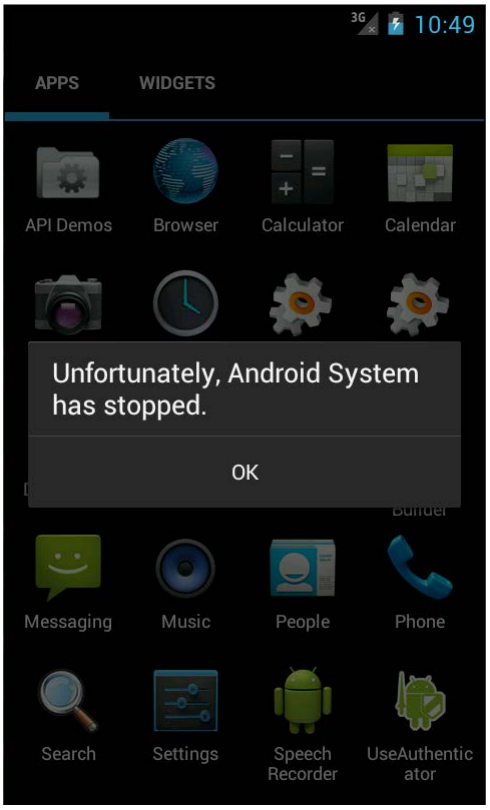
In case using methods group which AUTHENTICATE_ACCOUNTS Permission is necessary, there is a restriction related to package signature key along with Permission. Specifically, the key for signature of package that provides Authenticator and the key for signature of package in the application that uses methods, should be the same. So, when distributing an application which uses method group which AUTHENTICATE_ACCOUNTS Permission is necessary other than Authenticator, signature should be signed by the key which is the same as Authenticator.

In a development phase by Eclipse, since a fixed debug keystore might be shared by some eclipse projects, developers might implement and test Acount Manager by considering only permissions and no signature. It's necessary for especially developers who use the different signature keys per applications, to be very careful when selecting which key to use for applications, considering this restriction. In addition, since the data which is obtained by AccountManager includes the sensitive information, so need to handle with care in order to decrease the risk like leakage or unauthorized use.

5.3.3.2. Exception Occurs When Signature Keys of User Application and Authenticator Application Are Different, in Android 4.0.x

When authentication token acquisition function, is required by the user application which is signed by the developer key which is different from the signature key of Authenticator application that includes Authenticator, AccountManager verifies users whether to grant the usage of authentication token or not, by displaying the authentication token license screen (GrantCredentialsPermissionActivity.) However, there's a bug in Android Framework of Android 4.0.x, as soon as this screen in opened by AccountManager, exception occurs, and application is force closed. (Figure 5.3-3). See https://code.google.com/p/android/issues/detail?id=23421 for the details of the bug. This bug cannot be found in Android 2.3.x and earlier, and Android 4.1.x. and later.

Android 4.0.x



Android 4.1.x

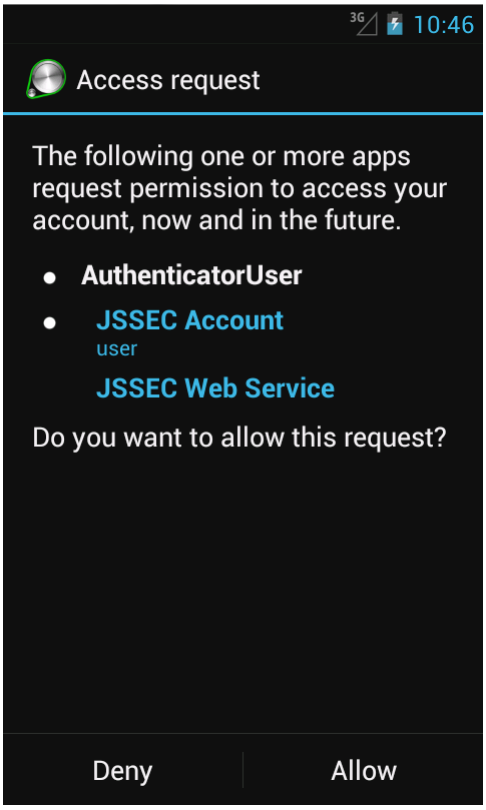


Figure 5.3-3 When displaying Android standard authentication token license screen.

5.4. Communicating via HTTPS

Most of smartphone applications communicate with Web servers on the Internet. As methods of communications, here we focus on the 2 methods of HTTP and HTTPS. From the security point of view, HTTPS communication is preferable. Lately, major Web services like Google or Facebook have been coming to use HTTPS as default.

In 2012, many defects in implementation of HTTPS communication were pointed out in Android applications. These defects might have been implemented for accessing testing Web servers operated by server certificates that are not issued by trusted third party certificate authorities, but issued privately (hereinafter, called private certificates).

In this section, communication methods of HTTP and HTTPS are explained and the method to access safely with HTTPS to a Web server operated by a private certificate is also described.

5.4.1. Sample Code

You can find out which type of HTTP/HTTPS communication you are supposed to implement through the following chart (Figure 5.4-1) shown below.

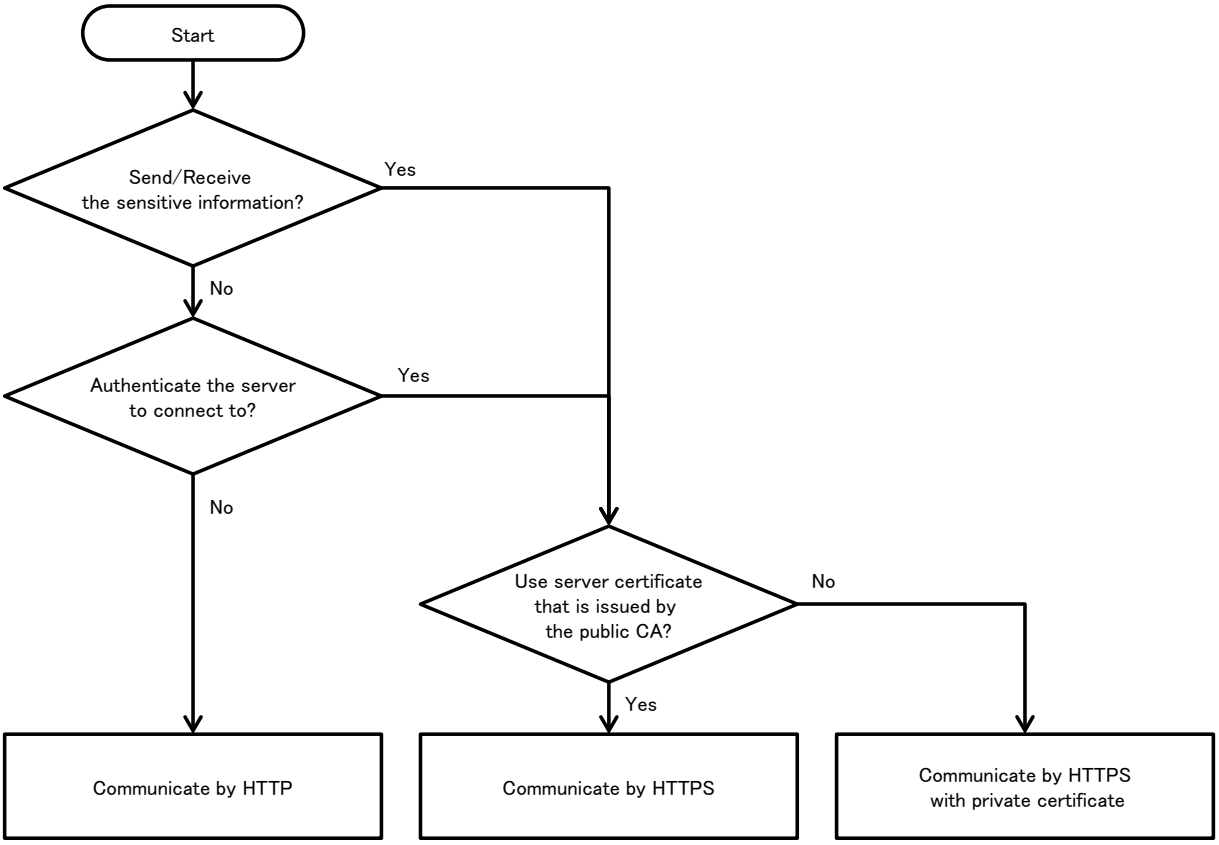


Figure 5.4-1 Flow Figure to select sample code of HTTP/HTTPS

When sensitive information is sent or received, HTTPS communication is to be used because its communication channel is encrypted with SSL/TLS. HTTPS communication is required for the

following sensitive information.

- Login ID/Password for Web services.
- Information for keeping authentication state (session ID, token, Cookie etc.)
- Important/confidential information depending on Web services (personal information, credit card information etc.)

A smartphone application with network communication is a part of "system" as well as a Web server. And you have to select HTTP or HTTPS for each communication based on secure design and coding considering the whole "system". Table 5.4-1 is for a comparison between HTTP and HTTPS. And Table 5.4-2 is for the differences in sample codes.

Table 5.4-1 Comparison between HTTP communication method and HTTPS communication method

		HTTP	HTTPS
Characteristics	URL Encrypting contents Tampering detection of contents Authenticating a server	Starting with http:// Not available Impossible Impossible	Starting with https:// Available Possible Possible
Damage Risk	Reading contents by attackers Modifying contents by attackers Application's access to a fake server	High High High	Low Low Low

Table 5.4-2 Explanation of HTTP/HTTPS communication Sample code

Sample code	Communi-cation	Sending/Receiving sensitive information	Server certificate
Communicating via HTTP	HTTP	Not applicable	-
Communicating via HTTPS	HTTPS	OK	Server certificates issued by trusted third party's certificate authorities like Cybertrust and VeriSign etc.
Communicating via HTTPS with private certificate	HTTPS	OK	Private certificate * Operation mode which can be often seen in intra server or in test server.

There are two major APIs for HTTP/HTTPS communications supported by Android. In this article and sample codes, Apache HttpClient library is used because applications can control communication way in detail.

- java.net.HttpURLConnection/javax.net.ssl.HttpsURLConnection comes from JavaSDK
- Apache HttpClient library comes from Apache HttpComponent

5.4.1.1. Communicating via HTTP

It is based on two premises that all contents sent/received through HTTP communications may be sniffed and tampered by attackers and your destination server may be replaced with fake servers prepared by attackers. HTTP communication can be used only if no damage is caused or the damage is within the permissible extent even under the premises. If an application cannot accept the premises, please refer to "5.4.1.2 Communicating via HTTPS" and "5.4.1.3 Communicating via HTTPS with private certificate."

The following sample code shows an application which performs an image search on a Web server, gets the result image and shows it. HTTP communication with the server is performed twice a search. The first communication is for searching image data and the second is for getting it. The worker thread for communication process using AsyncTask is created to avoid the communications performing on the UI thread. Contents sent/received in the communications with the server are not considered as sensitive (e.g. the character string for searching, the URL of the image, or the image data) here. So, the received data such as the URL of the image and the image data may be provided by attackers. To show the sample code simply, any countermeasures are not taken in the sample code by considering the received attacking data as tolerable. Also, the handlings for possible exceptions during JSON purse or showing image data are omitted. It is necessary to handle the exceptions properly depending on the application specs.

Points:

1. Sensitive information must not be contained in send data.
2. Suppose that received data may be sent from attackers.

HttpImageSearch.java

```
package org.jssec.android.https.imagesearch;

import org.apache.http.HttpException;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

import android.net.Uri;
import android.os.AsyncTask;

public abstract class HttpImageSearch extends AsyncTask<String, Void, Object> {

    @Override
    protected Object doInBackground(String... params) {

        // Since use HttpClient for 2 times of GET requests, it shutdown at finally block.
        DefaultHttpClient client = new DefaultHttpClient();

        try {
            // ------------------------------------------------------
            // Communication 1st time: Execute image search
            // ------------------------------------------------------
```

```
            // *** POINT 1 *** Sensitive information must not be contained in send data.
            // Send image search character string
            String search_url = Uri
                    .parse("http://ajax.googleapis.com/ajax/services/search/images?v=1.0")
                    .buildUpon()
                    .appendQueryParameter("q", params[0])
                    .build().toString();
            HttpGet request = new HttpGet(search_url);
            HttpResponse response = client.execute(request);
            checkResponse(response);

            // *** POINT 2 *** Suppose that received data may be sent from attackers.
            // This is sample, so omit the process in case of the searching result is the data from an attacker.
            // This is sample, so omit the exception process in case of JSON purse.
            String result_json = EntityUtils.toString(response.getEntity(), "UTF-8");
            String image_url = new JSONObject(result_json).getJSONObject("responseData")
                    .getJSONArray("results").getJSONObject(0).getString("url");

            // -----------------------------------------------------
            // Communication 2nd time: Get images
            // -----------------------------------------------------

            // *** POINT 1 *** Sensitive information must not be contained in send data.
            request = new HttpGet(image_url);
            response = client.execute(request);
            checkResponse(response);

            // *** POINT 2 *** Suppose that received data may be sent from attackers.
            return EntityUtils.toByteArray(response.getEntity());
        } catch(Exception e) {
            return e;
        } finally {
            // Shutdown HttpClient without fail
            client.getConnectionManager().shutdown();
        }
    }

    private void checkResponse(HttpResponse response) throws HttpException {
        int statusCode = response.getStatusLine().getStatusCode();
        if (HttpStatus.SC_OK != statusCode) {
            throw new HttpException("HttpStatus: " + statusCode);
        }
    }
}
```

ImageSearchActivity.java

```
package org.jssec.android.https.imagesearch;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
```



```
public class ImageSearchActivity extends Activity {

    private EditText mQueryBox;
    private TextView mMsgBox;
    private ImageView mImgBox;
    private AsyncTask<String, Void, Object> mAsyncTask ;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mQueryBox = (EditText)findViewById(R.id.querybox);
        mMsgBox = (TextView)findViewById(R.id.msgbox);
        mImgBox = (ImageView)findViewById(R.id.imageview);
    }

    @Override
    protected void onPause() {
        // After this, Activity may be deleted, so cancel the asynchronization process in advance.
        if (mAsyncTask != null) mAsyncTask.cancel(true);
        super.onPause();
    }

    public void onHttpSearchClick(View view) {
        String query = mQueryBox.getText().toString();
        mMsgBox.setText("HTTP:" + query);
        mImgBox.setImageBitmap(null);

        // Cancel, since the last asynchronous process might not have been finished yet.
        if (mAsyncTask != null) mAsyncTask.cancel(true);

        // Since cannot communicate by UI thread, communicate by worker thread by AsynchTask.
        mAsyncTask = new HttpImageSearch() {
            @Override
            protected void onPostExecute(Object result) {
                // Process the communication result by UI thread.
                if (result instanceof Exception) {
                    Exception e = (Exception)result;
                    mMsgBox.append("¥nException occurs¥n" + e.toString());
                } else {
                    // Exception process when image display is omitted here, since it's sample.
                    byte[] data = (byte[])result;
                    Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
                    mImgBox.setImageBitmap(bmp);
                }
            }
        }.execute(query);   // pass search character string and start asynchronous process
    }

    public void onHttpsSearchClick(View view) {
        String query = mQueryBox.getText().toString();
        mMsgBox.setText("HTTPS:" + query);
        mImgBox.setImageBitmap(null);

        // Cancel, since the last asynchronous process might not have been finished yet.
        if (mAsyncTask != null) mAsyncTask.cancel(true);

        // Since cannot communicate by UI thread, communicate by worker thread by AsynchTask.
```

```
        mAsyncTask = new HttpsImageSearch() {
            @Override
            protected void onPostExecute(Object result) {
                // Process the communication result by UI thread.
                if (result instanceof Exception) {
                    Exception e = (Exception)result;
                    mMsgBox.append("¥nException occurs¥n" + e.toString());
                } else {
                    byte[] data = (byte[])result;
                    Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
                    mImgBox.setImageBitmap(bmp);
                }
            }
        }.execute(query);   // pass search character string and start asynchronous process
    }
}
```

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.https.imagesearch"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ImageSearchActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Light" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```


5.4.1.2. Communicating via HTTPS

Transmitted and received data with HTTPS are encrypted. In addition HTTPS checks whether a connected server is trusted or not. To authenticate the server, Android HTTPS library verifies "server certificate" which is transmitted from the server in the handshake phase of HTTPS transaction with following points:

- The server certificate is signed by a trusted third party certificate authority
- The period and other properties of the server certificate are valid
- CN in Subject of the server certificate equals to the host name of the serve.

When an error is encountered during the verification above, a server certificate verification exception (SSLException) is thrown. The error occurs due to any defects in the server certificate or man-in-the-middle attacks by attackers. You have to handle the exception with an appropriate sequence based on the application specifications.

The next a sample code is for HTTPS communication which connects to a Web server with a server certificate issued by a trusted third party certificate authority. For HTTPS communication with a server certificate issued privately, please refer to "5.4.1.3 Communicating via HTTPS with private certificate."

The following sample code shows an application which performs an image search on a Web server, gets the result image and shows it. HTTPS communication with the server is performed twice a search. The first communication is for searching image data and the second is for getting it. The worker thread for communication process using AsyncTask is created to avoid the communications performing on the UI thread. All contents sent/received in the communications with the server are considered as sensitive (e.g. the character string for searching, the URL of the image, or the image data) here. To show the sample code simply, no special handling for SSLException is performed. It is necessary to handle the exceptions properly depending on the application specifications.

Points:

1. URI starts with https://.
2. Sensitive information may be contained in send data.
3. Received data can be trusted as same as the server.
4. SSLException should be handled with an appropriate sequence in an application.

HttpsImageSearch.java

```
package org.jssec.android.https.imagesearch;

import javax.net.ssl.SSLException;

import org.apache.http.HttpException;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
```

```
import org.json.JSONObject;

import android.net.Uri;
import android.os.AsyncTask;

public abstract class HttpsImageSearch extends AsyncTask<String, Void, Object> {

    @Override
    protected Object doInBackground(String... params) {

        // Since HttpClient is used for 2 times of GET requests,it shutdown at finally block.
        DefaultHttpClient client = new DefaultHttpClient();

        try {
            // ----------------------------------------------------------
            // Communication 1st time : Execute image search
            // ----------------------------------------------------------

            // *** POINT 1 *** URI starts with https://.
            // *** POINT 2 *** Sensitive information may be contained in send data.
            String search_url = Uri
                    .parse("https://ajax.googleapis.com/ajax/services/search/images?v=1.0")
                    .buildUpon()
                    .appendQueryParameter("q", params[0])
                    .build().toString();
            HttpGet request = new HttpGet(search_url);
            HttpResponse response = client.execute(request);
            checkResponse(response);

            // *** POINT 3 *** Received data can be trusted as same as the server.
            String result_json = EntityUtils.toString(response.getEntity(), "UTF-8");
            String image_url = new JSONObject(result_json).getJSONObject("responseData")
                    .getJSONArray("results").getJSONObject(0).getString("url");

            // ----------------------------------------------------------
            // Communication 2nd time : Get image
            // ----------------------------------------------------------

            // *** POINT 1 *** URI starts with https://.
            // *** POINT 2 *** Sensitive information may be contained in send data.
            request = new HttpGet(image_url);
            response = client.execute(request);
            checkResponse(response);

            // *** POINT 3 *** Received data can be trusted as same as the server.
            return EntityUtils.toByteArray(response.getEntity());
        } catch(SSLException e) {
            // *** POINT 4 *** SSLException should be handled with an appropriate sequence in an application.
            // Omit exception process, since it's sample
            return e;
        } catch(Exception e) {
            return e;
        } finally {
            // Shutdown HttpClient without fail.
            client.getConnectionManager().shutdown();
        }
    }

    private void checkResponse(HttpResponse response) throws HttpException {
        int statusCode = response.getStatusLine().getStatusCode();
```



```
        if (HttpStatus.SC_OK != statusCode) {
            throw new HttpException("HttpStatus: " + statusCode);
        }
    }
}
```

Other sample code files are the same as "5.4.1.1 Communicating via HTTP," so please refer to "5.4.1.1 Communicating via HTTP."

5.4.1.3. Communicating via HTTPS with private certificate

This section shows a sample code of HTTPS communication with a server certificate issued privately (private certificate), but not with that issued by a trusted third party authority. Please refer to "5.4.3.1 How to Create Private Certificate and Configure Server Settings" for creating a root certificate of a private certificate authority and private certificates and setting HTTPS settings in a Web server. The sample program has a cacert.crt file in assets. It is a root certificate file of private certificate authority.

The following sample code shows an application which gets an image on a Web server and shows it. HTTPS is used for the communication with the server. The worker thread for communication process using AsyncTask is created to avoid the communications performing on the UI thread. All contents (the URL of the image and the image data) sent/received in the communications with the server are considered as sensitive here. To show the sample code simply, no special handling for SSLException is performed. It is necessary to handle the exceptions properly depending on the application specifications.

Points:

1. Verify a server certificate with the root certificate of a private certificate authority.
2. URI starts with https://.
3. Sensitive information may be contained in send data.
4. Received data can be trusted as same as the server.
5. SSLException should be handled with an appropriate sequence in an application.

PrivateCertificathettpsGet.java

```
package org.jssec.android.https.privatecertificate;

import java.security.KeyStore;

import javax.net.ssl.SSLException;

import org.apache.http.HttpException;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

import android.content.Context;
import android.os.AsyncTask;

public abstract class PrivateCertificateHttpsGet extends AsyncTask<String, Void, Object> {

    private Context mContext;

    public PrivateCertificateHttpsGet(Context context) {
        mContext = context;
    }
```



```
    @Override
    protected Object doInBackground(String... params) {

        DefaultHttpClient client = new DefaultHttpClient();

        try {
            // *** POINT 1 *** Verify a server certificate with the root certificate of a private certificate authorit
y.
            // Set keystore which includes only private certificate that is stored in assets, to client.
            KeyStore ks = KeyStoreUtil.getEmptyKeyStore();
            KeyStoreUtil.loadX509Certificate(ks,
                    mContext.getResources().getAssets().open("cacert.crt"));
            Scheme sch = new Scheme("https", new SSLSocketFactory(ks), 443);
            client.getConnectionManager().getSchemeRegistry().register(sch);

            // *** POINT 2 *** URI starts with https://.
            // *** POINT 3 *** Sensitive information may be contained in send data.
            HttpGet request = new HttpGet(params[0]);
            HttpResponse response = client.execute(request);
            checkResponse(response);

            // *** POINT 4 *** Received data can be trusted as same as the server.
            return EntityUtils.toByteArray(response.getEntity());
        } catch(SSLException e) {
            // *** POINT 5 *** SSLException should be handled with an appropriate sequence in an application.
            // Exception process is omitted here since it's sample.
            return e;
        } catch(Exception e) {
            return e;
        } finally {
            // Shutdown HttpClient without fail.
            client.getConnectionManager().shutdown();
        }
    }

    private void checkResponse(HttpResponse response) throws HttpException {
        int statusCode = response.getStatusLine().getStatusCode();
        if (HttpStatus.SC_OK != statusCode) {
            throw new HttpException("HttpStatus: " + statusCode);
        }
    }
}
```

KeyStoreUtil.java

```
package org.jssec.android.https.privatecertificate;

import java.io.IOException;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Enumeration;

public class KeyStoreUtil {
```

```
    public static KeyStore getEmptyKeyStore() throws KeyStoreException,
            NoSuchAlgorithmException, CertificateException, IOException {
        KeyStore ks = KeyStore.getInstance("BKS");
        ks.load(null);
        return ks;
    }

    public static void loadAndroidCAStore(KeyStore ks)
            throws KeyStoreException, NoSuchAlgorithmException,
            CertificateException, IOException {
        KeyStore aks = KeyStore.getInstance("AndroidCAStore");
        aks.load(null);
        Enumeration<String> aliases = aks.aliases();
        while (aliases.hasMoreElements()) {
            String alias = aliases.nextElement();
            Certificate cert = aks.getCertificate(alias);
            ks.setCertificateEntry(alias, cert);
        }
    }

    public static void loadX509Certificate(KeyStore ks, InputStream is)
            throws CertificateException, KeyStoreException {
        try {
            CertificateFactory factory = CertificateFactory.getInstance("X509");
            X509Certificate x509 = (X509Certificate)factory.generateCertificate(is);
            String alias = x509.getSubjectDN().getName();
            ks.setCertificateEntry(alias, x509);
        } finally {
            try { is.close(); } catch (IOException e) { }
        }
    }
}
```

PrivateCertificathettpsActivity.java

```
package org.jssec.android.https.privatecertificate;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

public class PrivateCertificateHttpsActivity extends Activity {

    private EditText mUrlBox;
    private TextView mMsgBox;
    private ImageView mImgBox;
    private AsyncTask<String, Void, Object> mAsyncTask ;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```



```
        mUrlBox = (EditText)findViewById(R.id.urlbox);
        mMsgBox = (TextView)findViewById(R.id.msgbox);
        mImgBox = (ImageView)findViewById(R.id.imageview);
    }

    @Override
    protected void onPause() {
        // After this, Activity may be discarded, so cancel asynchronous process in advance.
        if (mAsyncTask != null) mAsyncTask.cancel(true);
        super.onPause();
    }

    public void onClick(View view) {
        String url = mUrlBox.getText().toString();
        mMsgBox.setText(url);
        mImgBox.setImageBitmap(null);

        // Cancel, since the last asynchronous process might have not been finished yet.
        if (mAsyncTask != null) mAsyncTask.cancel(true);

        // Since cannot communicate through UI thread, communicate by worker thread by AsynchTask.
        mAsyncTask = new PrivateCertificateHttpsGet(this) {
            @Override
            protected void onPostExecute(Object result) {
                // Process the communication result through UI thread.
                if (result instanceof Exception) {
                    Exception e = (Exception)result;
                    mMsgBox.append("¥nException occurs¥n" + e.toString());
                } else {
                    byte[] data = (byte[])result;
                    Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
                    mImgBox.setImageBitmap(bmp);
                }
            }
        }.execute(url); // Pass URL and start asynchronization process
    }
}
```

5.4.2. Rule Book

Follow the rules below to communicate with HTTP/HTTPS.

1.	Sensitive Information Must Be Sent/Received over HTTPS Communication (Required)
2.	Received Data over HTTP Must be Handled Carefully and Securely (Required)
3.	SSLException Must Be Handled Appropriately like Notification to User (Required)
4.	TrustManager Must Not Be Changed and Custom TrustManager Must Not Be Created (Required)
5.	HostnameVerifier Must Not Be Changed and Custom HostnameVerifier Must Not Be Created (Required)

5.4.2.1. Sensitive Information Must Be Sent/Received over HTTPS Communication (Required)

In HTTP transaction, sent and received information might be sniffed or tampered and the connected server might be masqueraded. Sensitive information must be sent/ received by HTTPS communication.

5.4.2.2. Received Data over HTTP Must be Handled Carefully and Securely (Required)

Received data in HTTP communications might be generated by attackers for exploiting vulnerability of an application. So you have to suppose that the application receives any values and formats of data and then carefully implement data handlings for processing received data so as not to put any vulnerabilities in. Please refer to "3.2 Handling Input Data Carefully and Securely"

5.4.2.3. SSLException Must Be Handled Appropriately like Notification to User (Required)

In HTTPS communication, SSLException occurs as a verification error when a server certificate is not valid or the communication is under the man-in-the-middle attack. So you have to implement an appropriate exception handling for SSLException. Notifying the user of the communication failure, logging the failure and so on can be considered as typical implementations of exception handling. On the other hand, no special notice to the user might be required in some case. Like this, because how to handle SSLException depends on the application specs and characteristics you need to determine it after first considering thoroughly.

As mentioned above, the application may be attacked by man-in-the-middle attack when SSLException occurs, so it must not be implemented like trying to send/receive sensitive information again via non secure protocol such as HTTP.

5.4.2.4. TrustManager Must Not Be Changed and Custom TrustManager Must Not Be Created (Required)

Just Changing KeyStore which is used for verifying server certificates is enough to communicate via HTTPS with a private certificate like self-signed certificate. However, as explained in "5.4.3.3 Risky

Code that Disables Certificate Verification," there are so many dangerous TrustManager implementations as sample codes for such purpose on the Internet. An Application implemented by referring to these sample codes may have the vulnerability.

When you need to communicate via HTTPS with a private certificate, refer to the secure sample code in "5.4.1.3 Communicating via HTTPS with private certificate."

Of course, custom TrustManager can be implemented securely, but enough knowledge for encryption processing and encryption communication is required so as not to implement vulnerable codes. So this rule dare be (Required).

5.4.2.5. HostnameVerifier Must Not Be Changed and Custom HostnameVerifier Must Not Be Created (Required)

Just Changing KeyStore which is used for verifying server certificates is enough to communicate via HTTPS with a private certificate like self-signed certificate. However, as explained in "5.4.3.3 Risky Code that Disables Certificate Verification," there are so many dangerous HostnameVerifier implementations as sample codes for such purpose on the Internet. An Application implemented by referring to these sample codes may have the vulnerability.

When you need to communicate via HTTPS with a private certificate, refer to the secure sample code in "5.4.1.3 Communicating via HTTPS with private certificate."

Of course, custom HostnameVerifier can be implemented securely, but enough knowledge for encryption processing and encryption communication is required so as not to implement vulnerable codes. So this rule dare be (Required).

5.4.3. Advanced Topics

5.4.3.1. How to Create Private Certificate and Configure Server Settings

In this section, how to create a private certificate and configure server settings in Linux such as Ubuntu and CentOS is described. Private certificate means a server certificate which is issued privately and is told from server certificates issued by trusted third party certificate authorities like Cybertrust and VeriSign.

Create private certificate authority

First of all, you need to create a private certificate authority to issue a private certificate. Private certificate authority means a certificate authority which is created privately as well as private certificate. You can issue plural private certificates by using the single private certificate authority. PC in which the private certificate authority is stored should be limited strictly to be accessed just by trusted persons.

To create a private certificate authority, you have to create two files such as the following shell script newca.sh and the setting file openssl.cnf and then execute them. In the shell script, CASTART and CAEND stand for the valid period of certificate authority and CASUBJ stands for the name of certificate authority. So these values need to be changed according to a certificate authority you create. While executing the shell script, the password for accessing the certificate authority is asked for 3 times in total, so you need to input it every time.

newca.sh – Shell Script to create certificate authority

```
#!/bin/bash

umask 0077

CONFIG=openssl.cnf
CATOP=./CA
CAKEY=cakey.pem
CAREQ=careq.pem
CACERT=cacert.pem
CAX509=cacert.crt
CASTART=1301010000000Z   # 2013/01/01 00:00:00 GMT
CAEND=2301010000000Z     # 2023/01/01 00:00:00 GMT
CASUBJ="/CN=JSSEC Private CA/O=JSSEC/ST=Tokyo/C=JP"

mkdir -p ${CATOP}
mkdir -p ${CATOP}/certs
mkdir -p ${CATOP}/crl
mkdir -p ${CATOP}/newcerts
mkdir -p ${CATOP}/private
touch ${CATOP}/index.txt

openssl req -new -newkey rsa:2048 -sha256 -subj "${CASUBJ}" ¥
       -keyout ${CATOP}/private/${CAKEY} -out ${CATOP}/${CAREQ}
openssl ca -selfsign -md sha256 -create_serial -batch ¥
       -keyfile ${CATOP}/private/${CAKEY} ¥
       -startdate ${CASTART} -enddate ${CAEND} -extensions v3_ca ¥
```



```
       -in ${CATOP}/${CAREQ} -out ${CATOP}/${CACERT} ¥
       -config ${CONFIG}
openssl x509 -in ${CATOP}/${CACERT} -outform DER -out ${CATOP}/${CAX509}
```

openssl.cnf – Setting file of openssl command which 2 shell scripts refers in common.

```
[ ca ]
default_ca     = CA_default           # The default ca section

[ CA_default ]
dir            = ./CA                # Where everything is kept
certs          = $dir/certs           # Where the issued certs are kept
crl_dir        = $dir/crl             # Where the issued crl are kept
database       = $dir/index.txt       # database index file.
#Proprietary-defined  _subject = no                  # Set to 'no' to allow creation of
                                    # several ctificates with same subject.
new_certs_dir   = $dir/newcerts        # default place for new certs.
certificate    = $dir/cacert.pem      # The CA certificate
serial         = $dir/serial          # The current serial number
crlnumber      = $dir/crlnumber       # the current crl number
                                    # must be commented out to leave a V1 CRL
crl            = $dir/crl.pem         # The current CRL
private_key    = $dir/private/cakey.pem# The private key
RANDFILE       = $dir/private/.rand    # private random number file
x509_extensions = usr_cert             # The extentions to add the cert
name_opt       = ca_default           # Subject Name options
cert_opt       = ca_default           # Certificate field options
policy         = policy_match

[ policy_match ]
countryName            = match
stateOrProvinceName    = match
organizationName       = supplied
organizationalUnitName  = optional
commonName             = supplied
emailAddress           = optional

[ usr_cert ]
basicConstraints=CA:FALSE
nsComment                      = "OpenSSL Generated Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = CA:true
```

After executing the above shall script, a directory named CA is created just under the work directory. This CA directory is just a private certificate authority. CA/cacert.crt file is the root certificate of the private certificate authority. And it's stored in assets directory of an application as described in "5.4.1.3 Communicating via HTTPS with private certificate," or it's installed in Android device as described in "5.4.3.2 Install Root Certificate of Private Certificate Authority to Android OS's Certification Store (Android 4.0 and later)."

Create private certificate

To create a private certificate, you have to create a shell script like the following newca.sh and execute it. In the shell script, SVSTART and SVEND stand for the valid period of private certificate, and SVSUBJ stands for the name of Web server, so these values need to be changed according to the target Web server. Especially, you need to make sure not to set a wrong host name to /CN of SVSUBJ with which the host name of Web server is to be specified. While executing the shell script, the password for accessing the certificate authority is asked, so you need to input the password which you have set when creating the private certificate authority. After that, y/n is asked 2 times in total and you need to input y every time.

newsv.sh – Shell script which issues private certificate

```
#!/bin/bash

umask 0077

CONFIG=openssl.cnf
CATOP=./CA
CAKEY=cakey.pem
CACERT=cacert.pem
SVKEY=svkey.pem
SVREQ=svreq.pem
SVCERT=svcert.pem
SVX509=svcert.crt
SVSTART=130101000000Z   # 2013/01/01 00:00:00 GMT
SVEND=230101000000Z     # 2023/01/01 00:00:00 GMT
SVSUBJ="/CN=selfsigned.jssec.org/O=JSSEC Secure Cofing Group/ST=Tokyo/C=JP"

openssl genrsa -out ${SVKEY} 2048
openssl req -new -key ${SVKEY} -subj "${SVSUBJ}" -out ${SVREQ}
openssl ca -md sha256 ¥
       -keyfile ${CATOP}/private/${CAKEY} -cert ${CATOP}/${CACERT} ¥
       -startdate ${SVSTART} -enddate ${SVEND} ¥
       -in ${SVREQ} -out ${SVCERT} -config ${CONFIG}
openssl x509 -in ${SVCERT} -outform DER -out ${SVX509}
```

After executing the above shell script, both svkey.pem (private key file) and svcert.pem (private certificate file) for Web server are generated just under work directory.

When Web server is Apache, you will specify prikey.pem and cert.pem in the confituration file as follows.

```
SSLCertificateFile    "/path/to/svcert.pem"
SSLCertificateKeyFile "/path/to/svkey.pem"
```


5.4.3.2. Install Root Certificate of Private Certificate Authority to Android OS's Certification Store (Android 4.0 and later)

In the sample code of "5.4.1.3 Communicating via HTTPS with private certificate," the method to establish HTTPS sessions to a Web server from one application using a private certificate by installing the root certificate into the application is introduced. In this section, the method to establish HTTPS sessions to Web servers from all applications using private certificates by installing the root certificate into Android OS is to be introduced. Note that all you install should be certificates issued by trusted certificate authorities including your own certificate authorities. This step can be applied to just devices with Android 4.0 or later.

First of all, you need to copy the root certificate file "cacert.crt" to the internal storage of an Android device. You can also get the root certificate file used in the sample code from https://selfsigned.jssec.org/cacert.crt.

And then, you will open Security page from Android Settings and you can install the root certificate in an Android device by doing as follows.

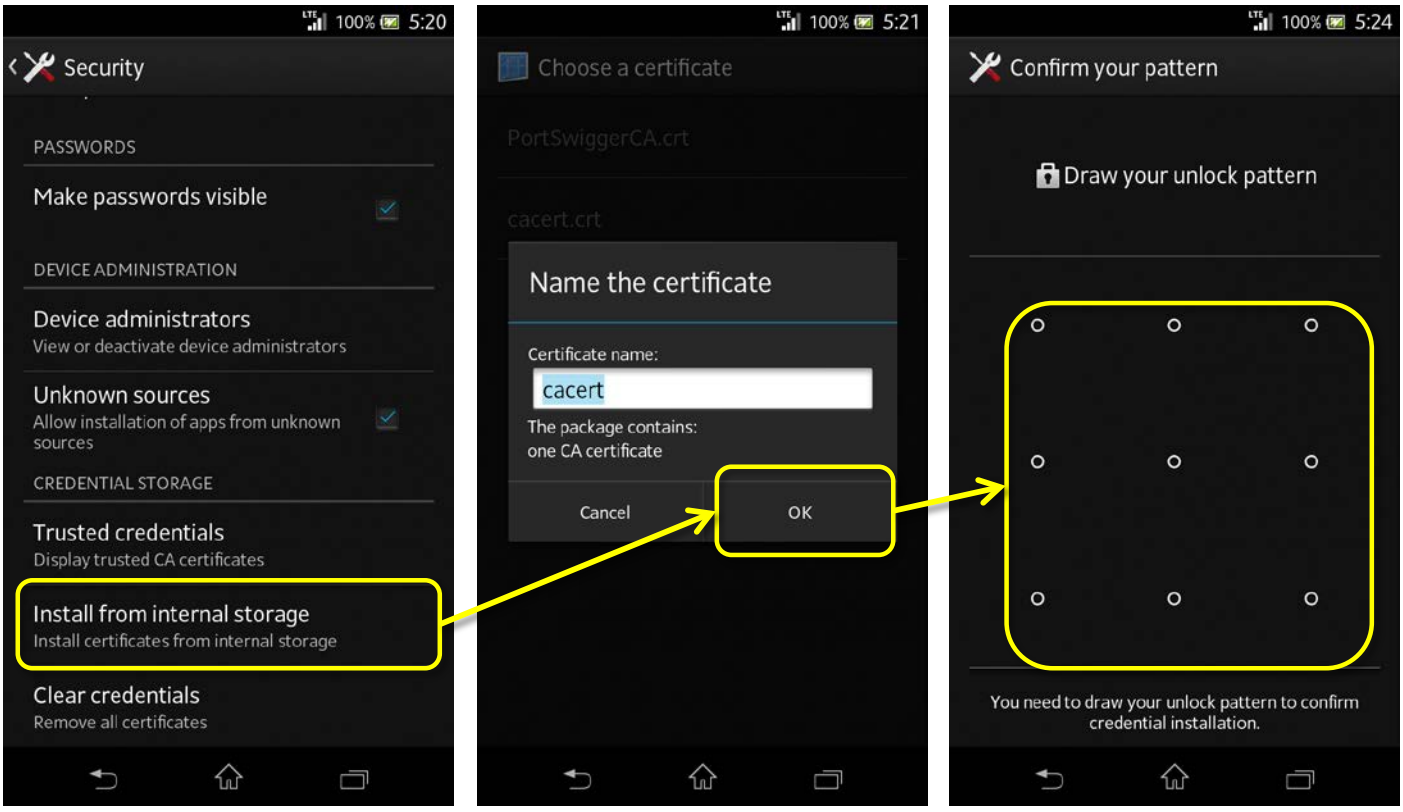


Figure 5.4-2 Steps to install root certificate of private certificate authority

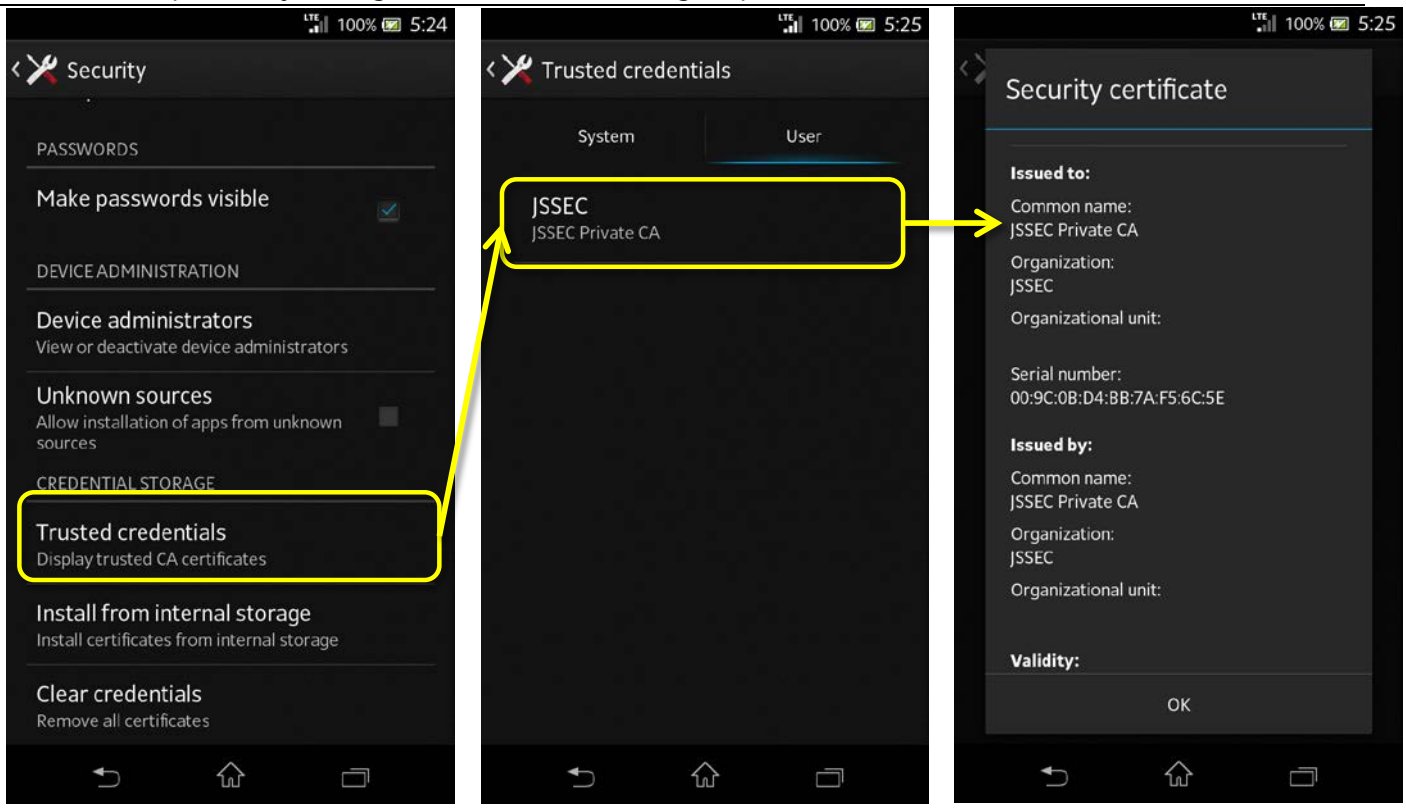


Figure 5.4-3 Checking if root certificate is installed or not

Once the root certificate is installed in Android OS, all applications can correctly verify every private certificate issued by the certificate authority. The following figure shows an example when displaying https://selfsigned.jssec.org/droid_knight.png in Chrome browser.

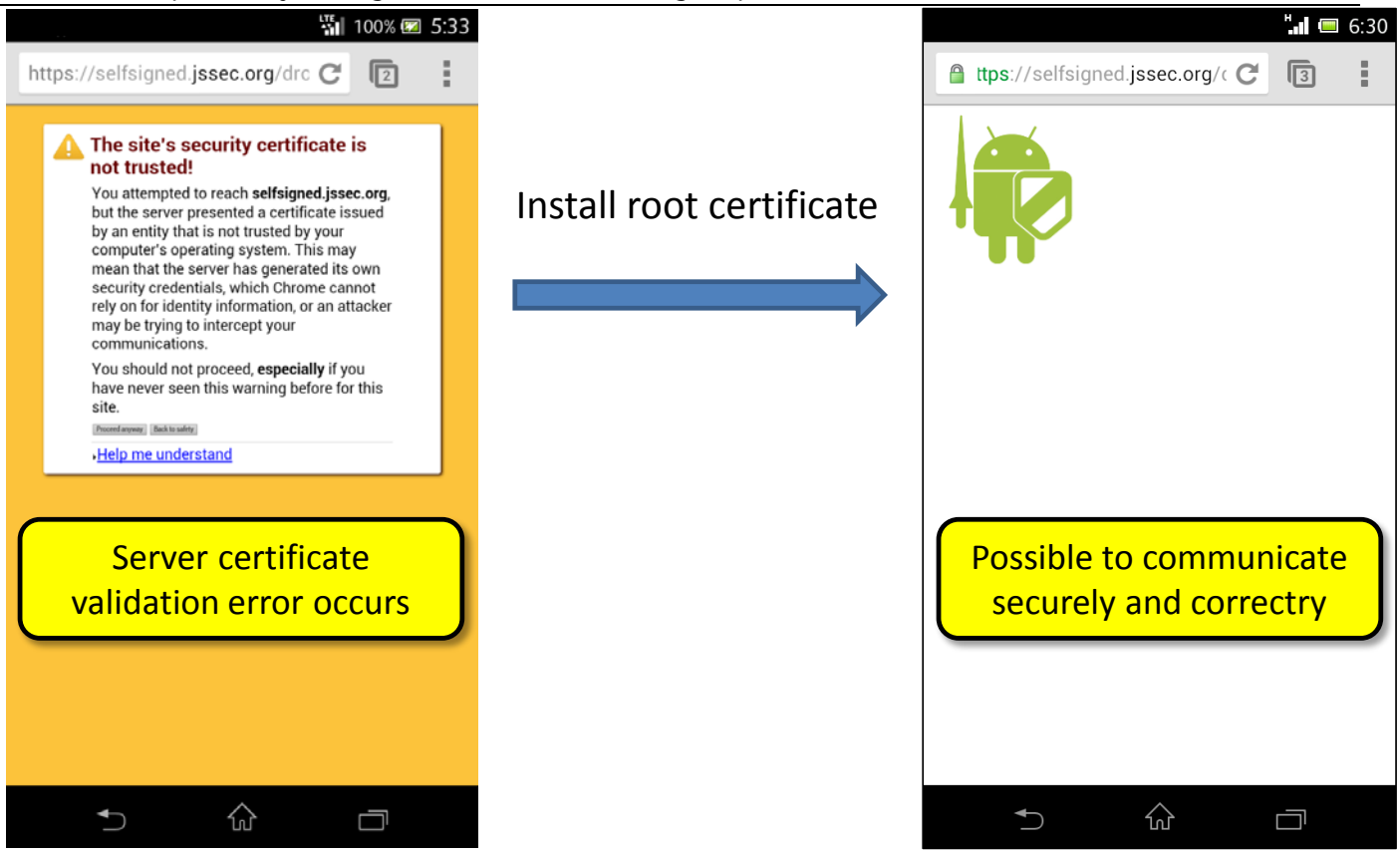


Figure 5.4-4 Once root certificate installed, private certificates can be verified correctly.

By installing the root certificate this way, even applications using the sample code "5.4.1.2 Communicating via HTTPS" can correctly connect via HTTPS to a Web server which is operated with a private certificate.

5.4.3.3. Risky Code that Disables Certificate Verification

A lot of incorrect samples (code snipets), which allow applications to continue to communicate via HTTPS with Web servers even after certificate verification errors occur, are found on the Internet. Since they are introduced as the way to communicate via HTTPS with a Web server using a private certificate, there have been so many applications created by developers who have used those sample codes by copy and paste. Unfortunately, most of them are vulnerable to man-in-the-middle attack. As mentioned in the top of this article, "In 2012, many defects in implementation of HTTPS communication were pointed out in Android applications", many Android applications which would have implemented such vulnerable codes have been reported.

Several code snipets to cause vulnerable HTTPS communication are shown below. When you find this type of code snipets, it's highly recommended to replace the sample code of "5.4.1.3 Communicating via HTTPS with private certificate."

Risk:Case which creates empty TrustManager

```
TrustManager tm = new X509TrustManager() {

    @Override
    public void checkClientTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
        // Do nothing -> accept any certificates
    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
        // Do nothing -> accept any certificates
    }

    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return null;
    }
};
```

Risk:Case which creates empty HostnameVerifier

```
HostnameVerifier hv = new HostnameVerifier() {
    @Override
    public boolean verify(String hostname, SSLSession session) {
        // Always return true -> Accespt any host names
        return true;
    }
};
```

Risk:Case that ALLOW_ALL_HOSTNAME_VERIFIER is used.

```
SSLSocketFactory sf;
...
sf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
```


6. Difficult Problems

In Android, there are some problems that it is difficult to assure a security by application implementation due to a specification of Android OS or a function which Android OS provides. By being abused by the malicious third party or used by users carelessly, these functions are always holding risks that may lead to security problems like information leakage. In this chapter, by indicating risk mitigation plans that developers can take against these functions, some topics that needs calling attentions, are picked up as articles.

6.1. Risk of Information Leakage from Clipboard

Copy & paste are the functions which users often use in a casual manner. For example, not a few users use these functions to store curious information or important information to remember in a mail or a web page into a notepad, or to copy and to paste a password from a notepad in which passwords are stored in order not to forget in advance. These are very casual actions at a glance, but actually there's a hidden risk that user handling information may be stolen.

The risk is related to mechanism of copy & paste in Android system. The information which was copied by user or application, is once stored in the buffer called Clipboard. The information stored in Clipboard is distributed to other applications when it is pasted by a user or an application. So there is a risk which leads to information leakage in this Clipboard function. It is because the entity of Clipboard is single in a system and any application can obtain the information stored in Clipboard at any time by using ClipboardManager. It means that all the information which user copied/cut, is leaked out to the malicious application.

Hence, application developers need to take measures to minimize the possibility of information leakage, considering the Android OS specifications.

6.1.1. Sample Code

Roughly speaking, there are two outlooks of counter-measures to mitigate the risk of information leakage form Clipboard.

1. Counter-measure when copying from other applications to your application.
2. Counter-measure when copying from your application to other applications.

Firstly, let us discuss the countermeasure 1 above. Supposing that a user copies character strings from other applications like note pad, Web browser or mailer application, and then paste it to EditText in your application. As it turns out, there's no basic counter-measure to prevent from sensitive information leakage due to copy & paste, in this scenario. Since there's no function in Android to control copy operations by the third party application.

So, regarding the countermeasure 1, there's no method other than explaining users the risk of copying & pasting sensitive information, and just continuing to enlighten users to decrease the

actions themselves continuously.

Next discussion is the countermeasure 2 above, supposing that the scenario that a user copies sensitive information displayed in your application. In this case, the sound counter-measure for leakage is to prohibit copying/cutting operations from View (TextView, EditText etc). If there are no copy/cut functions in View where the sensitive information (like personal information) is input/output, information leakage will never happen from your application via Clipboard.

There are several methods to prohibit copying/cutting. This section herein describes the easy and effective methods: One method is to disable long press View and another method is to delete copy/cut items from menu when selecting character string. "OK" in Table 6.1-1 stands for realizable, and "NG" stands for unrealizable. As per shown in Table 6.1-1, pay attention that copy/cut item cannot be deleted from the menu of character string selection in the case of API level is 10 or earlier.

Table 6.1-1 Method to prohibit copying/cutting in View which sensitive information is input/output

API Level	Disabling LongClick View	Delete copy/cut item from menu when character strings selected
10 or eariler	OK	NG
11 or later	OK	OK

Necessary of counter-measure can be determined as per the flow of Figure 6.1-1. In Figure 6.1-1, "Input type is fixed to Password attribute" means, the input type is necessarily either of the followings three when application is running. In this case, no counter-measures are required since copy/cut are prohibited as default.

- InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD
- InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_WEB_PASSWORD
- InputType.TYPE_CLASS_NUMBER | InputType.TYPE_NUMBER_VARIATION_PASSWORD

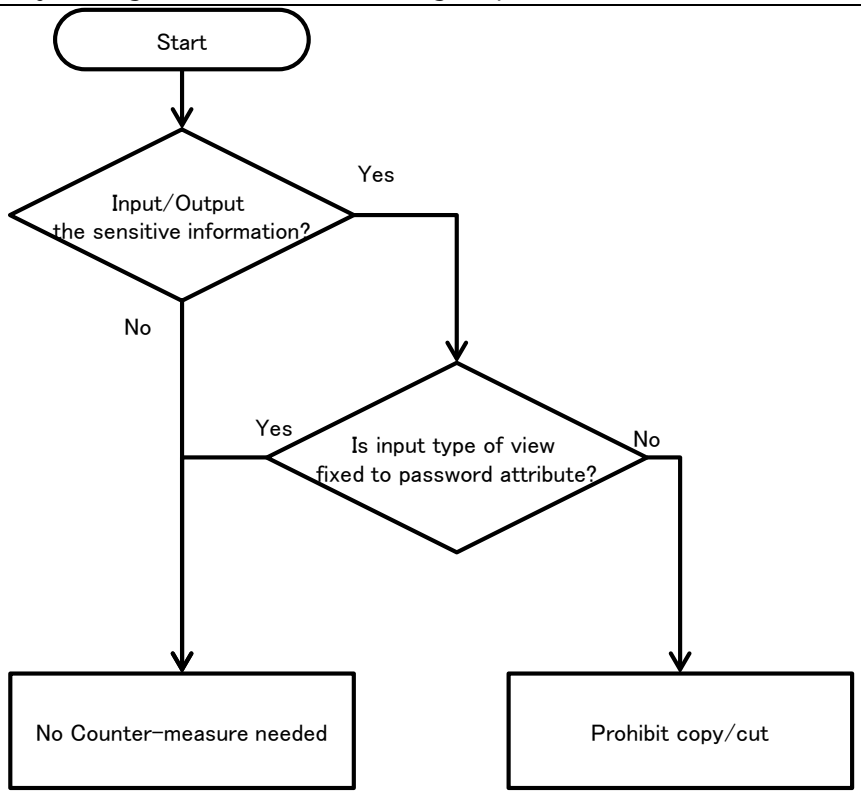


Figure 6.1-1Decision flow of counter-measure is required or not.

The following subsections detail each countermeasure shown in Table 6.1-1 along with sample codes.

6.1.1.1. Delete copy/cut from the menu when character string selection

TextView.setCustomSelectionActionMODECallback() method cannot be used in before Android 3.0(API Level 11). In this case, the easiest method to prohibit copying/cutting is to disable Long Click View. Disabling Long Click View can be specified in layout xml file.

Sample code to delete copy/cut item from menu of character string selection in EditText, is shown as per below.

Points:

1. Delete android.R.id.copy from the menu of character string selection.
2. Delete android.R.id.cut from the menu of character string selection.

UncopyableActivity.java

```
package org.jssec.android.clipboard.leakage;

import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.view.ActionMode;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
```

```
public class UncopyableActivity extends Activity {
    private EditText copyableEdit;
    private EditText uncopyableEdit;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.uncopyable);

        copyableEdit = (EditText) findViewById(R.id.copyable_edit);
        uncopyableEdit = (EditText) findViewById(R.id.uncopyable_edit);
        // When API Level 11 and more, by setCustomSelectionActionMODECallback method,
        // Possible to customize menu of character string selection.
        uncopyableEdit.setCustomSelectionActionModeCallback(actionModeCallback);
    }

    private ActionMode.Callback actionModeCallback = new ActionMode.Callback() {
        public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
            return false;
        }

        public void onDestroyActionMode(ActionMode mode) {
        }

        public boolean onCreateActionMode(ActionMode mode, Menu menu) {
            // *** POINT 1 *** Delete android.R.id.copy from the menu of character string selection.
            MenuItem itemCopy = menu.findItem(android.R.id.copy);
            if (itemCopy != null) {
                menu.removeItem(android.R.id.copy);
            }
            // *** POINT 2 *** Delete android.R.id.cut from the menu of character string selection.
            MenuItem itemCut = menu.findItem(android.R.id.cut);
            if (itemCut != null) {
                menu.removeItem(android.R.id.cut);
            }
            return true;
        }

        public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
            return false;
        }
    };

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.uncopyable, menu);
        return true;
    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
```



```
}
```

6.1.1.2. Disable Long Click View

TextView.setCustomSelectionActionMODECallback() method cannot be used in before Android 3.0 (API Level 11). In this case, the easiest method to prohibit copying/cutting is to disable Long Click View. Disabling Long Click View can be specified in layout xml file.

Point:

1. Set false to android:longClickable in View to prohibit copy/cut.

unlongclickable.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/unlongclickable_description" />

    <!-- EditText to prohibit copy/cut EditText -->
    <!-- *** POINT 1 *** Set false to android:longClickable in View to prohibit copy/cut. -->
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:longClickable="false"
        android:hint="@string/unlongclickable_hint" />
</LinearLayout>
```

6.1.2. Rule Book

Follow the rule below when copying sensitive information from your application to other applications.

1. Disabling Copy/Cut Character Strings that Are Displayed in View (Required)

6.1.2.1. Disabling Copy/Cut Character Strings that Are Displayed in View (Required)

If there's a View which displays sensitive information in an application and besides the information is allowed to be copied/cut like EditText in the View, the information may be leaked via Clipboard. Therefore, copy/cut must be disabled in View where sensitive information is displayed.

There are two methods to disable copy/cut. One method is to delete items of copy/cut from menu of character string selection, and another method is to disable Long Click View. The former method can be achieved by using setCustomSelectionActionMODECallback() method in Android 3.0 (API Level 11) and later version. And the latter one can be achieved in whatever before and after Android 3.0 (API Level 11).

Please refer to "6.1.3.1 Precautions When Applying Rules."

6.1.3. Advanced Topics

6.1.3.1. Precautions When Applying Rules

In TextView, selecting character string is impossible as default, so normally no counter-measure is required, but in some cases copying is possible depends on application's specifications. In Android3.0(API Level 11) and later, possible to select/copy character strings or not can be dynamically set by using TextView.setTextIsSelectable() method. When setting copying possible in TextView, investigate the possibility that any sensitive information is displayed in TextView, and if there are any possibilities, it should not be set as possible to copy.

In addition, described in the decision flow of "6.1.1Sample Code" regarding EditText which is input type (InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD etc), supposing password input, normally any counter-measures are not required since copying character strings are prohibited as default. However, as described in "5.1.2.2 Provide the Option to Display Password in a Plain Text (Required)," when the option to [display password in a plain text] is prepared, in case of displaying password in a plain text, input type will change and copy/cut is enabled. So the same counter-measure should be required.

Note that, developers should also take usability of application into consideration when applying rules. For example, in the case of View which user can input text freely, if copy/cut is disabled because there is the slight possibility that sensitive information is input, users may feel inconvenience. Of course, the rule should unconditionally be applied to View which treats highly important information or independent sensitive information, but in the case of View other than those, the following questions will help developers to understand how properly to treat View.

- Prepare some other component for the exclusive use of sensitive information
- Send information with alternative methods when the pasted-to application is obvious
- Call users for cautions about inputting/outputting information
- Reconsider the necessity of View

The root cause of the information leakage risk is that the specifications of Clipboard and ClipboardManager in Android OS leave the security risk out of consideration. Application developers need to create higher quality applications in terms of user integrity, usability, functions, and so forth.

6.1.3.2. Operating Information Stored in Clipboard

As mentioned in "6.1 Risk of Information Leakage from Clipboard," an application can manipulate information stored in Clipboard by using ClipboardManager. In addition, there is no need to set particular Permission for using ClipboardManager and thus the application can use ClipboardManager without being recognized by user.

Here is how to realize the above mentioned implementation in API Level 11 and later, for your reference.

Information, called ClipData, stored in Clipboard can be obtained with ClipboardManager.getPrimaryClip() method. If a listener is registered to ClipboardManager by ClipboardManager.addPrimaryClipChangedListener() method implementing OnPrimaryClipChangedListener, the listener is called every time copy/cut operations occurred by user. Therefore ClipData can be got without overlooking the timing. Listener call is executed when copy/cut operations occur in any application regardless.

The following shows the source code of Service, which gets ClipData whenever copy/cut is executed in a device and displays it through Toast. You can realize that information stored in Clipboard is leaked out doe to simple codes as follows. It's necessary to pay attention that the sensitive information is not taken at least by the following source code.

ClipboardListeningService.java

```
package org.jssec.android.clipboard;

import android.app.Service;
import android.content.ClipData;
import android.content.ClipboardManager;
import android.content.ClipboardManager.OnPrimaryClipChangedListener;
import android.content.Context;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class ClipboardListeningService extends Service {
    private static final String TAG = "ClipboardListeningService";
    private ClipboardManager mClipboardManager;

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        mClipboardManager = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
        if (mClipboardManager != null) {
            mClipboardManager.addPrimaryClipChangedListener(clipListener);
        } else {
            Log.e(TAG, "Failed to get ClipboardService . Service is closed.");
            this.stopSelf();
        }
    }

    @Override
    public int onStartCommand(Intent intent, int flag, int startId) {
        super.onStartCommand(intent, flag, startId);
        ServiceRunningStatus.setStatus(getApplicationContext(), true);
        return START_STICKY;
    }

    @Override
```



```
    public void onDestroy() {
        super.onDestroy();
        if (mClipboardManager != null) {
            mClipboardManager.removePrimaryClipChangedListener(clipListener);
            ServiceRunningStatus.setStatus(getApplicationContext(), false);
        }
    }

    private OnPrimaryClipChangedListener clipListener = new OnPrimaryClipChangedListener() {
        public void onPrimaryClipChanged() {
            if (mClipboardManager != null && mClipboardManager.hasPrimaryClip()) {
                ClipData data = mClipboardManager.getPrimaryClip();
                ClipData.Item item = data.getItemAt(0);
                Toast
                    .makeText(
                        getApplicationContext(),
                        "Character stirng that is copied or cut:¥n"
                            + item.coerceToText(getApplicationContext()),
                        Toast.LENGTH_SHORT)
                    .show();
            }
        }
    };
}
```

Next, below shows an example code of Activity which uses ClipboardListeningService touched in the above.

ClipboardListeningActivity.java

```
package org.jssec.android.clipboard;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

public class ClipboardListeningActivity extends Activity {
    private static final String TAG = "ClipboardListeningActivity";
    private boolean isServiceRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_clipboard_listening);

        isServiceRunning = ServiceRunningStatus.getStatus(getApplicationContext());
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_clipboard_listening, menu);
```

```
        return true;
    }

    public void onClickStartService(View view) {
        if (view.getId() != R.id.start_service_button) {
            Log.w(TAG, "View ID is incorrect.");
        } else {
            if (!isServiceRunning) {
                ComponentName cn = startService(
                        new Intent(ClipboardListeningActivity.this, ClipboardListeningService.class));
                if (cn != null) {
                    this.isServiceRunning = true;
                } else {
                    Log.e(TAG, "Failed to launch the service.");
                    Toast.makeText(this, "Failed to launch the service.", Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(this, "Service has been already launched.", Toast.LENGTH_SHORT).show();

            }
        }
    }

    public void onClickStopService(View view) {
        if (view.getId() != R.id.stop_service_button) {
            Log.w(TAG, "View ID is incorrect.");
        } else {
            if (isServiceRunning) {
                boolean res = stopService(
                        new Intent(ClipboardListeningActivity.this, ClipboardListeningService.class));
                if (res) {
                    this.isServiceRunning = false;
                } else {
                    Log.e(TAG, "Failed to stop the service.");
                    Toast.makeText(this, "Failed to stop the service.", Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(this, "Service has been already stopped.", Toast.LENGTH_SHORT).show();

            }
        }
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(this);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

By the way, ClipboardManager in before API Level 11 does not have any method to register Listener like setOnPrimaryClipChangedListener(). Hence, an application cannot get ClipData whenever copy

/cut operations are executed by user, but an application which is equivalent in behavior can be made in such a manner as to continuously obtain information stored in Clipboard at short intervals by using AlarmManager and the like. Note also that, when implementing before API Level 11, getPrimaryClip() method cannot be used. Therefore the information stored in Clipboard should be obtained as character strings with getText() method. Sample codes for this are omitted here.

How to obtain information stored in Clipboard is described so far. On the other hand it is also possible to store new information in Clipboard. In the case of API Level 11 or later, ClipboardManager.setPrimaryClip() can be used. And in the case of before that API 11, ClipboardManager.setText() method can be used.

Note that setPrimaryClip() or setText() method will overwrite the information stored in Clipboard, therefore the information stored by user's copy/cut may be lost. When providing custom copy/cut functions with these methods, it's necessary to design/implement in order not that the contents stored in Clipboard are changed to unexpected contents, by displaying a dialogue to notify the contents are to be changed, according the necessity.