Mobile Phone Security

Dr. Digvijaysinh Rathod
Associate Professor
(Cyber Security and Digital Forensics)
Institute of Forensic Science
Gujarat Forensic Sciences University

digvijay.rathod@gfsu.edu.in

App Permission

Reference

www.developer.google.com

https://www.geeksforgeeks.org/android-how-to-request-permissions-in-android-application/

Permissions overview

- ✓ The purpose of a **permission is to protect the privacy of an Android user.**
- ✓ Android apps must **request permission to access sensitive user data** (such as **contacts and SMS**), as well as certain system features (such as **camera and internet**).
- ✓ Depending on the feature, the system might grant the permission **automatically** or might **prompt** the user to approve the request.

Permissions overview

- ✓ A central design point of the Android security architecture is that **no app, by default, has permission to perform any operations** that would adversely impact other apps, the operating system, or the user.
- ✓ This includes **reading or writing the user's private data** (such as contacts or emails), reading or writing another app's files, performing network access, keeping the device awake, and so on.

Permission approval

- ✓ An app must publicize the permissions it requires by including **<uses-permission>** tags in the app manifest.
- ✓ For example, an app that needs to **send SMS messages** would have this line in the manifest:

```
<manifest        xmlns:android=http://schemas.android.com/apk/res/android
 package="com.example.snazzyapp">
<uses-permission android:name="android.permission.SEND_SMS"/>
<application...>
       ...
   </application>
</manifest>
```

Permission approval

- ✓ If your app lists **normal permissions** in its manifest (that is, permissions that **don't pose much risk** to the user's privacy or the device's operation), the system automatically grants those permissions to your app.
- ✓ If your app lists **dangerous permissions** in its manifest (that is, permissions that could potentially affect the user's privacy or the device's normal operation), such as the **SEND_SMS permission above**, the user must explicitly agree to grant those permissions.

Protection levels

- ✓ Permissions are divided into several **protection levels.**
- ✓ The protection level affects whether runtime permission requests are required.
- ✓ There are three protection levels that affect **third-party apps: normal, signature, and dangerous permissions.**

Protection levels

1. **Normal permissions:**

- ✓ Normal permissions cover areas where **your app needs to access data or resources outside the app's sandbox**, but where there's very **little risk** to the user's privacy or the operation of other apps.
- ✓ For example, permission to **set the time zone** is a normal permission.

Protection levels

1. **Normal permissions:**

✓ If an app declares in its manifest that it needs a **normal permission**, the **system automatically** grants the app that permission at install time.

The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

Protection levels

1. **Signature permissions**

- ✓ The system grants these app permissions at install time, but only when the app that attempts to use a permission is **signed by the same certificate** as the app that defines the permission.
- ✓ It’s granted automatically by the system if both applications are signed with the same certificate

Protection levels

1. **Signature permissions**

- ✓ both applications have been developed by the same company, this means that they most certainly will share the same application signature (**are signed with the same .keystore)**;
- ✓ so we can take this into advantage and define a custom permission with this increased level of security — signature.

Protection levels

1. **Signature permissions**

- ✓ There are two major advantages of this:
  - ✓ There’s no need to prompt the user asking for a different type of permission in order to communicate with another application.
  - ✓ Since they share the same signature, the OS automatically grants this access; it’s expected that since the company behind them is the same they’re trustworthy.

Protection levels

1. Signature permissions

- ✓ There are two major advantages of this:
  - ✓ Since this is addressed by the OS, there’s no need to implement a validation mechanism.

Protection levels

1. **Dangerous permissions**

- ✓ Dangerous permissions cover areas where the app wants **data or resources that involve the user's private information**, or could potentially affect the user's stored data or the operation of other apps.
- ✓ For example, the ability to read the **user's contacts** is a dangerous permission.

Protection levels

1. **Dangerous permissions**

- ✓ android.permission_group.CALENDAR
  - ✓ android.permission.READ_CALENDAR
  - ✓ android.permission.WRITE_CALENDAR
- ✓ android.permission_group.CAMERA
  - ✓ android.permission.CAMERA
- ✓ android.permission_group.LOCATION
  - ✓ android.permission.ACCESS_FINE_LOCATION
  - ✓ android.permission.ACCESS_COARSE_LOCATION

Protection levels

1. **Dangerous permissions**

- ✓ android.permission_group.CONTACTS
  - ✓ android.permission.READ_CONTACTS
  - ✓ android.permission.WRITE_CONTACTS
  - ✓ android.permission.GET_ACCOUNTS
- ✓ android.permission_group.MICROPHONE
  - ✓ android.permission.RECORD_AUDIO
- ✓ android.permission_group.SENSORS
  - ✓ android.permission.BODY_SENSORS

Protection levels

1. **Dangerous permissions**

- ✓ android.permission_group.PHONE
  - ✓ android.permission.READ_PHONE_STATE
  - ✓ android.permission.CALL_PHONE
  - ✓ android.permission.READ_CALL_LOG
  - ✓ android.permission.WRITE_CALL_LOG
  - ✓ android.permission.ADD_VOICEMAIL
  - ✓ android.permission.USE_SIP
  - ✓ android.permission.PROCESS_OUTGOING_CALLS

Protection levels

1. Dangerous permissions

- ✓ android.permission_group.SMS
  - ✓ android.permission.SEND_SMS
  - ✓ android.permission.RECEIVE_SMS
  - ✓ android.permission.READ_SMS
  - ✓ android.permission.RECEIVE_WAP_PUSH
  - ✓ android.permission.RECEIVE_MMS
  - ✓ android.permission.READ_CELL_BROADCASTS

Protection levels

1. **Dangerous permissions**

- ✓ android.permission_group.STORAGE
  - ✓ android.permission.READ_EXTERNAL_STORAGE
  - ✓ android.permission.WRITE_EXTERNAL_STORAGE

Protection levels

1. Dangerous permissions

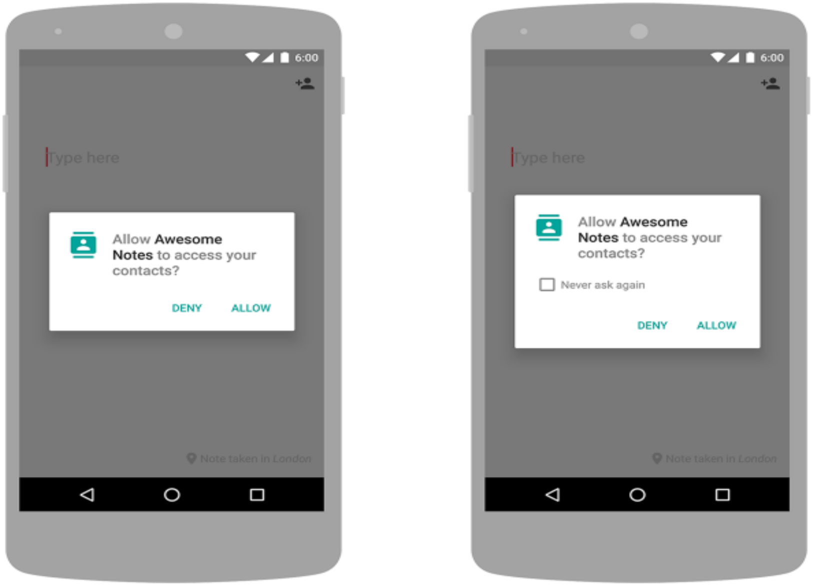
- ✓ If an app declares that it needs a dangerous permission, the user has to **explicitly grant the permission** to the app.
- ✓ Until the user approves the permission, your app cannot provide functionality that depends on that permission.
- ✓ To use a dangerous permission, your app must **prompt the user to grant permission at runtime**.

Runtime requests (Android 6.0 and higher) - marshmallow

- ✓ If the device is **running Android 6.0 (API level 23)** or higher, and the app's **targetSdkVersion is 23** or higher, the user isn't notified of any app permissions at install time.
- ✓ Your app must ask the user to **grant the dangerous permissions at runtime.**
- ✓ When your app requests permission, the user sees a system dialog (as shown in figure 1, left) telling the user which permission group your app is trying to access. The dialog includes a Deny and Allow button.

Runtime requests (Android 6.0 and higher)

- ✓ The dialog includes a **Deny and Allow button.**
- ✓ If the user denies the permission request, the next time your app requests the permission, the dialog contains a checkbox that, when checked, indicates the user doesn't want to be prompted for the permission again

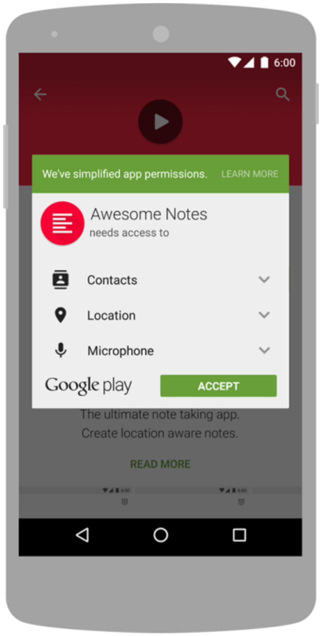


Allow Awesome Notes to access your contacts?

Allow Awesome Notes to access your contacts?

Install-time requests (Android 5.1.1 and below) - Lollipop

- ✓ If the device is running **Android 5.1.1 (API level 22)** or lower, or the app's **targetSdkVersion is 22 or lower** while running on any version of Android, the system automatically asks the user to **grant all dangerous permissions for your app at install-time**.



6:00

We've simplified app permissions. LEARN MORE

Awesome Notes
needs access to

Contacts

Location

Microphone

Google play ACCEPT

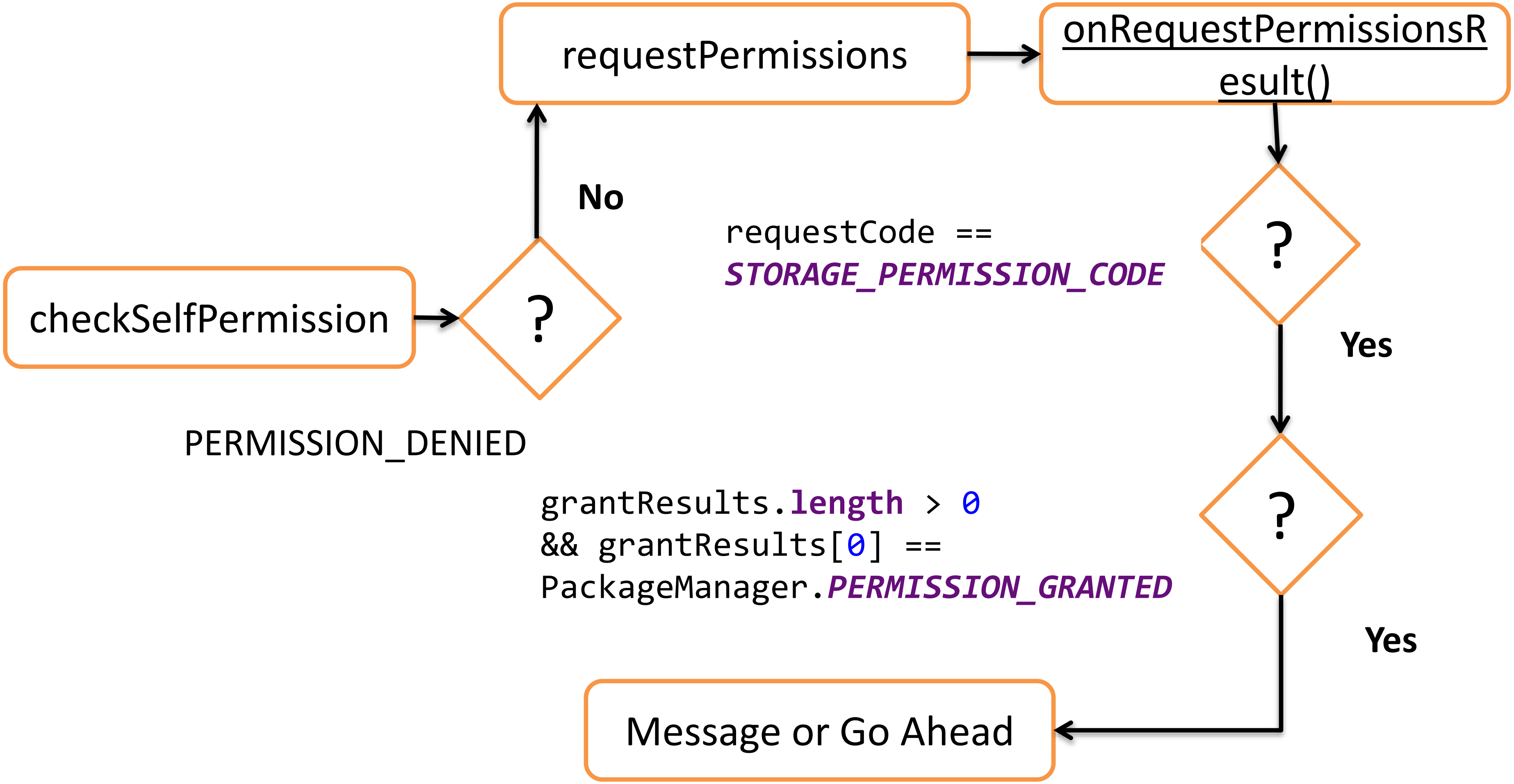
The ultimate note taking app.
Create location aware notes.

READ MORE

Install-time requests (Android 5.1.1 and below)

- ✓ If the user clicks **Accept**, all permissions the app requests are granted. If the user denies the permissions request, the system cancels the installation of the app.
- ✓ If an app **update includes** the need for additional permissions the user is prompted to accept those new permissions before updating the app.

Steps



checkSelfPermission

?

PERMISSION_DENIED

No

requestPermissions

onRequestPermissionsResult()

?

```
requestCode ==
STORAGE_PERMISSION_CODE
```

Yes

?

```
grantResults.length > 0
&& grantResults[0] ==
PackageManager.PERMISSION_GRANTED
```

Yes

Message or Go Ahead

Steps for Requesting permissions at run time

- ✓ Declare the permission in Android Manifest file: In Android permissions are declared in AndroidManifest.xml file using the uses-permission tag.
- ✓ **<uses-permission android:name="android.permission.READ_CONTACTS" />**
- ✓ Here we are declaring storage and camera permission.

Steps for Requesting permissions at run time

- ✓ **Step-1 :** Declare the permission in Android Manifest file: In Android permissions are declared in AndroidManifest.xml file using the uses-permission tag.
- ✓ **<uses-permission android:name="android.permission.READ_CONTACTS" />**
- ✓ Here we are declaring storage and camera permission.

Steps for Requesting permissions at run time

- ✓ **Step – II** Check whether permission is already granted or not.
- ✓ If permission isn’t already granted, request user for the permission:
- ✓ In order to use any service or feature, the permissions are required.
- ✓ Hence we have to ensure that the permissions are given for that. If not, then the permissions are requested.

Steps for Requesting permissions at run time

Syntax:

```
✓ If (ContextCompat.checkSelfPermission(thisActivity,
  Manifest.permission.READ_CONTACTS)
  != PackageManager.PERMISSION_GRANTED)
  { // Permission is not granted }
```

Steps for Requesting permissions at run time

- ✓ **Step – III** Request Permissions:
  - ✓ When PERMISSION_DENIED is returned from the checkSelfPermission() method in the above syntax, we need to prompt the user for that permission. Android provides several methods that can be used to request permission, such **as requestPermissions().**

Steps for Requesting permissions at run time

Syntax:

- ✓ ActivityCompat.requestPermissions(MainActivity.this, permissionArray, requestCode);
- ✓ Here permissionArray is an array of type String.

Steps for Requesting permissions at run time

- ✓ **Step – III** Request Permissions:
  - ✓ When PERMISSION_DENIED is returned from the checkSelfPermission() method in the above syntax, we need to prompt the user for that permission. Android provides several methods that can be used to request permission, such **as requestPermissions().**

Steps for Requesting permissions at run time

- ✓ **Step – III**

  Override onRequestPermissionsResult() method:

  - ✓ onRequestPermissionsResult() is called when user grant or decline the permission.
  - ✓ RequestCode is one of the parameteres of this function which is used to check user action for corresponding request.
  - ✓ Here a toast message is shown indicating the permission and user action.

Steps for Requesting permissions at run time

- ✓ **Step – III**

  Override onRequestPermissionsResult() method:

- ✓ **Suntax**

  onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)

Mobile Phone Security

Dr. Digvijaysinh Rathod
Associate Professor
(Cyber Security and Digital Forensics)
Institute of Forensic Science
Gujarat Forensic Sciences University

digvijay.rathod@gfsu.edu.in