Unit-4: Permissions 4

Unit Structure

4.0 Learning Objectives

After studying this unit, students will be able to:

- Define permission
- List different types of permission and their uses
- Define custom app permission
- Understand various permission protection levels

4.1 Introduction

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.

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.

This unit provides an overview of how Android permissions work, including: how permissions are presented to the user, the difference between install-time and runtime permission requests, how permissions are enforced, and the types of permissions and their groups.

4.2 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>
```

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.

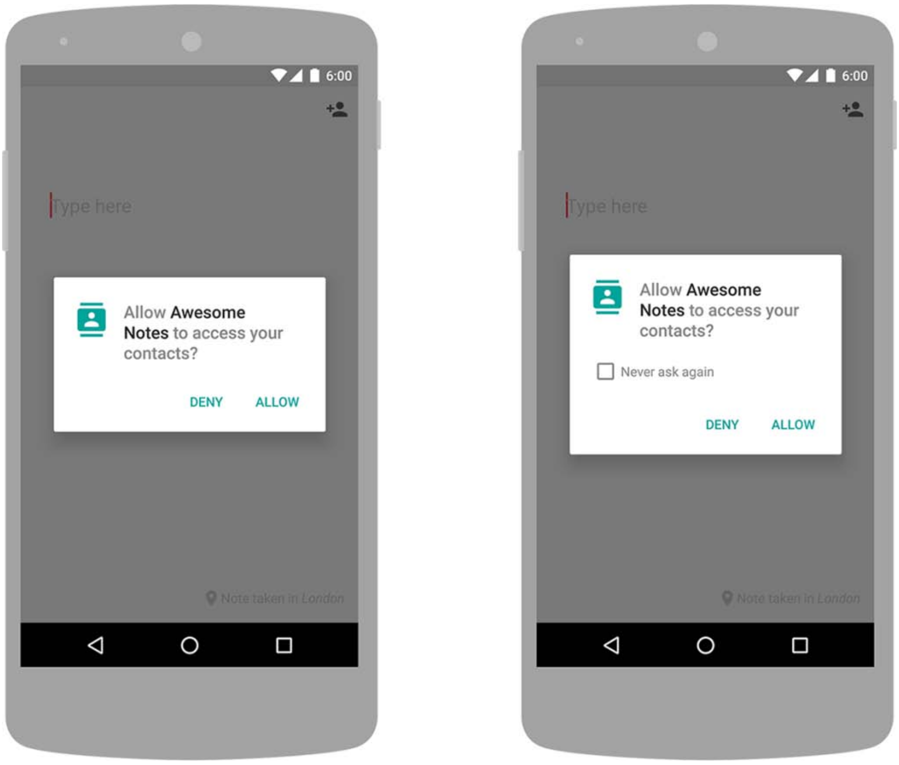
4.3 Request prompts for dangerous permissions

Only dangerous permissions require user agreement. The way Android asks the user to grant dangerous permissions depends on the version of Android running on the user's device, and the system version targeted by your app.

Runtime requests (Android 6.0 and higher)

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 telling the user which permission group your app is trying to access. 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 as shown in figure 2.



Allow **Awesome Notes** to access your contacts?

Allow **Awesome Notes** to access your contacts?

□ Never ask again

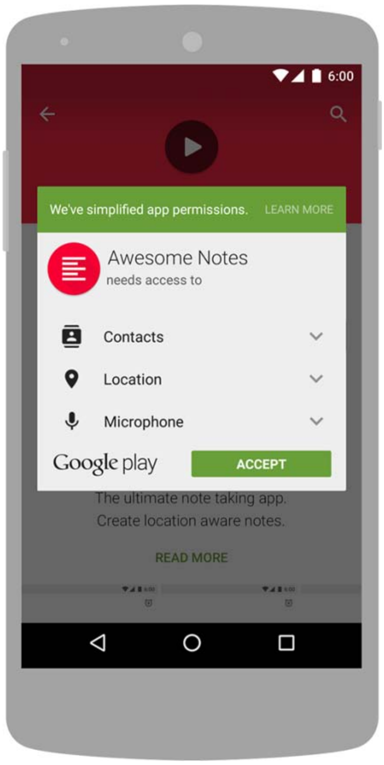
Figure-58

If the user checks the Never ask again box and taps Deny, the system no longer prompts the user if you later attempt to requests the same permission.

Even if the user grants your app the permission it requested you cannot always rely on having it. Users also have the option to enable and disable permissions one-by-one in system settings. You should always check for and request permissions at runtime to guard against runtime errors (SecurityException).

Install-time requests (Android 5.1.1 and below)

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 as shown in figure 2.



We've simplified app permissions.

Awesome Notes needs access to

The ultimate note taking app.
Create location aware notes.

Figure-59

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.

4.4 Permissions for optional hardware features

Access to some hardware features such as Bluetooth or the camera requires app permission. However, not all Android devices actually have these hardware features. So if your app requests the CAMERA permission, it's important that you also include the <uses-feature> tag in your manifest to declare whether or not this feature is actually required. For example:

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

If you declare android:required="false" for the feature, then Google Play allows your app to be installed on devices that don't have the feature. You then must check if the current device has the feature at runtime by calling

PackageManager.hasSystemFeature(), and gracefully disable that feature if it's not available.

If you don't provide the <uses-feature> tag, then when Google Play sees that your app requests the corresponding permission, it assumes your app requires this feature. So it filters your app from devices without the feature, as if you declared android:required="true" in the <uses-feature> tag.

4.5 Custom App Permission

Permissions aren't only for requesting system functionality. Services provided by apps can enforce custom permissions to restrict who can use them.

Activity permission enforcement

Permissions applied using the android:permission attribute to the <activity> tag in the manifest restrict who can start that Activity. The permission is checked during Context.startActivity() and Activity.startActivityForResult(). If the caller doesn't have the required permission then SecurityException is thrown from the call.

Service permission enforcement

Permissions applied using the android:permission attribute to the <service> tag in the manifest restrict who can start or bind to the associated Service. The permission is checked during Context.startService(), Context.stopService() and Context.bindService(). If the caller doesn't have the required permission then SecurityException is thrown from the call.

Broadcast permission enforcement

Permissions applied using the android:permission attribute to the <receiver> tag restrict who can send broadcasts to the associated BroadcastReceiver. The permission is checked after Context.sendBroadcast() returns, as the system tries to deliver the submitted broadcast to the given receiver. As a result, a permission failure doesn't result in an exception being thrown back to the caller; it just doesn't deliver the Intent.

In the same way, a permission can be supplied to Context.registerReceiver() to control who can broadcast to a programmatically registered receiver. Going the other way, a permission can be supplied when calling Context.sendBroadcast() to restrict which broadcast receivers are allowed to receive the broadcast.

Note that both a receiver and a broadcaster can require permission. When this happens, both permission checks must pass for the intent to be delivered to the associated target.

Content Provider permission enforcement

Permissions applied using the android:permission attribute to the <provider> tag restrict who can access the data in a ContentProvider. Unlike the other components, there are two separate permission attributes you can set: android:readPermission restricts who can read from the provider, and android:writePermission restricts who can write to it. Note that if a provider is protected with both a read and write permission, holding only the write permission doesn't mean you can read from a provider.

The permissions are checked when you first retrieve a provider and as you perform operations on the provider.

Using ContentResolver.query() requires holding the read permission;

using ContentResolver.insert(), ContentResolver.update(), ContentResolver.delete() requires the write permission. In all of these cases, not holding the required permission results in a SecurityException being thrown from the call.

URI permissions

The standard permission system described so far is often not sufficient when used with content providers. A content provider may want to protect itself with read and write permissions, while its direct clients also need to hand specific URIs to other apps for them to operate on.

A typical example is attachments in a email app. Access to the emails should be protected by permissions, since this is sensitive user data. However, if a URI to an image attachment is given to an image viewer, that image viewer no longer has permission to open the attachment since it has no reason to hold a permission to access all email.

The solution to this problem is per-URI permissions: when starting an activity or returning a result to an activity, the caller can set Intent.FLAG_GRANT_READ_URI_PERMISSION and/or Intent.FLAG_GRANT_WRITE_URI_PERMISSION. This grants the receiving activity permission access the specific data URI in the intent, regardless of whether it has any permission to access data in the content provider corresponding to the intent.

This mechanism allows a common capability-style model where user interaction (such as opening an attachment or selecting a contact from a list) drives ad-hoc granting of fine-grained permission. This can be a key facility for reducing the permissions needed by apps to only those directly related to their behavior.

To build the most secure implementation that makes other apps accountable for their actions within yor app, you should use fine-grained permissions in this manner and declare your app's support for it with the android:grantUriPermissions attribute or <grant-uri-permissions> tag.

Other permission enforcement

Arbitrarily fine-grained permissions can be enforced at any call into a service. This is accomplished with the Context.checkCallingPermission() method. Call with a desired permission string and it returns an integer indicating whether that permission has been granted to the current calling process. Note that this can only be used when you are executing a call coming in from another process, usually through an IDL interface published from a service or in some other way given to another process.

There are a number of other useful ways to check permissions. If you have the process ID (PID) of another process, you can use the Context.checkPermission() method to check a permission against that PID. If you have the package name of another app, you can use the PackageManager.checkPermission() method to find out whether that particular package has been granted a specific permission.

Check your progress-1

a) What is purpose of permission?

b) Every apphas by default, permission to perform any operations that would adversely impact other apps, the operating system, or the user (True/False)

c) An app must publicize the permissions it requires by including ______tags in the app manifest.

d) In mobile with Android 6.0 and higher, the request for permission is requested to user at __________

(A) Runtime (B) Install-time (C) Either (A) or (B) (D) Neither (A) nor (B)

e) Services provided by apps can enforce custom permissions to restrict who can use them.

4.6 Permission 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.

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.

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.

As of Android 9 (API level 28), the following permissions are classified as PROTECTION_NORMAL:

ACCESS_LOCATION_EXTRA_COMMANDS
ACCESS_NETWORK_STATE
ACCESS_NOTIFICATION_POLICY
ACCESS_WIFI_STATE
BLUETOOTH
BLUETOOTH_ADMIN
BROADCAST_STICKY
CHANGE_NETWORK_STATE
CHANGE_WIFI_MULTICAST_STATE
CHANGE_WIFI_STATE
DISABLE_KEYGUARD
EXPAND_STATUS_BAR
FOREGROUND_SERVICE
GET_PACKAGE_SIZE
INSTALL_SHORTCUT
INTERNET
KILL_BACKGROUND_PROCESSES
MANAGE_OWN_CALLS
MODIFY_AUDIO_SETTINGS
NFC
READ_SYNC_SETTINGS
READ_SYNC_STATS
RECEIVE_BOOT_COMPLETED
REORDER_TASKS
REQUEST_DELETE_PACKAGES
SET_ALARM
SET_WALLPAPER
SET_WALLPAPER_HINTS
TRANSMIT_IR
USE_FINGERPRINT
VIBRATE
WAKE_LOCK
WRITE_SYNC_SETTINGS

Signature permissions

The system grants these app permissions at install time, but only when the app that attempts to use permission is signed by the same certificate as the app that defines the permission.

As of Android 8.1 (API level 27), the following permissions that third-party apps can use are classified as PROTECTION_SIGNATURE:

BIND_ACCESSIBILITY_SERVICE
BIND_AUTOFILL_SERVICE
BIND_CARRIER_SERVICES
BIND_CHOOSER_TARGET_SERVICE
BIND_CONDITION_PROVIDER_SERVICE
BIND_DEVICE_ADMIN
BIND_DREAM_SERVICE
BIND_INCALL_SERVICE
BIND_INPUT_METHOD
BIND_MIDI_DEVICE_SERVICE
BIND_NFC_SERVICE
BIND_NOTIFICATION_LISTENER_SERVICE
BIND_PRINT_SERVICE
BIND_SCREENING_SERVICE
BIND_TELECOM_CONNECTION_SERVICE
BIND_TEXT_SERVICE
BIND_TV_INPUT
BIND_VISUAL_VOICEMAIL_SERVICE
BIND_VOICE_INTERACTION
BIND_VPN_SERVICE
BIND_VR_LISTENER_SERVICE
BIND_WALLPAPER
CLEAR_APP_CACHE
MANAGE_DOCUMENTS
READ_VOICEMAIL
REQUEST_INSTALL_PACKAGES
SYSTEM_ALERT_WINDOW
WRITE_SETTINGS
WRITE_VOICEM

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. 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. For a list of dangerous permissions, see table 16 below.

Permission Group	Permissions
CALENDAR	READ_CALENDAR WRITE_CALENDAR
CALL_LOG	READ_CALL_LOG WRITE_CALL_LOG PROCESS_OUTGOING_CALLS
CAMERA	CAMERA
CONTACTS	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE READ_PHONE_NUMBERS CALL_PHONE ANSWER_PHONE_CALLS ADD_VOICEMAIL USE_SIP
SENSORS	BODY_SENSORS
SMS	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Table-16: Dangerous permissions and permission groups.

Special permissions

There are a couple of permissions that don't behave like normal and dangerous permissions. SYSTEM_ALERT_WINDOW and WRITE_SETTINGS are particularly sensitive, so most apps should not use them. If an app needs one of these permissions, it must declare the permission in the manifest, and send an intent requesting the user's authorization. The system responds to the intent by showing a detailed management screen to the user.

4.7 How to View app's permissions

You can view all the permissions currently defined in the system using the Settings app and the shell command adb shell pm list permissions. To use the Settings app, go to Settings > Apps. Pick an app and scroll down to see the permissions that the app uses. For developers, the adb '-s' option displays the permissions in a form similar to how the user sees them:

```
$ adb shell pm list permissions -s

All Permissions:

Network communication: view Wi-Fi state, create Bluetooth connections, fullinternet access, view network state

Your location: access extra location provider commands, fine (GPS) location,mock location sources for testing, coarse (network-based) location

Services that cost you money: send SMS messages, directly call phone numbers

...
```

You can also use the adb -g option to grant all permissions automatically when installing an app on an emulator or test device:

```
$ adb shell install -g MyApp.apk
```

Check your progress-2

- Which of the following is a protection level that affects third-party apps?

  (A) Normal (B) Signature (C) Dangerous (D) All of these

- SEND_SMS is ____________ type of permission

  (A) Normal (B) Signature (C) Dangerous (D) All of these

- READ_VOICEMAIL is ____________ type of permission

(A) Normal (B) Signature (C) Dangerous (D) All of these

d) SET_WALLPAPER is ____________ type of permission

(A) Normal (B) Signature (C) Dangerous (D) All of these

4.8 Let us sum up

In this unit you have learned about permission, different types of permissions, how to define custom permission and various permission protection levels that affect third-party apps.

4.9 Check your Progress: Possible Answers

1-a) The purpose of a *permission* is to protect the privacy of an Android user

1-b) False

1-c) <uses-permission>

1-d) (C)Either (A) or (B)

1-e) True

2-a) (D) All of these

2-b) (C) Dangerous

2-c) (B) Signature

2-d) (A) Normal

4.10 Further Reading

- https://developer.android.com/training/permissions/usage-notes
- https://developer.android.com/guide/topics/permissions/default-handlers
- https://developer.android.com/guide/topics/permissions/defining

4.11 Assignment

- Write detailed note on permissions
- Explain permissions for optional hardware features
- Explain Permission Protection levels in details

4.12 Activity

- Check the permissions used by different Apps installed in your Android Mobile and remove any unnecessary permission granted.