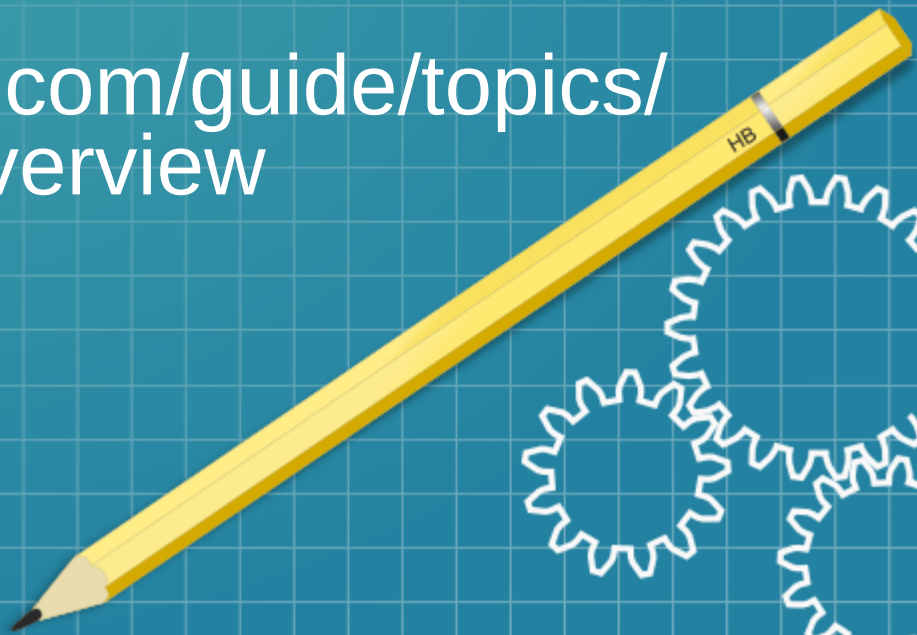
Permissions

https://developer.android.com/guide/topics/permissions/overview



Cont ...

- 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.

Cont ...

- 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"/>
</manifest>
```

Cont ...

- 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.

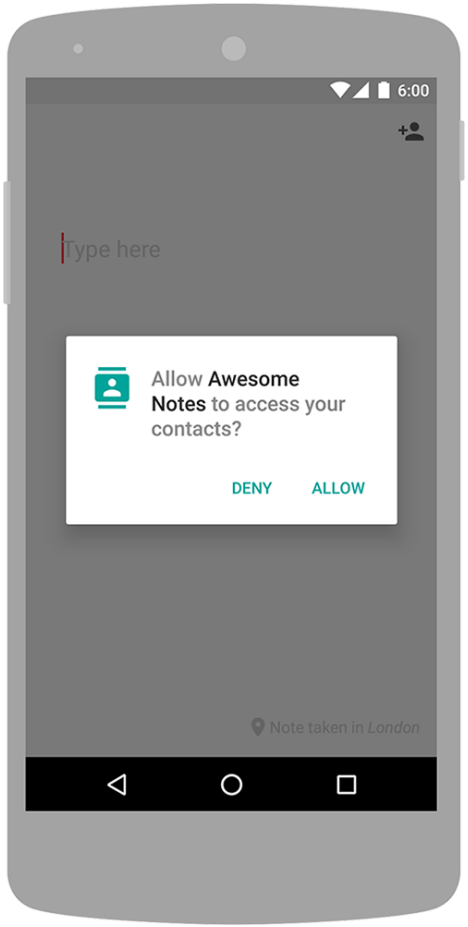
Request prompts for dangerous permissions

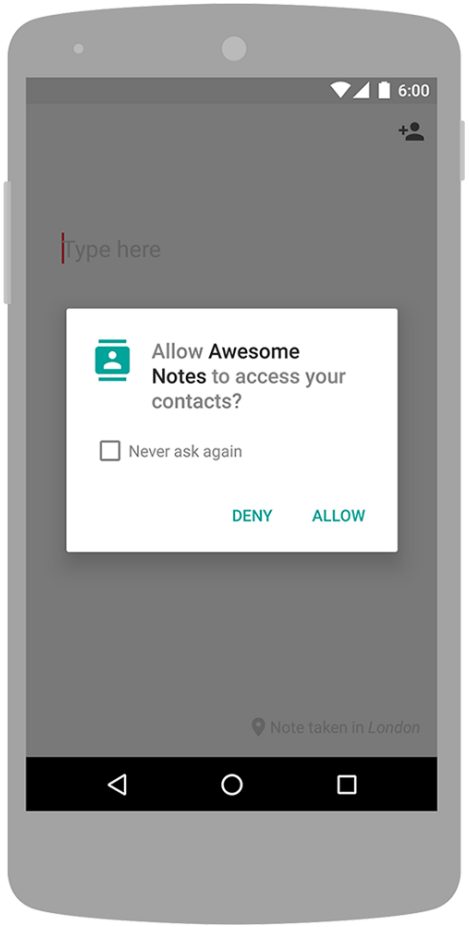
- Only dangerous permissions require user agreement.
- The way Android asks the user to grant dangerous permissions depends on the version of Android running on the user's device, and the system version targeted by your app.
  - Runtime requests (Android 6.0 and higher)
    - If the device is running 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.

Cont ...

- 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.
- 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

Cont ...


6:00
Type here
Allow Awesome Notes to access your contacts?
DENY
ALLOW
Note taken in London


6:00
Type here
Allow Awesome Notes to access your contacts?
Never ask again
DENY
ALLOW
Note taken in London

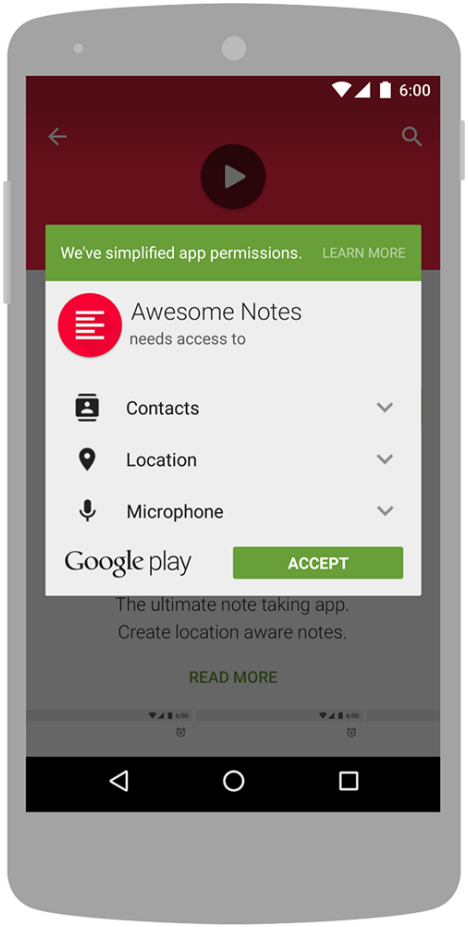
Cont ...

- 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.

Cont ...

- 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 (see figure 2).

Cont ...


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

Cont ...

- 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.

Permission enforcement

- 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.

Cont ...

- 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.

Cont ...

- 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.

Cont ...

- Note that both a receiver and a broadcaster can require a permission.
- When this happens, both permission checks must pass for the intent to be delivered to the associated target.
- For more information, see Restricting broadcasts with permissions.

Cont ...

- Content Provider permission enforcement
  - Permissions applied using the android:permission attribute to the <provider> tag restrict who can access the data in a ContentProvider.
  - (Content providers have an important additional security facility available to them called URI permissions which is described next.) 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.

Cont ...

- The permissions are checked when you first retrieve a provider (if you don't have either permission, a SecurityException is thrown), 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.

Cont ...

- 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.

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.

Cont ...

- 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.

Cont ...

- 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.
- 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.

Cont ...

- 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.

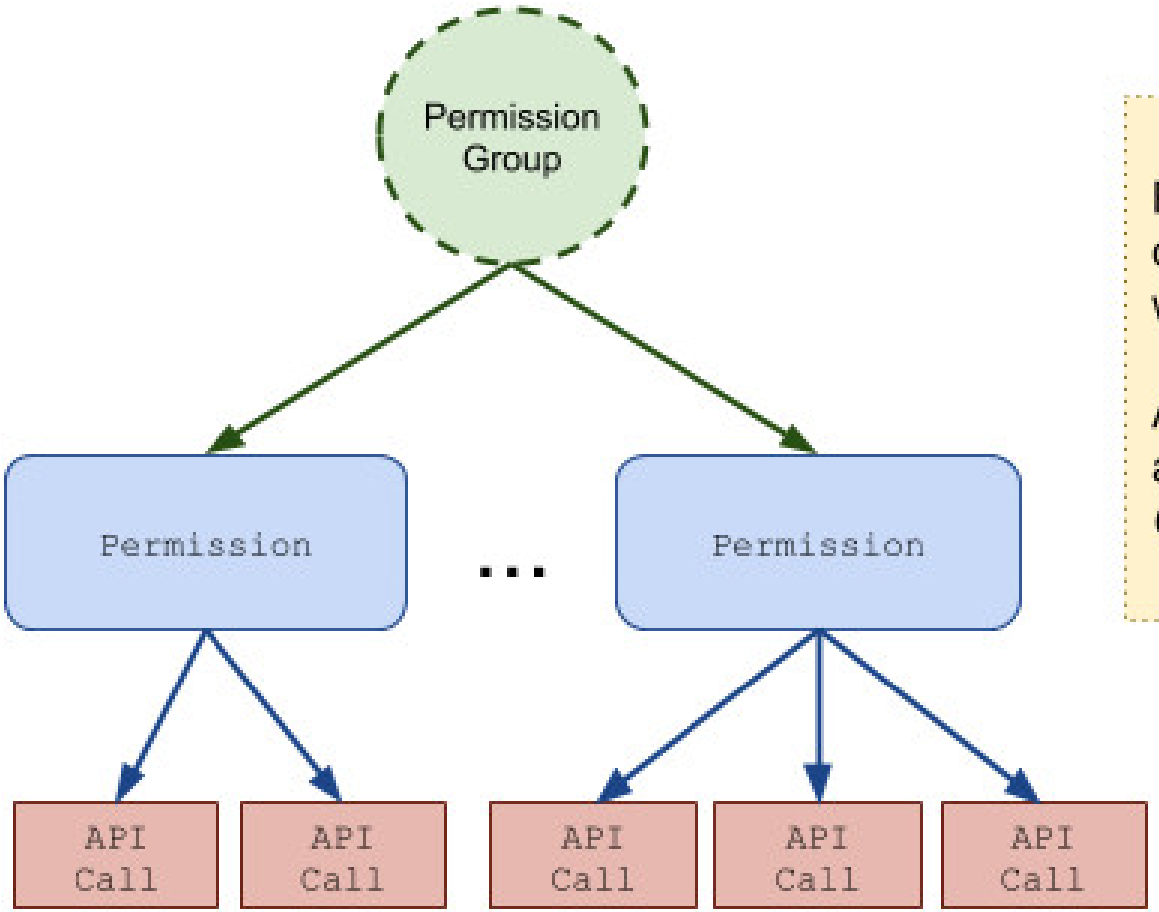
– Special permissions

- 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.

Permission groups

- Permissions are organized into groups related to a device's capabilities or features.
- Under this system, permission requests are handled at the group level and a single permission group corresponds to several permission declarations in the app manifest.
- For example, the SMS group includes both the READ_SMS and the RECEIVE_SMS declarations.
- Grouping permissions in this way enables the user to make more meaningful and informed choices, without being overwhelmed by complex and technical permission requests.

Cont ...



Related API Calls and access to object properties are associated with a specific permission request.

And related permission requests are rolled up into a *Permissions Group*.

Cont ...

- All dangerous Android permissions belong to permission groups.
- Any permission can belong to a permission group regardless of protection level.
- However, a permission's group only affects the user experience if the permission is dangerous.

Check for permissions

- If your app needs a dangerous permission, you must check whether you have that permission every time you perform an operation that requires that permission.
- To check if you have a permission, call the ContextCompat.checkSelfPermission() method.
- For example, this snippet shows how to check if the activity has permission to write to the calendar:

Cont ...

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

Cont ...

- If the app has the permission, the method returns PERMISSION_GRANTED, and the app can proceed with the operation.
- If the app does not have the permission, the method returns PERMISSION_DENIED, and the app has to explicitly ask the user for permission.

Request permissions

- When your app receives PERMISSION_DENIED from checkSelfPermission(), you need to prompt the user for that permission.
- Android provides several methods you can use to request a permission, such as requestPermissions()

Cont ...

```
// Permission is not granted
    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
            Manifest.permission.READ_CONTACTS)) {
    } else {
        // No explanation needed; request the permission
       ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);
        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
```

App permissions best practices

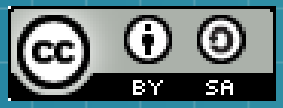
- Only use the permissions necessary for your app to work
  - Depending on how you are using the permissions, there may be another way to do what you need (system intents, identifiers, backgrounding for phone calls) without relying on access to sensitive information.
- Pay attention to permissions required by libraries
  - When you include a library, you also inherit its permission requirements.
  - You should be aware of what you're including, the permissions they require, and what those permissions are used for.

Cont ...

- Be transparent.
  - When you make a permissions request, be clear about what you’re accessing, and why, so users can make informed decisions.
  - Make this information available alongside the permission request including install, runtime, or update permission dialogues.

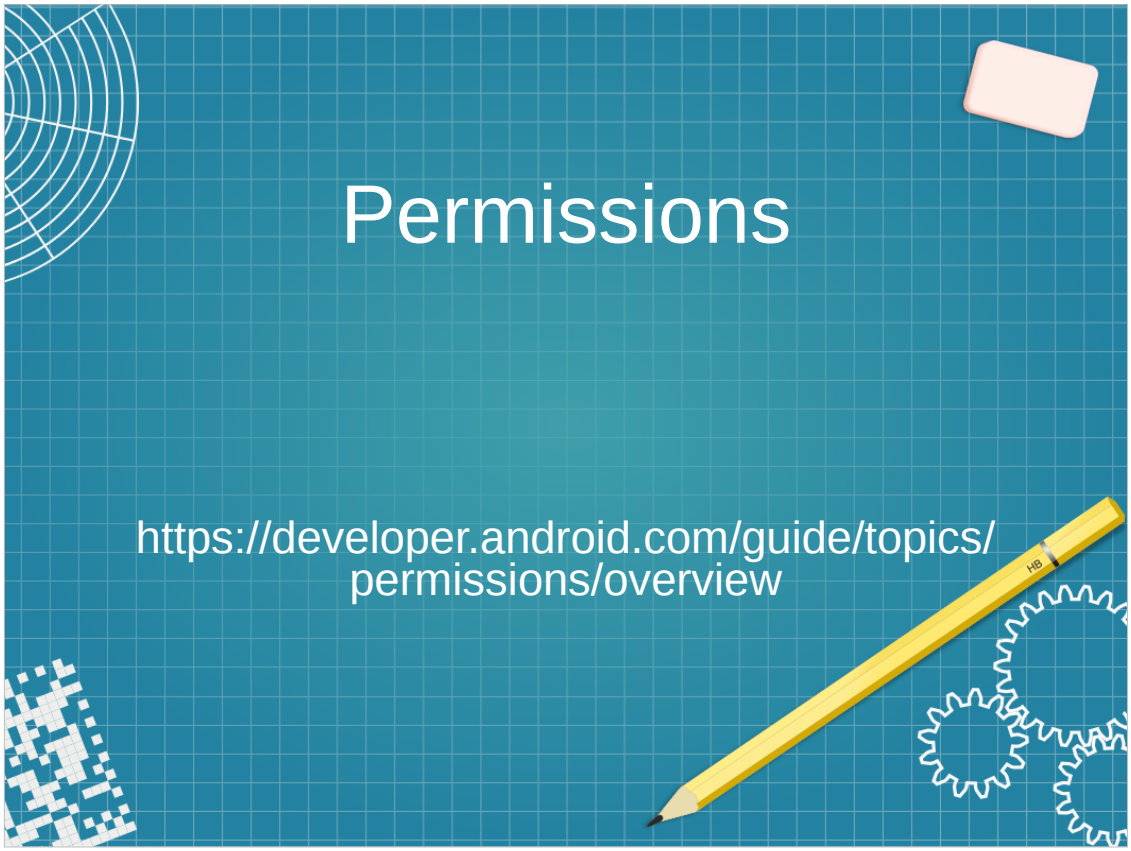
Cont ...

- Make system accesses explicit
  - Providing continuous indications when you access sensitive capabilities (for example, the camera or microphone) makes it clear to users when you’re collecting data and avoids the perception that you're collecting data surreptitiously.



Permissions

https://developer.android.com/guide/topics/permissions/overview



Cont ...

- 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.

Cont ...

- 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"/>
</manifest>
```

Cont ...

- 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.

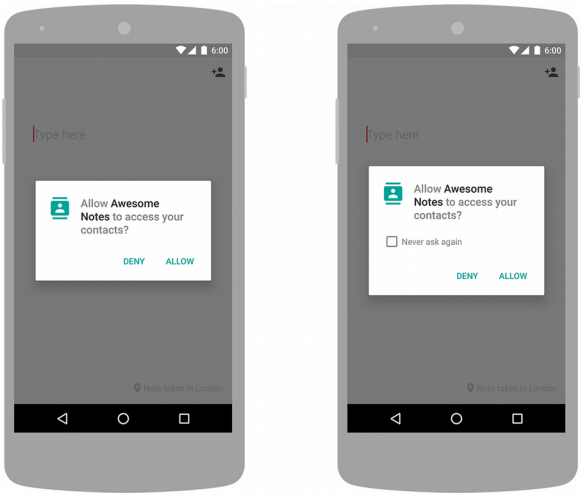
Request prompts for dangerous permissions

- 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.

Cont ...

- 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.
- 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

Cont ...


6:00
Type here
Allow Awesome Notes to access your contacts?
DENY
ALLOW
Note taken in London
6:00
Type here
Allow Awesome Notes to access your contacts?
Never ask again
DENY
ALLOW
Note taken in London

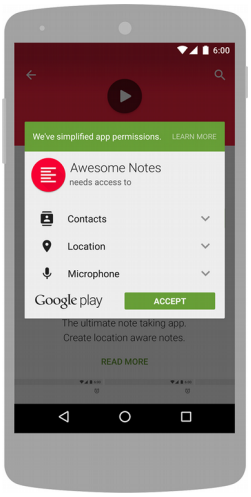
Cont ...

- 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.

Cont ...

- 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 (see figure 2).

Cont ...


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

Cont ...

- 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.

Permission enforcement

- 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.

Cont ...

- 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.

Cont ...

- 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.

Cont ...

- Note that both a receiver and a broadcaster can require a permission.
- When this happens, both permission checks must pass for the intent to be delivered to the associated target.
- For more information, see Restricting broadcasts with permissions.

Cont ...

- Content Provider permission enforcement
  - Permissions applied using the android:permission attribute to the <provider> tag restrict who can access the data in a ContentProvider.
  - (Content providers have an important additional security facility available to them called URI permissions which is described next.) 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.

Cont ...

- The permissions are checked when you first retrieve a provider (if you don't have either permission, a SecurityException is thrown), 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.

Cont ...

- 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.

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.

Cont ...

- 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.

Cont ...

- 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.
- 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.

Cont ...

- 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.

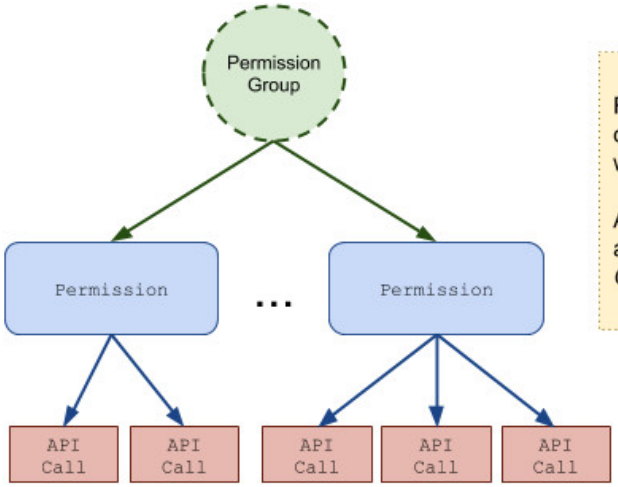
– Special permissions

- 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.

Permission groups

- Permissions are organized into groups related to a device's capabilities or features.
- Under this system, permission requests are handled at the group level and a single permission group corresponds to several permission declarations in the app manifest.
- For example, the SMS group includes both the READ_SMS and the RECEIVE_SMS declarations.
- Grouping permissions in this way enables the user to make more meaningful and informed choices, without being overwhelmed by complex and technical permission requests.

Cont ...



Related API Calls and access to object properties are associated with a specific permission request.

And related permission requests are rolled up into a *Permissions Group*.

Cont ...

- All dangerous Android permissions belong to permission groups.
- Any permission can belong to a permission group regardless of protection level.
- However, a permission's group only affects the user experience if the permission is dangerous.

Check for permissions

- If your app needs a dangerous permission, you must check whether you have that permission every time you perform an operation that requires that permission.
- To check if you have a permission, call the ContextCompat.checkSelfPermission() method.
- For example, this snippet shows how to check if the activity has permission to write to the calendar:

Cont ...

```
if
(ContextCompat.checkSelfPermission(thisActivi
ty, Manifest.permission.WRITE_CALENDAR)
        !=
PackageManager.PERMISSION_GRANTED) {
    // Permission is not granted
}
```

Cont ...

- If the app has the permission, the method returns PERMISSION_GRANTED, and the app can proceed with the operation.
- If the app does not have the permission, the method returns PERMISSION_DENIED, and the app has to explicitly ask the user for permission.

Request permissions

- When your app receives PERMISSION_DENIED from checkSelfPermission(), you need to prompt the user for that permission.
- Android provides several methods you can use to request a permission, such as requestPermissions()

Cont ...

```
// Permission is not granted
    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
            Manifest.permission.READ_CONTACTS)) {
    } else {
        // No explanation needed; request the permission
       ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);
        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
```

App permissions best practices

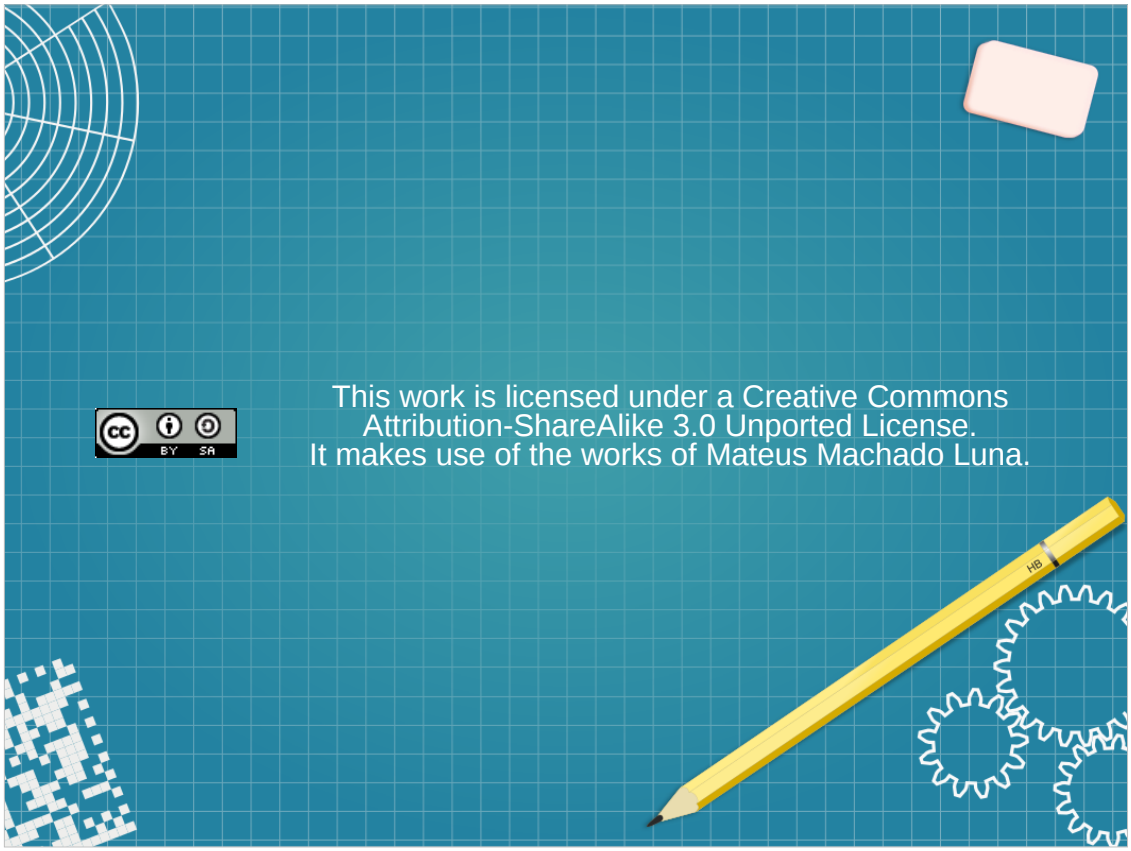
- Only use the permissions necessary for your app to work
  - Depending on how you are using the permissions, there may be another way to do what you need (system intents, identifiers, backgrounding for phone calls) without relying on access to sensitive information.
- Pay attention to permissions required by libraries
  - When you include a library, you also inherit its permission requirements.
  - You should be aware of what you're including, the permissions they require, and what those permissions are used for.

Cont ...

- Be transparent.
  - When you make a permissions request, be clear about what you’re accessing, and why, so users can make informed decisions.
  - Make this information available alongside the permission request including install, runtime, or update permission dialogues.

Cont ...

- Make system accesses explicit
  - Providing continuous indications when you access sensitive capabilities (for example, the camera or microphone) makes it clear to users when you’re collecting data and avoids the perception that you're collecting data surreptitiously.


This work is licensed under a Creative Commons
Attribution-ShareAlike 3.0 Unported License.
It makes use of the works of Mateus Machado Luna.
BY SA
HB