Security and Permissions

http://javat.in

Security and Permissions

- A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user.
- This includes reading or writing the user's private data (such as contacts or e-mails), reading or writing another application's files, performing network access, keeping the device awake, etc.
- The permissions required by an application are declared statically in that application, so they can be known up-front at install time and will not change after that.

User IDs and File Access

- Each Android package (.apk) file installed on the device is given its own unique Linux user ID.
- This user ID is assigned to it when the application is installed on the device, and remains constant for the duration of its life on that device.
- The code of any two packages can not normally run in the same process, since they need to run as different Linux users.
  - Use the sharedUserId attribute in the AndroidManifest.xml's manifest tag of each package to have them assigned the same user ID.

User IDs and File Access

- Any data stored by an application will be assigned that application's user ID, and not normally accessible to other packages.
- When creating a new file with getSharedPreferences(String, int), openFileOutput(String, int), or openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory), you can use the MODE_WORLD_READABLE and/or MODE_WORLD_WRITEABLE flags to allow any other package to read/write the file.
- When setting these flags, the file is still owned by your application, but its global read and/or write permissions have been set appropriately so any other application can see it.

Using Permissions

- A basic Android application has no permissions associated with it, meaning it can not do anything that would adversely impact the user experience or any data on the device.
- To make use of protected features of the device, you must include in your AndroidManifest.xml one or more <uses-permission> tags declaring the permissions that your application needs.

Using Permissions - Example

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
       package="com.android.app.myapp" >

<uses-permission
    android:name="android.permission.RECEIVE_SMS" />

    </manifest>
```

Using Permissions

- At application install time, permissions requested by the application are granted to it by the package installer, based on checks against the signatures of the applications declaring those permissions and/or interaction with the user.
- *No* checks with the user are done while an application is running.
- permission failure will result in a SecurityException being thrown back to the application.
  - However, this is not guaranteed to occur everywhere. E.g. sendBroadcast(Intent)
  - However a permission failure will be printed to the system log.
- The permissions provided by the Android system can be found at Manifest.permission.

Using Permissions

- A particular permission may be enforced at a number of places during your program's operation:
  - At the time of a call into the system, to prevent an application from executing certain functions.
  - When starting an activity, to prevent applications from launching activities of other applications.
  - Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you.
  - When accessing and operating on a content provider.
  - Binding or starting a service.

Enforcing Permissions in AndroidManifest.xml

- High-level permissions restricting access to entire components of the system or application can be applied through your AndroidManifest.xml.
- All that this requires is including an android:permission attribute on the desired component, naming the permission that will be used to control access to it.

Enforcing Permissions in AndroidManifest.xml

- **Activity** permissions:
  - Restrict who can start the associated activity.
  - The permission is checked during Context.startActivity() and Activity.startActivityForResult(); if the caller does not have the required permission then SecurityException is thrown from the call.
- **Service** permissions
  - 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 does not have the required permission then SecurityException is thrown from the call.