Android Security Mechanisms

Lecture 4

Security of Mobile Devices

2022

Outline

Security Goals

- Protect application and user data
- Protect system resources
- Isolate app from the system, other apps and the user

Key Security Features

- Linux kernel security
- Application sandbox
- Signed applications
- Permissions
- Secure IPC

Outline

Sandboxing

- Mechanism based on UIDs
  - Isolate applications
  - Unique UID assigned to each application at installation time
  - Dedicated process running as that UID
  - Dedicated directory - only that UID has rwx permissions
- Process-level and file-level sandbox
- Enforced at kernel-level

Application Files Permissions

- Each app - dedicated data directory
  - rwx permissions only for that app UID/GID
  - Other apps cannot access those files

UIDs

- Well-defined UIDs for system services and daemons
- User `root` UID 0
  - Very few daemons under root UID 0
- User *system* UID 1000
  - Special priviledges
- UIDs for system services start at 1000
- App UIDs start at 10000

Shared UID

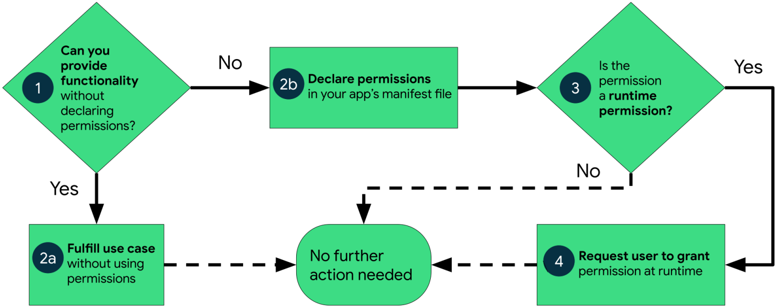
- Apps with the same UID
  - Share files
  - Run in the same process
- Frequently used by system apps
  - Not recommended for non-system apps
- Implementation:
  - Signed with the same code signing key
  - `sharedUserId` attribute of `<manifest>`

Outline

Android Permissions

- By default, applications cannot perform operations to impact other apps, the OS or the user
- Permission - the ability to perform a particular operation
- Built-in permissions documented in the platform API reference
  - Defined in the `android` package
- Custom permissions - defined by system or user apps
- `pm list permissions`

Workflow


1
Can you provide functionality without declaring permissions?
No
2b
Declare permissions in your app's manifest file
3
Is the permission a runtime permission?
Yes
Yes
No
2a
Fulfill use case without using permissions
No further action needed
4
Request user to grant permission at runtime

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

Android Permissions

- Defining package + .permission + name
  - `android.permission.REBOOT`
  - `com.android.launcher3.permission.RECEIVE_LAUNCH_-BROADCASTS`
- Apps request permissions in `AndroidManifest.xml`

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

- Install-time & runtime permissions

Android Permissions

- Permissions handled by the PackageManager service
- Central database of installed packages
- Programatically access package information from `android.content.pm.PackageManager`
  - `getPackageInfo()` returns `PackageInfo` instance

Types of Permissions

- Previous protection levels: normal, dangerous, signature, signatureOrSystem
- Current types: install-time, runtime, special permissions
- Type => risk, scope of the permission

Install-time Permissions

- Granted at installation time
- Minimally affect other applications
- Should be declared on Google Play
- Examples: access network, view network connections, keep phone awake.

Normal and Signature Permissions

- Subtypes of install-time permissions: normal & signature
- Normal
  - Don't affect the system, other apps, user's privacy
  - Normal protection level
- Signature
  - Apps signed with the same certificate as the app that defined the permission
  - Signature protection level

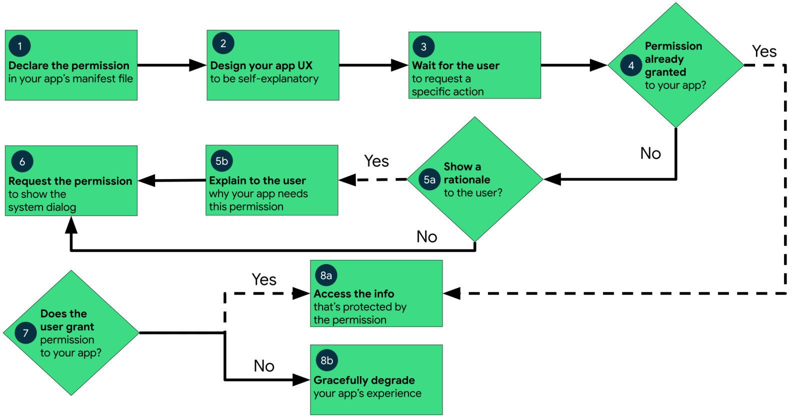
Runtime Permissions

- Dangerous permissions
- Access to restricted data and restricted actions
- Need to be requested from the user
- System prompt used for asking the user to allow or deny
- Needed for accessing user's private data
- Dangerous protection level

Special Permissions

- Defined only by the platform or OEMs
- Powerful and dangerous permissions
- Drawing over other apps
- System Settings $->$ Special App Access
- Allow operations for certain applications
- Appop protection level

Workflow for Runtime Permissions



1 Declare the permission in your app's manifest file
2 Design your app UX to be self-explanatory
3 Wait for the user to request a specific action
4 Permission already granted to your app?
Yes
No
5a Show a rationale to the user?
Yes
No
5b Explain to the user why your app needs this permission
6 Request the permission to show the system dialog
7 Does the user grant permission to your app?
Yes
No
8a Access the info that's protected by the permission
8b Gracefully degrade your app's experience

Source: `https://developer.android.com/training/permissions/requesting`

Permission Enforcement

- A permission can be enforced in a number of places
  - Making a call into the system
  - Starting an activity
  - Starting and binding a service
  - Sending and receiving broadcasts
  - Accessing a content provider

Kernel-Level Enforcement

- Access to regular files, device nodes and local sockets managed by the Linux kernel, based on UID, GID
- Permissions are mapped to supplementary GIDs
- Built-in permission mapping in `/etc/permission/platform.xml`

Kernel-Level Enforcement

- Example:
  - INTERNET permission associated with GID inet
  - Only apps with INTERNET permission can create network sockets
  - The kernel verifies if the app belongs to GID inet

Framework-Level Enforcement

- Static permission enforcement
  - System keeps track of permissions associated to each app component
  - Checks whether callers have the required permission before allowing access
  - Enforcement by runtime environment
  - Less flexible

Static Enforcement

- An app tries to call a component of another app - intent
- Target component - `android:permission` attribute
- Caller - `<uses-permission>`
- Activity Manager
  - Resolves intent
  - Checks if target component has an associated permission
  - Delegates permission check to Package Manager
- If caller has necessary permission, the target component is started
- Otherwise, a SecurityException is generated

Framework-Level Enforcement

- Dynamic permission enforcement
  - Components check to see if the caller has the necessary permissions
  - Decisions made by each component, not by runtime environment
  - More fine-grained access control
  - More operations in components

Dynamic Enforcement

- Helper methods in `android.content.Context` class to perform permission check
- `checkPermission(String permission, int pid, int uid)`
  - Returns `PERMISSION_GRANTED` or `PERMISSION_DENIED`
  - For root and system, permission is automatically granted
  - If permission is declared by calling app, it is granted
  - Deny for private components
  - Queries the Package Manager
- `enforcePermission(String permission, int pid, int uid, String message)`
  - Throws SecurityException with message if permission is not granted

Activity and Service Permission Enforcement

- Permission checks for activities
  - Intent is passed to `Context.startActivity()` or `startActivityForResult()`
  - Resolves to an activity that declares a permission

Activity and Service Permission Enforcement

- Permission checks for services
  - Intent passed to `Context.startService()` or `stopService()` or `bindService()`
  - Resolves to a service that declares a permission
- If caller does not have the necessary permission, generates SecurityExceptions

Content Provider Permission Enforcement

- Protect the whole component or a particular exported URI
- Different permissions for reading and writing
- Read permission - `ContentResolver.query()` on provider or URI
- Write permission - `ContentResolver.insert()`, `update()`, `delete()` on provider or URI
- Synchronous checks

Broadcast Permission Enforcement

- Receivers may be required to have a permission
  - `Context.sendBroadcast(Intent intent, String receiverPermission)`
  - Check when delivering intent to receivers
  - No permission - broadcast not received, no exception

Broadcast Permission Enforcement

- Senders may need to have a permission to send a broadcast
  - Specified in manifest or in `registerReceiver`
  - Checked when delivering broadcast
  - No permission - no delivery, no exception
- 2 checks for each delivery: for sender and receiver

Add Permission in Manifest

- On all Android versions

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

    <uses-permission android:name="android.permission.SEND_SMS"/>
    <!-- other permissions go here -->

    <application ...>
        ...
    </application>
</manifest>
```

Check for Permissions

- Dangerous permissions must be granted by the user
- Check if app has dangerous permission before performing operation
  - Permissions can be revoked from Android 6
- `ContextCompat.checkSelfPermission()`
  - Returns `PERMISSION_GRANTED` - operation can be performed
  - Returns `PERMISSION_DENIED` - permission must be requested

Request Permissions

- When `checkSelfPermission()` returns `PERMISSION_DENIED`
- Method `ActivityCompat.requestPermissions()`
  - Permission
  - Request code
- App needs to request every permission even if user grants whole group

Request Permissions

- Dialog box shown by the system
  - Requests permission group
  - Cannot be changed by the app
  - Explanation provided before requesting permissions
- Asynchronous
  - Response received in callback

Request Permissions - Example

```
if (ContextCompat.checkSelfPermission(thisActivity,
        Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {

    // Permission is not granted

    ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);

    // Results in received in callback

} else {
    // Permission has already been granted
}
```

Handle Permissions Request Response

- User responds -> system calls `onRequestPermissionsResult()` callback
    - App must override this method to receive results
    - Request code, permissions and results received as parameters
    - Check request code
    - Check if permission is granted

Handle Permissions Request Response

- Permission granted
  - Do permission-related task
- Permission denied
  - Disable functionality
  - Announce user

Handle Permissions Request Response - Example

```
@Override
public void onRequestPermissionsResult(int requestCode,
        String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // permission was granted, do task
            } else {
                // permission denied, disable functionality
            }
            return;
        }
    }
}
```

Custom Permissions

- Declared by apps
- Checked statically by the system or dynamically by the components
- Declared in `AndroidManifest.xml`

Custom Permissions - Example

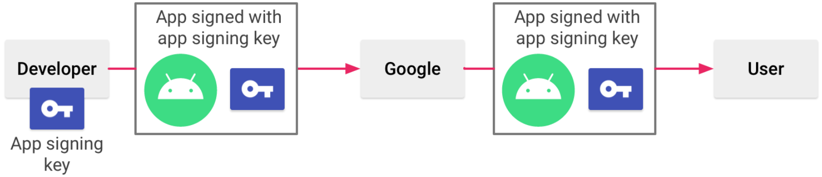
```
<permission-tree
        android:name="com.example.app.permission"
        android:label="@string/example_permission_tree_label" />

<permission-group
        android:name="com.example.app.permission-group.TEST_GROUP"
        android:label="@string/test_permission_group_label"
        android:description="@string/test_permission_group_desc" />

<permission
        android:name="com.example.app.permission.PERMISSION1"
        android:label="@string/permission1_label"
        android:description="@string/permission1_desc"
        android:permissionGroup="com.example.app.permission-group.TEST_GROUP"
        android:protectionLevel="signature" />
```

Outline

Signing Applications

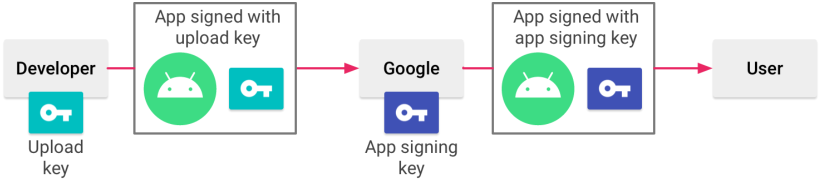
- Identify the developer of the application
- All apps must be signed
- Unsigned apps rejected by Google Play and package installer
- Developer is accountable for the behavior of the app
- Each apk signed with a certificate
- Identifies the developer of the application

Manage your own App Signing Key


Developer
App signing key
App signed with app signing key
Google
App signed with app signing key
User

Source: `https://developer.android.com/studio/publish/app-signing`

Let Google Manage the App Signing Key



Source: `https://developer.android.com/studio/publish/app-signing`

Signature Verification

- Package Manager verifies signature
- At installation time
- Verification uses the public key in the certificate included in the apk
- Grants package integrity
- System applications signed with the platform key

Outline

Bibliography

- Android Security Internals, Nikolay Elenkov
- `https://source.android.com/security/`
- `https://developer.android.com/guide/topics/permissions/overview`
- `https://developer.android.com/training/permissions/requesting`
- `https://developer.android.com/studio/publish/app-signing`

Keywords

- Permissions
- Protection levels
- Install-time permissions
- Runtime permissions
- Special permissions
- Signature permissions
- Static enforcement
- Dynamic enforcement
- Custom permissions
- Signed applications
- Upload key
- App signing key