Android Security Mechanisms

Lecture 4

Security of Mobile Devices

2023

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

SMD

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
- Cannot directly share files with other apps

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

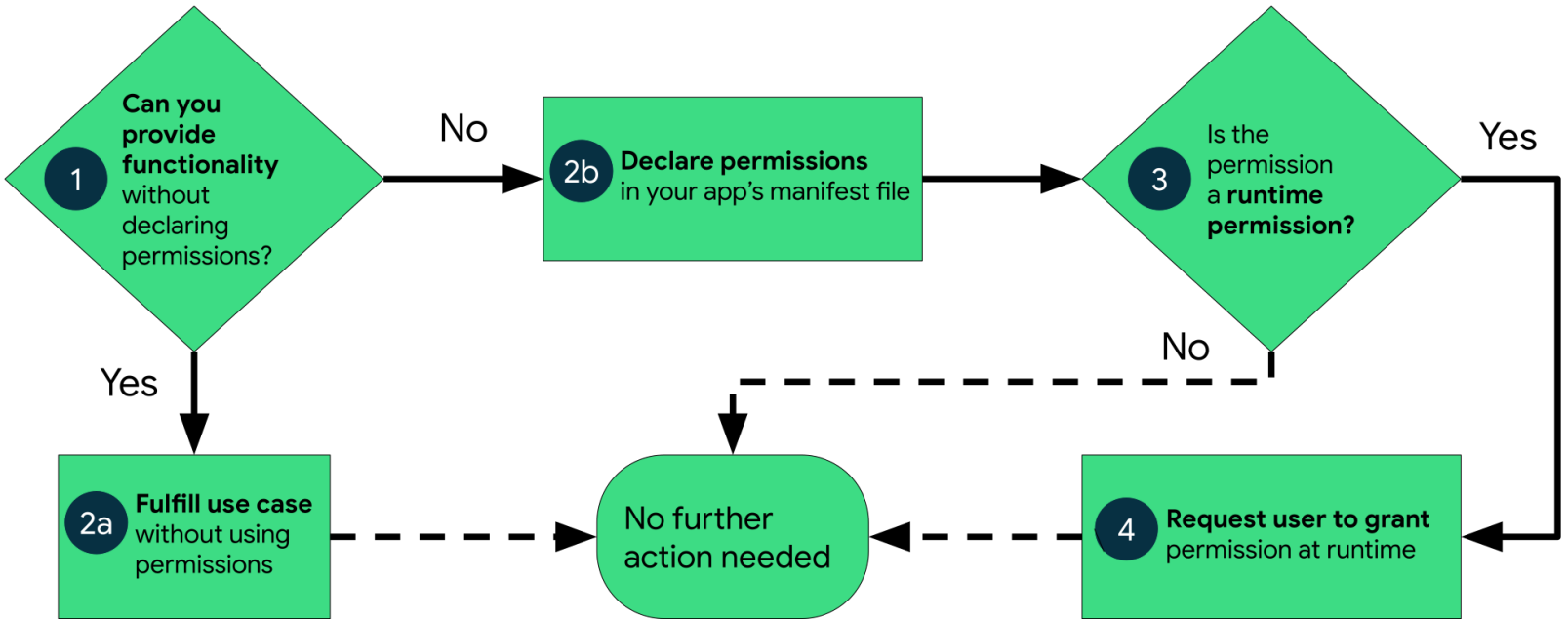
SMD

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

Workflow



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`
- Declare permissions in `AndroidManifest.xml`

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

- Install-time & runtime permissions

Types of Permissions

- 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

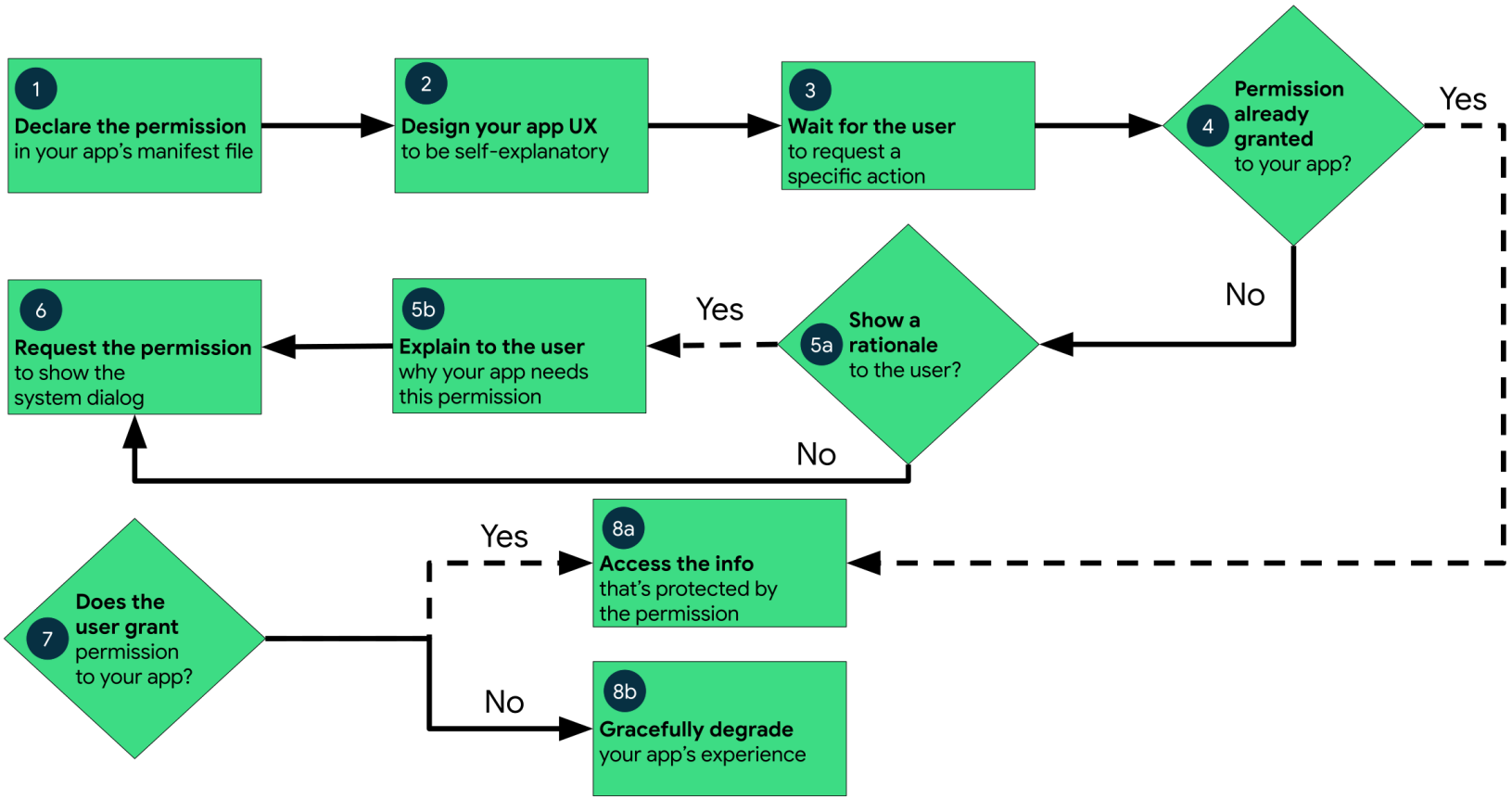
Runtime Permissions

- Dangerous permissions
- Access to restricted data and restricted actions
- Need to be requested from the user at runtime
- 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



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

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

- Runtime permissions must be granted by the user
- Check if app has runtime permission before performing operation
  - Permissions can be revoked from Settings
- `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()`
  - Permissions list
  - Request code
- For example:

```
ActivityCompat.requestPermissions(this, new String[]
{android.Manifest.permission.ACCESS_COARSE_LOCATION,
android.Manifest.permission.READ_PHONE_STATE,
android.Manifest.permission.WRITE_EXTERNAL_STORAGE}, MY_REQ);
```

Request Permissions

- Dialog box shown by the system
  - Requests permissions
  - 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

- Defined by apps
- Checked statically by the system or dynamically by the components
- Defined 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" />
```

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 Enforcement

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

Framework-Level Enforcement - 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 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

Framework-Level Enforcement - Dynamic Enforcement

- `Context.checkCallingPermission(String permission)`
  - IPC call to a service
- `Context.checkPermission(String permission, int pid, int uid)`
- `PackageManager.checkPermission(String permission, String package)`

Activity Permission Enforcement

- `android:permission` of `<activity>`
- Permission is checked when calling:
  - `Context.startActivity()`
  - `Activity.startActivityForResult()`
- `SecurityException` if caller does not have the permission

Service Permission Enforcement

- `android:permission` of `<service>`
- Permission is checked when calling:
  - `Context.startService()`
  - `Context.stopService()`
  - `Context.bindService()`
- `SecurityException` if caller does not have the permission

Broadcast Permission Enforcement

- `android:permission` of `<receiver>`
- Or supply permission to `Context.registerReceiver()`
- Permission is checked after `Context.sendBroadcast()`
- If caller does not have the permission
  - The broadcast will not be delivered
  - An exception will not be thrown

Broadcast Permission Enforcement

- Restrict which receivers can receive a broadcast
- Supply permission to `Context.sendBroadcast()`
- If receiver does not have the permission
  - The broadcast will not be delivered
  - An exception will not be thrown

Broadcast Permission Enforcement

- Both sender and receiver may require permissions
- Broadcast is delivered if both permission checks pass

Content Provider Permissions

- Single read-write provider-level permission
  - `android:permission`
- Separate read and write provider-level permission
  - `android:readPermission` and `android:writePermission`
- Path-level permission
  - `<path-permission>` to specify URI
  - Permissions for specific URIs
- Temporary permission
  - Delegate temporary access to an application
  - `android:grantUriPermissions` or `<grant-uri-permission>`

Content Provider Temporary Permissions

- Give another app temporary permissions for an URI
- Intent
  - URI
  - `Intent.FLAG_GRANT_READ_URI_PERMISSION`
  - `Intent.FLAG_GRANT_WRITE_URI_PERMISSION`
- `startActivity(Intent)`

Content Provider Permission Enforcement

- Permission checked when another app makes operations on the provider
- Read permission - `ContentResolver.query()`
- Write permission - `ContentResolver.insert()`, `update()`, `delete()`
- `SecurityException` if caller does not have the permission

Outline

Signing Applications

- Identify the developer of the application
- All apps must be signed
- Unsigned apps rejected by Google Play and package installer
- Each apk signed with a certificate
- Identifies the developer of the application
- Securing updates

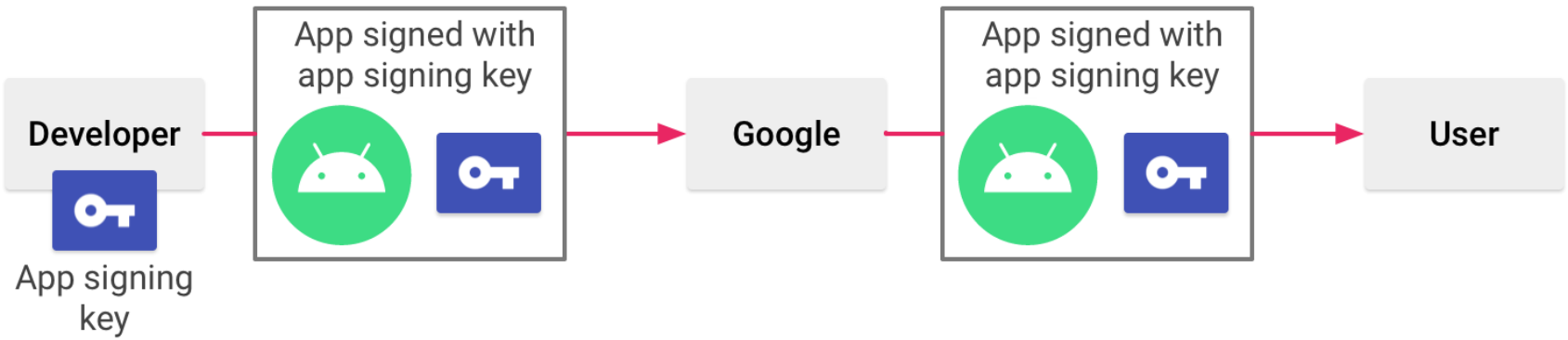
Signing Applications

- Two methods:
  - Using a personal app signing key
  - Using an upload key

Personal App Signing Key

- Developer generates, stores the key
- Strong password for keystore
- Sign apk and upload
- Problems: key is lost or gets stolen

Personal App Signing Key

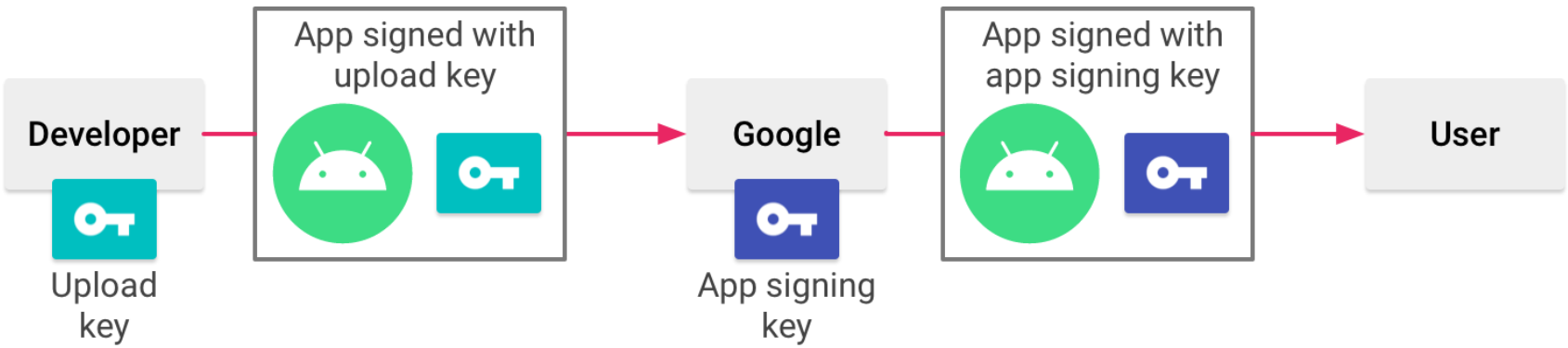


Source: `https://developer.android.com/studio/publish/app-signing`

Upload Key

- Developer generates, stores the upload key
- Sign apk and upload
- Google generates app singing key and signs your app
- Google Key Management System
- Upload key may be reset if it's lost or stolen
- Safer method

Upload 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

SMD

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/training/permissions/restrict-interactions
- 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