Writing Android system apps

for Android 13 and later

Chris Simmonds

- Normal apps are very restricted in what they can do
- But system apps can do anything ... can’t they?
- Let’s go into the detail
    - what are system apps?
    - what are they used for?
    - how do you create them in an AOSP build?

What are system apps?

- Pre-installed in one of the read-only partitions
    - `/system`, `/system_ext`, `/product`
- Signed by the OEM
- Cannot be uninstalled, but they can be disabled
- Can be updated from the app store
- Most, but not all, have super powers

Identifying system apps at run-time

A convenient method is to look at the SELinux context, which you can do using the ps command with A and Z:

```
$ ps -AZ
LABEL                               USER       PID  PPID  S NAME
u:r:platform_app:s0:c512,c768       u0_a98     753   373  S com.android.systemui
u:r:priv_app:s0:c512,c768           u0_a91    1090   373  S com.android.launcher3
u:r:system_app:s0                   system    1890   373  S com.android.localtransport
u:r:untrusted_app_25:s0:c512,c768   u0_a86    2282   373  S com.android.deskclock
```

Almost all Android apps have one of the following contexts:

- `untrusted_app`: a regular app, either pre-installed or user-installed
- `platform_app`: a pre-installed app, signed with the platform (OEM) keys
- `system_app`: a platform app with UID = 1000 (system)
- `priv_app`: a platform app or system app which can be granted permissions with protection level `signature|privileged`

A brief intro to SELinux

- SELinux is a Mandatory Access Control (MAC) mechanism used in Linux to give fine-grained access control to Linux processes
- In Android, it is the primary means of controlling the files, sockets, system properties that an app can access
- Each Linux process has a context
  - each app runs as a Linux process
- Each file, socket, etc., also has a context
- The SELinux Policy (sepolicy) is the set of rules that control who can do what
  - the policy is compiled into binary format and stored in the read-only storage `/system/etc/selinux`, `/system_ext/etc/selinux`, etc
  - can only be changed by the OEM

Package manager flags

Another way to see app capabilities is via package manager flags

```
$ dumpsys package packages
```

For example here is part of the dump for com.android.deskclock, edited to show only relevant fields:

```
  Package [com.android.deskclock] (98afa39):
    appId=10065
    codePath=/product/app/DeskClock
    pkgFlags=[ SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA ]
```


pkgFlags

Documented in
https://developer.android.com/guide/topics/manifest/application-element

Some examples:

```
HAS_CODE               The application contains DEX code (*)
SYSTEM                 Pre-installed
ALLOW_CLEAR_USER_DATA  The application may reset user data
```

(*) Does not say anything about native (C/C++) code. For example, if an APK contained no DEX code but did have native code, it would not have the HAS_CODE flag

An untrusted system app

Android.bp

```
android_app {
    name: "system-app-untrusted",
    srcs: ["src/**/*.java"],
    resource_dirs: ["res"],
    sdk_version: "current",

    // install in /product/app
    product_specific: true,
}
```

SE context

```
u:r:untrusted_app:s0:c71,c256,c512,c768 u0_a71    com.sirius.system_app_untrusted
```

pkgFlags

```
  Package [com.sirius.system_app_untrusted] (80eae5c):
    appId=10071
    codePath=/product/app/system-app-untrusted
    pkgFlags=[ SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
```

Platform apps

- Platform apps have SELinux context `platform_app`
- Signed with the OEM platform keys
  - Android.bp has `certificate: "platform",`
- Installed into `/system` or `system_ext`
  - Android.bp has `system_specific: true,` or `system_ext_specific: true,`
- Call low-level platform APIs
  - Android.bp has `platform_apis: true,`
- Super powers:
  - can call platfrom APIs
  - not killed

A platform app

Android.bp

```
android_app {
    name: "system-app-platform",
    srcs: ["src/**/*.java"],
    resource_dirs: ["res"],

    // Platform apps must be installed into /system/app or /system_ext/app
    system_ext_specific: true,

    // Sign with platform keys
    certificate: "platform",

    // Use low level platform apis
    platform_apis: true,
    // Note: platform_apis is not compatible with sdk_version: you can have one or the other but not both
}
```

SE context

```
u:r:platform_app:s0:c512,c768  u0_a88      com.sirius.system_app_platform
```

pkgFlags

```
  Package [com.sirius.system_app_platform] (4e001ad):
    appId=10088
    codePath=/system_ext/app/system-app-platform
    pkgFlags=[ SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
```

Signing apps in AOSP

https://source.android.com/docs/core/ota/sign_builds

AOSP has a set of default keys in `build/target/product/security/`

```
$ ls build/target/product/security/
Android.bp                   media.pk8              platform.pk8          shared.x509.pem
bluetooth.pk8                media.x509.pem         platform.x509.pem     testkey.pk8
bluetooth.x509.pem           networkstack.pk8       README                testkey.x509.pem
cts_uicc_2021.pk8            networkstack.x509.pem  sdk_sandbox.pk8
cts_uicc_2021.x509.pem       nfc.pk8                sdk_sandbox.x509.pem
fsverity-release.x509.der    nfc.x509.pem           shared.pk8
```

The APKs created by AOSP are signed with either

- "testkey": untrusted apps, this is the default
- "platform": platform apps

The other keys have specialized uses that I will briefly cover at the end

Creating keys

The OEM will create their own keys

```
subject='/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com'
mkdir ~/.android-certs
for x in releasekey platform shared media networkstack; do \
    ./development/tools/make_key ~/.android-certs/$x "$subject"; \
  done
```

And sign the APKs generated by the AOSP build with these keys

```
$ make dist
$ sign_target_files_apks \
-o \    # explained in the next section
--default_key_mappings ~/.android-certs out/dist/*-target_files-*.zip \
signed-target_files.zip
```

Platform APIs

Platform APIs are annotated with `@SystemApi, and @hide`

They are not exported in `android.jar`, which is part of the Android SDK

They cannot be accessed in a non-platform application

The annotations are described in `frameworks/libs/modules-utils/java/android/annotation` where you will find `Hide.java, NonNull.java, SystemApi.java` and many others

System apps

- System apps are platform apps with the system user ID
  - AndroidManifest.xml has `android:sharedUserId="android.uid.system"`
- They have SELinux context `system_app`
- Super powers:
  - having UID system means that these apps can behave like `system_server`

A system app

`AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.sirius.system_app_platform"
        android:sharedUserId="android.uid.system">
    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
        <activity android:name="SimpleActivity"
                  android:label="@string/app_name"
                  android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

SE context and user ID (system == Linux UID 1000)

```
u:r:system_app:s0    system      com.sirius.system_app_system
```

pkgFlags

```
  Package [com.sirius.system_app_system] (3485d9b):
    appId=1000
    sharedUser=SharedUserSetting{5b2932b android.uid.system/1000}
    codePath=/system_ext/app/system-app-system
    pkgFlags=[ SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
```

Priv apps

`https://source.android.com/docs/core/permissions/perms-allowlist`

- Priv, or Privileged, apps are granted permissions with protection level `"signature|privileged"`
- To create
  - Android.bp has `privileged: true,`
- Must be allow listed in `privapp-permissions*.xml` - see next slide

privapp allow list

A privileged permission - from `frameworks/base/core/res/AndroidManifest.xml`

```
<!-- Allows mounting and unmounting file systems for removable storage.
<p>Not for use by third-party applications.-->
<permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
    android:protectionLevel="signature|privileged" />
```

Allow listing

- Allow lists are in `[partition]/etc/permissions`
- Files named `privapp-permissions-platform.xml`, `privapp-permissions-[device name].xml`, or `[package name].xml`

Example - this is the section in `/system/etc/permissions/privapp-permissions-platform.xml` that relates to package `com.android.externalstorage`

```
<privapp-permissions package="com.android.externalstorage">
    <permission name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <permission name="android.permission.WRITE_MEDIA_STORAGE"/>
    <permission name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
</privapp-permissions>
```

A priv app 1/3

Android.bp

```
android_app {
    name: "system-app-priv",
    srcs: ["src/**/*.java"],
    resource_dirs: ["res"],

    // Platform apps must be installed into /system/app or /system_ext/app
    system_ext_specific: true,

    // Sign with platform keys
    certificate: "platform",

    // Use low level platform apis
    platform_apis: true,

    // make this a privileged app
    privileged: true,

    required: ["allowed_privapp_com.sirius_system_priv"],
}

// Add this app to the priv-app allow list
prebuilt_etc {
    name: "allowed_privapp_com.sirius_system_priv",
    system_ext_specific: true,
    sub_dir: "permissions",
    src: "com.sirius.system_app_priv.xml",
    filename_from_src: true,
}
```


A priv app 2/3

com.sirius.system_app_priv.xml

```
<permissions>
    <privapp-permissions package="com.sirius.system_app_priv">
    <permission name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    </privapp-permissions>
</permissions>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.sirius.system_app_priv">
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>

    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
        <activity android:name="SimpleActivity"
                  android:label="@string/app_name"
                  android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

A priv app 3/3

SE context

```
u:r:priv_app:s0:c512,c768     com.sirius.system_app_priv
```

pkgFlags

```
Package [com.sirius.system_app_priv] (f788d12):
   appId=10086
   pkg=Package{f3bbce3 com.sirius.system_app_priv}
   codePath=/system_ext/priv-app/system-app-priv
   pkgFlags=[ SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
   install permissions:
     android.permission.MOUNT_UNMOUNT_FILESYSTEMS: granted=true
```

Note that `install permissions` grants `MOUNT_UNMOUNT_FILESYSTEMS`

Persistent apps

- Persistet apps are started by Activity Manager on boot
  - before the `BOOT_COMPLETED` intent is broadcast
- If they fail for some reason, they are automatically restarted
- Have `android:persistent="true"` in `AndroidManifest.xml`

A persistent app

`Android.bp`

```
android_app {
    name: "system-app-persistent",
    srcs: ["src/**/*.java"],
    resource_dirs: ["res"],

    // Platform apps must be installed into /system/app or /system_ext/app
    system_ext_specific: true,

    // Sign with platform keys
    certificate: "platform",
}
```

SE context

```
u:r:platform_app:s0:c512,c768  u0_a88      com.sirius.system_app_persistent
```

pkgFlags

```
  Package [com.sirius.system_app_persistent] (4e001ad):
    appId=10088
    codePath=/system_ext/app/system-app-persistent
    pkgFlags=[ SYSTEM HAS_CODE PERSISTENT ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
```

Updating system apps

- System apps can be updated from the app store - without needing an OTA image update
- The original APK is still in the read-only partition
- But it is shadowed by a copy installed in /data

Example - Youtube updated on my phone

```
Package [com.google.android.youtube] (a62aee4):
  appId=10172
  codePath=/data/app/~~zSv7PXLJbpcvUEDDzVPi0g==/com.google.android.youtube-ukdlVTSv6i07ES9y2fEvdQ==
  resourcePath=/data/app/~~zSv7PXLJbpcvUEDDzVPi0g==/com.google.android.youtube-ukdlVTSv6i07ES9y2fEvdQ==
  pkgFlags=[ SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA UPDATED_SYSTEM_APP ...]
```

Note codePath is on `/data` partition

Also that `pkgFlags` contains `UPDATED_SYSTEM_APP`

Other system app SE contexts

In addition to `untrusted_app`, etc, some system apps have their own special SE contexts

I am not going to cover them any further, but for completeness, here is the list for Android 16, basic Cuttlefish phone (lunch target `lunch aosp_cf_x86_64_phone-bp2a-userdebug`)

```
network_stack       com.android.networkstack.process
bluetooth           com.android.bluetooth
nfc                 com.android.nfc
secure_element      com.android.se
radio               com.android.phone
mediaprovider_app   com.android.providers.media.module
traceur_app         com.android.traceur
mediaprovider       android.process.media
gceservice          com.android.google.gce.gceservice
rkpdapp             com.android.rkpdapp
mediaprovider       com.android.photopicker
```

Can’t I just copy an APK into /system/app?

- There are many tutorials that show how to copy an arbitary APK into /system/app to make it into a system app
  - Example: `https://www.getdroidtips.com/install-third-party-apps-system-apps-android/`
- This used to work on old versions of Android, but not today

Questions?

Slides and video will be posted on https://aospandaaos.github.io/meetup.html

You can always contact me via the https://aosp-devs.org Discord server:
https://discord.gg/hH59SPKYv8