Factory Resets and Obtaining Notifications on Samsung Android Devices

Ryan Johnson
Angelos Stavrou
Kryptowire
USA

ABSTRACT

We have identified software vulnerabilities present in two different system apps in multiple Samsung Android releases. The two vulnerabilities expose systemic problems with the inadequacy of testing for pre-loaded vendor applications. The vulnerable system apps are persistent and cannot be disabled or uninstalled by the user unless the user has root privileges on the device. The first vulnerability can be manipulated to programmatically initiate a factory reset of certain devices from a zero-permission Android app. During a factory reset, the user would most likely lose application data, photos, videos, text messages, and anything else that is not synced or backed up to a separate device. The attack is tested to be successful for certain Android 5.0.2 and 5.1.1 releases for the following Samsung Galaxy devices: S6, S6 Edge, S6 Edge+, and Note 5. The vulnerability is introduced by a system app declaring an application component twice in an app's manifest file, which causes the access requirements of the second declaration to overwrite those of the first declaration. The second vulnerability is the ability to obtain the text of notifications from a zero-permission Android app on 5.0.2 builds for the Samsung Galaxy S6 Edge. This vulnerability was introduced in the initial Android 5.0.2 builds for the Samsung Galaxy S6 Edge devices, but the vulnerability can persist on the device even after the device has been upgraded to an Android 5.1.1 or 6.0.1 build. The vulnerable system app gives a non-existent app the ability to read the notifications from the device, which a third-party app can utilize. This vulnerability allows an unprivileged third-party app to obtain the text of the user's notifications, which tend to contain personal data.

KEYWORDS: Mobile Security, Android, Vulnerabilities, Permission Leakage, Samsung

INTRODUCTION

Currently, mobile devices are being delivered to the end users with a wealth of pre-loaded software from the device manufacturer and other vendors that usually include the telecommunications company that offers the service plans. In many cases, this pre-loaded software runs as a system application that enables higher levels of privilege than what is granted to third-party applications. Moreover, some apps cannot be removed or disabled by the end-user without utilizing third-party software to "root" the device, in essence violating both the device warranty and exposing the user to potentially malicious third-party software. Thus, there is a clear need for the device manufacturers and other vendors to have a more rigorous process for vetting applications before their apps reach end-user devices because any vulnerabilities that they might contain cannot be easily fixed since a major patch that affects millions of devices has to be issued. In the next paragraphs, we provide the technical details that can allow malicious actors to perform privileged activities and disrupt the operation of devices equipped with the vulnerable versions of system applications. These applications exemplify the dangers from manufacturer and vendor applications that contain software flaws that can be translated into security threats.

INITIATING FACTORY RESETS FROM A ZERO-PERMISSION THIRD-PARTY APP

Indeed, as part of our research, we were able to identify Samsung-developed applications that were vulnerable to different attacks. One of the applications is installed on the system partition of certain Samsung Android 5.0.2 and 5.1.1 builds that allows a zero-permission third-party app to factory reset the device by sending a single broadcast intent. A factory reset results in the data and cache partitions being wiped on the device. The user will lose any data that is not synced or backed up to another device. We have confirmed that a vulnerable version of this system app exists on certain Android 5.0.2 and 5.1.1 builds for Samsung Galaxy S6, Samsung Galaxy S6 Edge, Samsung Galaxy S6 Edge+, and Samsung Galaxy Note 5 devices. The entire list containing 82 vulnerable builds we identified for these devices is provided in Appendix 1. The vulnerability is introduced by a system app that declares a

broadcast receiver application component with the same name twice in its *AndroidManifest.xml* file. The first declaration uses a signature or system level custom permission to prevent third-party apps from accessing it. The second declaration of the broadcast receiver is not protected by a custom permission and is exported by default. The second declaration of the broadcast receiver overwrites the access requirements from the first declaration, which makes it accessible to third-party apps, while preserving the intent filters from the first declaration. Permission leakage occurs when an app without a specific permission is able to perform permission-protected functionality or obtain permission-protected data. Researchers (Bagheri, 2015; Chin, 2011, June; Grace, 2012, February; Octeau, 2013) have developed approaches to detect and prevent permission leakage in Android apps. In Android, it is incumbent on the app developer to prevent permission leakage by setting the access requirements for their application components in the app's *AndroidManifest.xml* file. In addition, using a double declaration of an application component in the *AndroidManifest.xml* file where the access requirements of the second declaration overwrite those of the first declaration creates a vulnerability is introduced and explained in this manuscript.

ATTACK METHOD

The vulnerable app has a package name of *com.sec.android.app.servicemodeapp* and has a path of */system/priv-app/serviceModeApp_FB.apk*. This app declares a broadcast receiver application component twice with the same value for the *android:name* attribute in its *AndroidManifest.xml* file. The application component named *ServiceModeAppBroadcastReceiver* can initiate a factory reset of the device. This broadcast receiver declares an intent filter with an action string of *com.samsung.intent.action.SEC_FACTORY_RESET_WITHOUT_FACTORY_UI*. The first declaration of the broadcast receiver is protected by a custom permission named *com.sec.android.app.servicemodeapp.permission.KEYSTRING* that has an *android:protectionLevel* of *signatureOrSystem*. The *KEYSTRING* custom permission is declared in the *serviceModeApp_FB.apk* app's *AndroidManifest.xml* file. A permission-protected application component that has the *android:protectionLevel* attribute set to a value of *signatureOrSystem* restricts applications that can interact with it to applications installed on the system partition or apps that are signed with the same certificate. The second declaration with same application component name is not protected by a custom permission and does not explicitly declare that the application component should not be exported. This second declaration of the application component will be exported by default since it declares at least one intent filter and does not explicitly declare that it should not be exported. Appendix 2 shows the double declaration of the broadcast receiver application with the same name and differing access requirements from the *serviceModeApp_FB.apk* app's *AndroidManifest.xml* file.

We reference the Android Open Source Project (AOSP) Android 6 source code to explain what occurs when a broadcast receiver application component with the same name is registered twice in an app's AndroidManifest.xml file. The *com.android.server.pm.PackageManagerService* class ("PackageManagerService," n.d.) handles the installation, uninstallation, and updating of Android applications. *PackageManagerService* uses the *android.content.pm.PackageParser* class ("PackageParser," n.d.) to parse an Android Package (APK) file including its *AndroidManifest.xml* file. The *PackageManagerService.installPackageLI(InstallArgs, PackageInstalledInfo)* method is used to install apps on the device. This method uses the *PackageParser* class to parse the APK file, which returns a *PackageParser.Package* object. This object contains the data from all the application components that are declared in the app's *AndroidManifest.xml* file. If the APK is a new app and is not replacing a previous package, the *PackageManagerService.installNewPackageLI(PackageParser.Package, int, int, UserHandle, String, String, PackageInstalledInfo)* method is called, which will occur when the *serviceModeApp_FB.apk* app is first installed. This method calls the *PackageManagerService.scanPackageLI(PackageParser.Package, int, int, long, UserHandle)* method. This method then calls the *PackageManagerService.scanPackageDirtyLI(PackageParser.Package, int, int, long, UserHandle)* method, which performs various actions to install the app. It will iterate through a *java.util.ArrayList* object, which contains all the broadcast receivers declared in the app's *AndroidManifest.xml* file from when it was parsed. This *ArrayList* object contains the two instances of the *ServiceModeAppBroadcastReceiver* broadcast receiver. It will add each broadcast receiver, whose type is *PackageParser.Activity*, into a *PackageManagerService.ActivityIntentResolver* object using its *addActivity(PackageParser.Activity, String)* method. The *ActivityIntentResolver.addActivity(PackageParser.Activity, String)* method will add each broadcast receiver into an instance variable named *mActivities* contained within the *PackageManagerService* object. The *mActivities* object has a type of *android.util.ArrayMap* and contains all the activity application components and broadcast receiver application components from apps installed on the device. The *addActivity(PackageParser.Activity, String)* method

will also add the intent filters for the first instance of *ServiceModeAppBroadcastReceiver* broadcast receiver, which is protected by a signature or system level custom permission, using the *addFilter(F)* method of the *com.android.server.IntentResolver* class ("IntentResolver," n.d.) so the intent filters can be resolved and associated with the broadcast receiver by its component name.

The second instance of the *ServiceModeAppBroadcastReceiver* broadcast receiver, which is not protected by a custom permission and is exported by default, will be obtained from the *ArrayList* object containing all broadcast receivers. Again the *ActivityIntentResolver.addActivity(PackageParser.Activity, String)* method is called to process the broadcast receiver. It will be added to the mActivities object that has a type of *android.util.ArrayMap*. The *android.util.ArrayMap* class ("ArrayMap," n.d.) stores key-value pairs that behaves similar to a *java.util.HashMap* object even though the underlying implementation contains an *int* array to hold hashes of the keys (i.e., component name) and a *java.lang.Object* array to hold the values (i.e., the corresponding *PackageParser.Activity* object). The second instance of the *ServiceModeAppBroadcastReceiver* broadcast receiver is added to the *ArrayMap* object using its *put(K key, V value)* method. This will effectively overwrite the previous *PackageParser.Activity* object for the *ServiceModeAppBroadcastReceiver* broadcast component, which has a signature or system level custom permission, with the next *PackageParser.Activity* object for the *ServiceModeAppBroadcastReceiver* broadcast component which is not protected by a custom permission and is exported by default. All of the intent filters for both instances of the *ServiceModeAppBroadcastReceiver* broadcast receivers will have been registered via the *com.android.server.IntentResolver.addFilter(F)* method call that is called for each instance of two broadcast receivers with the same name.

The second instance of the *ServiceModeAppBroadcastReceiver* broadcast receiver will overwrite the first instance of the *ServiceModeAppBroadcastReceiver* broadcast receiver declared in *AndroidManifest.xml* file. The AndroidManifest.xml file is parsed serially from beginning to the end and the second instance of *ServiceModeAppBroadcastReceiver* broadcast receiver is exported and not protected by a custom permission. In addition, the *ArrayMap* object's *put(K key, V value)* method allows for the overwriting due to its behavior to mimic a *HashMap* object. This removes the permission protection from the application component so that any third-party application resident on the device can successfully send a broadcast intent to the *ServiceModeAppBroadcastReceiver* broadcast receiver application component. We are not sure how prevalent this weak programming practice is in general, but some of Samsung's developers made this error, so it may exist in other Android applications. The code to factory reset devices with a vulnerable version of the *serviceModeApp_FB.apk* app is shown in Figure 1.

```
Intent i = new Intent();
i.setComponent(ComponentName.unflattenFromString("com.sec.android.app.servicemodeapp/com.sec.android.app.servicemodeapp.ServiceModeAppBroadcastReceiver"));
i.setAction("com.samsung.intent.action.SEC_FACTORY_RESET_WITHOUT_FACTORY_UI");
sendBroadcast(i);
```

Figure 1: Source code to initiate a factory reset of vulnerable Samsung Android devices.

When this broadcast intent is received by the *ServiceModeAppBroadcastReceiver*, it will call the *ServiceModeAppBroadcastReceiver.DoCPReset(android.content.Context)* method. The *DoCPReset(android.content.Context)* method sends an intent with the action of *android.intent.action.MAIN* to the *com.sec.android.app.servicemodeapp/com.sec.android.app.modemui.activities.ModemReset* service application component with an extra value of *FACTORY* set to a Boolean value of true. This is received by the *ModemReset* service application component within the same app. In its *onStart(android.content.Intent, int)* callback method, the intent is checked to see if the *FACTORY* extra has a Boolean value of true. If this is the case, then the *ModemReset.SendResetCommandToRIL()* method is called. This method will send a raw Original Equipment Manufacturer (OEM) Radio Interface Layer (RIL) request with the function of 12 and a sub-function of 1 with a payload of 2. It sends the raw OEM RIL request using the *com.samsung.android.sec_platform_library.FactoryPhone.invokeOemRilRequestRaw(byte[], android.os.Message)* method.

This OEM RIL request will result in the *android.intent.action.MASTER_CLEAR* broadcast intent being sent. This broadcast intent is received by the *com.android.server.MasterClearReceiver* broadcast receiver application component in the core *android* package. This receiver declares an intent filter to receive intents with the action of *MASTER_CLEAR*. This receiver is protected by the *android.permission.MASTER_CLEAR* permission. The *MASTER_CLEAR* permission has an *android:protectionLevel* of *signatureOrSystem* for Android 5.1.1. The

serviceModeApp_FB.apk app requests the *MASTER_CLEAR* permission in its *AndroidManifest.xml* file and is a system app, so it has the capability to perform a factory reset of the device.

The *com.android.server.MasterClearReceiver* calls the *android.os.RecoverySystem.rebootWipeUserData(android.content.Context)* method. This method sends an ordered broadcast with the action of *android.intent.action.MASTER_CLEAR_NOTIFICATION*. The *MASTER_CLEAR_NOTIFICATION* is a protected broadcast that only the system can send. The *com.android.nfc.NfcService* service application component dynamically registers a broadcast receiver for this broadcast intent. Once the broadcast intent is received, it creates an *NfcService.EnableDisableTask* object, which has a type of *android.os.AsyncTask*, with the *TASK_EE_WIPE* constant as a parameter. The *EnableDisableTask.executeEeWipe()* method is called which initiates a wipe of the device through the native methods of the *com.android.nfc.nxp.NativeNfcSecureElement* class via the Java Native Interface calls to the *com_android_nfc_NativeNfcSecureElement.cpp* class. This action results in a factory reset of the device.

THREAT MODEL AND THREAT RESOLUTION

We assume that the user downloads and installs the malicious app via an official or unofficial app marketplace or sideloads the app. The code to factory reset the device can be introduced by repackaging a popular app and posting it on an official or third-party application market which is a common method to distribute malware (Potharaju, 2012; Vidas, 2013; Zhou, 2012). It is also possible that a user can be lured into installing the malicious app via social engineering (Bhattacharya, 2014; Fedler, 2013). The app requires no permissions, so the user may think the app is generally unprivileged and cannot initiate a factory reset of the device. The app with a package name of *com.sec.android.app.servicemodeapp* contains the broadcast receiver application component with two declarations that have differing protection levels. This app cannot be disabled or uninstalled by the user unless the device is rooted. The vulnerable app will remain on the device until Samsung resolves the vulnerability by updating the app, although it does not appear to be present in the Android 6.0.1 builds.

SAMSUNG S6 EDGE NOTIFICATION LISTENER VULNERABILITY

Certain Samsung Galaxy S6 Edge devices contain a vulnerability where the installation of a zero-permission third-party app with a specific package name can allow the app to read the content of notifications on the device without any further action from the user. This vulnerability appears to be limited to Samsung Galaxy S6 Edge devices that started out with an Android 5.0.2 build. If the user utilizes the "Information Stream" feature while using an Android 5.0.2 build, this introduces the vulnerability on the device by allowing a specific component name to receive notifications even though the corresponding app is not installed on the device. A component name serves as a unique identifier that contains the package name of an app and a class name contained within the package. The "Information Stream" feature is where the notifications are displayed on the right edge of the screen while the rest of the screen is off. If the vulnerability was introduced while using an Android 5.0.2 build, then the vulnerability should still be present on the same device even after it has been updated to Android 5.1.1 or Android 6.0.1 unless the user has performed a factory reset of the device or uninstalled a different notification listener app without first disabling it as a notification listener in the Settings app. To exploit the vulnerability, a third-party app with a specific component name (*com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService*) needs to be installed on the device. Once the user installs the third-party app containing the specific component name, the app can read the content of all subsequent notifications on the device. This app will start receiving notifications once the user installs the app, and the user does not need to launch the app to activate it. After each reboot, the app starts execution right after the Samsung Galaxy S6 Edge has completed the boot process even without the *android.permission.RECEIVE_BOOT_COMPLETED* permission. Once the app is installed, it will persistently execute to obtain and optionally exfiltrate the notification data. This vulnerability was assigned the CVE-2016-6910 identifier in the Common Vulnerabilities and Exposures (CVE) database.

NOFITICATION LISTENER CAPABILITY

A third-party app can receive the user's notifications if the user explicitly grants this ability to an app via the Settings app. The *com.android.settings/.Settings.NotificationAccessSettingsActivity* component handles the enabling

and disabling of component names that can receive the user's notifications. For an app to receive the user's notifications, the app will need to have a class that extends the *android.service.notification.NotificationListenerService* class and declare it as a service in a particular way in the app's *AndroidManifest.xml* file ("NotificationListenerService," n.d.). A notification listener app implements callback methods from the *NotificationListenerService* class that will be executed as notifications are posted or removed. The notification listener app receives an *android.service.notification.StatusBarNotification* object when a notification is posted. The *StatusBarNotification* object contains an *android.app.Notification* object, which contains the actual text of the notification that is displayed to the user.

Once a user enables an app to be a notification listener via the Settings app, the Settings app will write the name of the component that extends the *NotificationListenerService* class of the user-selected app to the *secure* table of the *settings.db* file. The *settings.db* file is an SQLite database that contains various configuration and settings values that are used throughout the Android Operating System (OS). */data/data/com.android.providers.settings/databases/settings.db* is generally the path to the file. The Settings app, having a package name of *com.android.settings*, can write to the secure table of the *settings.db* file since it is a system app that has been granted the *android.permission.WRITE_SECURE_SETTINGS* permission. A third-party app will not be granted the *WRITE_SECURE_SETTINGS* permission since the app needs to be installed on the system partition or be signed with the device platform key to utilize this permission.

NON-EXISTENT NOTIFICATION LISTENER APP VULNERABILITY

The secure table in the *settings.db* file, corresponding to the *android.provider.Settings.Secure* class ("Settings.Secure," n.d.) in the Android Application Programming Interface (API), has three columns: *_id*, *name*, and *value*. There is a row in the *secure* table where the *name* column contains *enabled_notification_listeners* and the *value* column contains a list of components that are allowed to receive the user's notifications. We will refer to this list of components value as the list of *enabled_notification_listeners*. This value can also be empty or contain a single component name. If there is more than one component that is a notification listener, then the component names will be delimited by a colon. Any component in the colon-separated list of *enabled_notification_listeners* has the ability to receive the notifications on the device. The *enabled_notification_listeners* string corresponds to the *android.provider.Settings.Secure.ENABLED_NOTIFICATION_LISTENERS* string constant, although it is not visible as part of the public Android API ("Settings," n.d.), it can be accessed using Java reflection. If a component name is in the list of *enabled_notification_listeners*, and the corresponding app is not installed on the device, then a third-party app can be installed that contains this component name to utilize this pre-established capability to become a notification listener. Therefore, care should be taken not to introduce any components in the list of *enabled_notification_listeners* that do not have a corresponding app installed on the device.

We are unsure of the exact scope of the affected Samsung Galaxy S6 Edge builds. We have confirmed that the vulnerability can be introduced in all of the models we have tested so far (*SM-G925V*, *SM-G925F*, *SM-G925A*, *SM-G925X*, *SM-G9250*, *SM-G925K*, *SM-G925L*, *SM-G925P*, *SM-G925R4*, *SM-G925S*, *SM-G925T*, *SM-G925I*, and *SM-G925W8*). Appendix 3 contains 40 different Android 5.0.2 builds that we identified which can introduce the vulnerability. We will focus on the Samsung Galaxy S6 Edge *SM-G925V* model to explain the vulnerability. All of the Android 5.0.2 builds (i.e., *LRX22G.G925VVRU1AOC3*, *LRX22G.G925VVRU1AOE2*, and *LRX22G.G925VVRU2AOF1*) for the *SM-G925V* model can introduce the vulnerability on the device. The Android 5.1.1 builds for the *SM-G925V* model cannot introduce the vulnerability, although the vulnerability should still exist on the device if it was present before the device was upgraded to an Android 5.1.1 build.

Focusing on the *SM-G925V* Samsung Galaxy S6 Edge Android 5.0.2 *LRX22G.G925VVRU1AOC3* build, we will explain how the vulnerability is introduced on the device. This build has the *CocktailBarService.apk* installed on the system partition with a package name of *com.samsung.android.app.cocktailbarservice*. We noticed that the *com.samsung.android.app.cocktailbarservice.policy.CocktailBarOverlayPolicy* class of this app gives two component names the ability to read the notifications on the device by writing them to list of *enabled_notification_listeners* in the *secure* table of the *settings.db* file. This app has the *WRITE_SECURE_SETTINGS* permission, which allows it to write to the secure table of the *settings.db* file. The *CocktailBarOverlayPolicy.callOnCreate()* method calls the *loadEnabledListeners()* method and then calls the *saveEnabledListeners()* method in the *CocktailBarOverlayPolicy* class. The *loadEnabledListeners()* method reads

the list of *enabled_notification_listeners* from the *secure* table of the *settings.db* file into a *java.util.HashSet* object. It then attempts to add the *com.samsung.android.app.catchfavorites/.catchnotifications.CatchNotificationsService* and *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* component names to the *HashSet* object. Since the data structure is a *HashSet* object, each element must be unique. Therefore, these two component names will not be added if they already exist in the *HashSet* object. The *saveEnabledListeners()* method writes the contents of the *HashSet* object as a colon-separated list to the *value* column of the row that has a value of *enabled_notification_listeners* for the *name* column in the *secure* table of the *settings.db* file.

The *CocktailBarOverlayPolicy.callOnCreate()* method is called by the *com.samsung.android.app.cocktailbarservice.CocktailBarService.onCreate()* service method. Therefore, whenever the *CocktailBarService* service application component is created, it will add the two previously mentioned component names to the colon-separated list of *enabled_notification_listeners* if they were not previously in the list. In the *LRX22G.G925VVRU1AOC3* build, the app with a package name of *com.samsung.android.app.catchfavorites* is installed on the device, but the app with a package name of *com.samsung.android.app.portalservicewidget* is not installed on the device. Therefore, if a third-party app that has a component with the same component name (i.e., *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService*) is installed on the device, then it will obtain the capability of being a notification listener as soon as it is installed.

The *CocktailBarService* service application component needs to be activated while the device is running an Android 5.0.2 build to add these two component names to the list of *enabled_notification_listeners*. The *CocktailBarService* service application component is activated when the user rubs the right edge of the screen when the device's display is off. Rubbing the right edge of the screen while the device's display is off will make the clock and current notifications visible to the user by activating the "Information Stream." This is a feature of the Samsung Galaxy S6 Edge that differentiates it from the Samsung Galaxy S6. If the user has never rubbed the right edge of the screen while the device's display is off while the device was running an Android 5.0.2 build, then these two component names, mentioned above, will most likely not have been added to the list of *enabled_notification_listeners*. If the device is still running an Android 5.0.2 build, then an app with a package name of *com.samsung.android.app.portalservicewidget* can send an *android.content.Intent* object to start the *CocktailBarService* service application component so it will add the *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* component name to the list of *enabled_notification_listeners*. This will enable the app to become a notification listener. The source code snippet in Figure 2 will launch the *CocktailBarService* service from a service application component of an external app.

```
Intent i = new Intent();
ComponentName cn = ComponentName.unflattenFromString("com.samsung.android.app.cocktailbarservice/.CocktailBarService");
i.setComponent(cn);
i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startService(i);
```

Figure 2. Launching the *CocktailBarService* application component from a third-party app.

In Samsung Galaxy S6 Edge Android 5.1.1 builds, the source code snippet above will not write the *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* component name to the list of *enabled_notification_listeners* since the *CocktailBarOverlayPolicy* class will not add a component to the list of *enabled_notification_listeners* unless the *com.samsung.android.app.catchfavorites* app is not actually installed on the device. This app is present on the Android 5.1.1 builds we have examined, but only the Android 5.0.2 builds will introduce the vulnerability, whereas the Android 5.1.1 and 6.0.1 builds will not.

In addition, it appears that the Samsung Galaxy S6 Edge Android 5.1.1 builds (and possibly the Android 5.0.2 builds) will remove non-existent notification listeners (i.e., apps that have a component name is the list of *enabled_notification_listeners* but are not installed on the device) under certain circumstances. For example, if the user enabled an app as a notification listener in the Settings app and uninstalls the app without first disabling it as a notification listener via the Settings app, this can force each component name in the list of *enabled_notification_listeners* to be examined to see if the corresponding app is installed on the device. When this occurs, the Android OS will remove any component name in the list of *enabled_notification_listeners* that does not have a corresponding app installed. If the notification listener app that the user previously enabled is first disabled from being a notification listener and then is uninstalled, this will not trigger an examination of the list of *enabled_notification_listeners* to remove non-existent notification listeners. If the user has never explicitly enabled

an app as a notification listener, then the removal of non-existent notification listeners should never have been triggered.

This *settings.db* file, which contains the list of *enabled_notification_listeners*, does not get overwritten when the device receives a Firmware Over-The-Air (FOTA) update. The files that reside on the data (i.e., *userdata*) partition, including the *settings.db* file, generally survive intact as the system partition is updated ("OTA Updates," n.d.). The *settings.db* file should remain the same unless the Settings app or another system app that has the *WRITE_SECURE_SETTINGS* permission explicitly modifies the *settings.db* file. In the Samsung Galaxy S6 Edge devices we examined, the *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* component was in the list of *enabled_notification_listeners* even though the Samsung Galaxy S6 Edge devices were running Android 5.1.1. Therefore, this component name will most likely persist in these devices even as they are updated, unless the user performs a factory reset on the device or a subsequent FOTA update of the Android OS contains a system app that specifically removes the *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* entry from the list of *enabled_notification_listeners*.

THREAT MODEL

We assume that the user downloads and installs a malicious notification listener app via an official or unofficial app marketplace or sideloads the app. The code to obtain the content of notifications can be introduced by repackaging a popular app and posting it on a third-party application market which is a common method to distribute malware (Potharaju, 2012; Vidas, 2013; Zhou, 2012). It is also possible that a user can be lured into installing the malicious app via social engineering (Bhattacharya, 2014; Fedler, 2013). The app needs to have a package name of *com.samsung.android.app.portalservicewidget* and also have component named *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* that extends the *NotificaitonListenerService* class which will allow it to obtain the notifications and execute whenever the device is on. The app will start execution after the boot process has completed even without the *RECEIVE_BOOT_COMPLETED* permission. This functionality can be accomplished from a zero-permission app, although the *INTERNET* permission may be required if the app is to surreptitiously exfiltrate the content of the notifications to a remote location.

THREAT RESOLUTION

The Android 5.1.1 builds for the Samsung Galaxy S6 Edge will not add the *com.samsung.android.app.catchfavorites/.catchnotifications.CatchNotificationsService* or the *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* component names to list of *enabled_notification_listeners* unless the corresponding apps are actually installed on the device. This precaution made it so that the vulnerability will not be introduced on devices that have never run an Android 5.0.2 build, but the vulnerability should still be present on devices that have run an Android 5.0.2 build, since the *settings.db* file, residing on the data partition, generally remains the same after FOTA updates. Therefore, the next FOTA update for the Samsung Galaxy S6 Edge should explicitly remove the *com.samsung.android.app.portalservicewidget/.notifications.CatchNotificationsService* component, if it exists, in the colon-separated list of *enabled_notification_listeners*. This should be performed by a system app that has the *WRITE_SECURE_SETTINGS* permission.

The user can install an app with the package name of *com.samsung.android.app.portalservicewidget* and then subsequently uninstall this app. When an app is uninstalled and it has a component name in list of *enabled_notification_listeners*, then its component name will be removed from the list of *enabled_notification_listeners* if it is not disabled as a notification listener prior to uninstalling it. The *com.samsung.android.app.portalservicewidget* app was never installed on the device, so it's component name was never removed from the list of *enabled_notification_listeners* after it was written to the list of *enabled_notification_listeners* by the app with a package name of *com.samsung.android.app.cocktailbarservice*.

CONCLUSION

We have shown the risk of insecure system applications that come pre-loaded on certain Android devices by explaining two vulnerabilities present in certain Samsung Android builds. Our analysis resulted in the discovery of two vulnerable system applications that allow a zero-permission third-party Android application the ability to factory reset the device and also the ability to obtain the text of the notifications received by the user. In addition, we have discovered and explained the consequences of declaring two application components having the same name but having differing access requirements. Third-party applications may be able to interact with more-privileged system applications depending on the access requirements. In certain circumstances, an unprivileged third-party application can have the system application perform a privileged action on its behalf, which can result in permission leakage. Moreover, system applications can introduce vulnerabilities that can be exploited by third-party applications even when the system applications are not directly accessible to the third-party applications. The system applications on Android should undergo a more thorough security analysis before being put onto a user's device, especially when they cannot be removed or disabled by the user.

ACKNOWLEGEMENTS

This paper was partly supported by Department of Homeland Security, Science and Technology contracts D15PC00178 and D15PC00154. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DHS or the US government.

APPENDIX 1

Models and Builds that contain a vulnerable version of the *serviceModeApp_FB.apk* that allows the phone to be factory reset from a zero-permission third party app.

Device	Model	OS Version	Build Number
GS6	SM-G920I	5.1.1	LMY47X.G920IDVU3DOJ6
GS6	SM-G920I	5.1.1	LMY47X.G920IDVU3COJ7
GS6	SM-G920I	5.1.1	LMY47X.G920IDVU2COGA
GS6	SM-G920K	5.0.2	LRX22G.G920KKKU1AOF6
GS6	SM-G920K	5.0.2	LRX22G.G920KKKU1AOD8
GS6	SM-G920K	5.0.2	LRX22G.G920KKKU1AODC
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU2COH7
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU3COJ2
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU2BOG7
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU2BOG8
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU2COH9
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU2BOGB
GS6	SM-G920K	5.1.1	LMY47X.G920KKKU3COI6
GS6	SM-G920V	5.0.2	LRX22G.G920VVRU1A0E2
GS6	SM-G920V	5.1.1	LMY47X.G920VVRU3BOG5
Note 5	SM-N920C	5.1.1	LMY47X.N920CXXU1AOGE
Note 5	SM-N920G	5.1.1	LMY47X.N920GUBU1AOH6
Note 5	SM-N920G	5.1.1	LMY47X.N920GUBU1AOI1
Note 5	SM-N920G	5.1.1	LMY47X.N920GUBU1AOI2
Note 5	SM-N920G	5.1.1	LMY47X.N920GDDU2AOJ5
Note 5	SM-N920K	5.1.1	LMY47X.N920KKKU2AOI8
Note 5	SM-N920L	5.1.1	LMY47X.N920LKLU2AOI8
Note 5	SM-N920P	5.1.1	LMY47X.N920PVPS2AOK3
Note 5	SM-N920P	5.1.1	LMY47X.N920PVPU1AOI6
Note 5	SM-N920S	5.1.1	LMY47X.N920SKSU2AOI8
Note 5	SM-N920T	5.1.1	LMY47X.N920TUVU2COI5
Note 5	SM-N920V	5.1.1	LMY47X.N920VVRU2AOGJ
GS6 Edge+	SM-G9280	5.1.1	LMY47X.G9280ZCU2AOJ8
GS6 Edge+	SM-G9280	5.1.1	LMY47X.G9280ZCU2AOJ9
GS6 Edge+	SM-G9287	5.1.1	LMY47X.G9287ZHU1AOGF
GS6 Edge+	SM-G9287	5.1.1	LMY47X.G9287ZHU1AOH5
GS6 Edge+	SM-G9287	5.1.1	LMY47X.G9287ZHU2AOJ6
GS6 Edge+	SM-G9287	5.1.1	LMY47X.G9287ZHU1AOH2
GS6 Edge+	SM-G9287	5.1.1	LMY47X.G9287ZHU1AOI1
GS6 Edge+	SM-G9287	5.1.1	LMY47X.G9287ZHU2AOJ7
GS6 Edge+	SM-G928C	5.1.1	LMY47X.G928CXXU1AOH3
GS6 Edge+	SM-G928C	5.1.1	LMY47X.G928CXXU1AOH4
GS6 Edge+	SM-G928C	5.1.1	LMY47X.G928CXXU1AOI1
GS6 Edge+	SM-G928C	5.1.1	LMY47X.G928CXXU2AOJ5
GS6 Edge+	SM-G928G	5.1.1	LMY47X.G928GDDU1AOH3
GS6 Edge+	SM-G928G	5.1.1	LMY47X.G928GDDU1AOGL
GS6 Edge+	SM-G928G	5.1.1	LMY47X.G928GUBU1AOH6
GS6 Edge+	SM-G928G	5.1.1	LMY47X.G928GUBU1AOGJ
GS6 Edge+	SM-G928G	5.1.1	LMY47X.G928GUBU1AOH4
GS6 Edge+	SM-G928G	5.1.1	LMY47X.G928GUBU1AOH5
GS6 Edge+	SM-G928T	5.1.1	LMY47X.G928TUVU1AOGD
GS6 Edge+	SM-G928T	5.1.1	LMY47X.G928TUVU1BOH4
GS6 Edge+	SM-G928T	5.1.1	LMY47X.G928TUVU1BOH6
GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZTU1AODC

GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZCU1AOE7
GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZTU1AOEA
GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZCU1AOE4
GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZCU1AOE8
GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZCU1AOF8
GS6 Edge	SM-G9250	5.0.2	LRX22G.G9250ZTU1AOF1
GS6 Edge	SM-G925A	5.1.1	LMY47X.G925AUCU3BOJ7
GS6 Edge	SM-G925A	5.1.1	LMY47X.G925AUCU3BOJ9
GS6 Edge	SM-G925F	5.0.2	LRX22G.G925FXXUA1OCZ
GS6 Edge	SM-G925F	5.0.2	LRX22G.G925FXXU1AOCV
GS6 Edge	SM-G925I	5.0.2	LRX22G.G925IDVU1AOE2
GS6 Edge	SM-G925K	5.0.2	LRX22G.G925KKKU1AOD8
GS6 Edge	SM-G925K	5.0.2	LRX22G.G925KKKU1AODC
GS6 Edge	SM-G925K	5.0.2	LRX22G.G925KKKU1AOE6
GS6 Edge	SM-G925K	5.0.2	LRX22G.G925KKKU1AOF6
GS6 Edge	SM-G925K	5.1.1	LMY47X.G925KKKU2BOG7
GS6 Edge	SM-G925K	5.1.1	LMY47X.G925KKKU2COH7
GS6 Edge	SM-G925K	5.1.1	LMY47X.G925KKKU2COH7
GS6 Edge	SM-G925L	5.0.2	LRX22G.G925LKLU1AOD8
GS6 Edge	SM-G925L	5.0.2	LRX22G.G925LKLU1AODC
GS6 Edge	SM-G925L	5.0.2	LRX22G.G925LKLU1AOE6
GS6 Edge	SM-G925S	5.0.2	LRX22G.G925SKSU1AOD5
GS6 Edge	SM-G925S	5.0.2	LRX22G.G925SKSU1AOD8
GS6 Edge	SM-G925S	5.0.2	LRX22G.G925SKSU1AODC
GS6 Edge	SM-G925S	5.0.2	LRX22G.G925SKSU1AOE6
GS6 Edge	SM-G925S	5.0.2	LRX22G.G925SKSU1AOF3
GS6 Edge	SM-G925V	5.0.2	LRX22G.G925VVRU1AOE2
GS6 Edge	SM-G925V	5.0.2	LRX22G.G925VVRU2AOF1
GS6 Edge	SM-G925V	5.1.1	LMY47X.G925VVRU3BOG5
GS6 Edge	SM-G925W8	5.0.2	LRX22G.G925W8VLU1AOE1
GS6 Edge	SM-G925W8	5.0.2	LRX22G.G925W8VLU2AOG2
GS6 Edge	SC-04G	5.0.2	LRX22G.SC04GOMU1AOD2
GS6 Edge	SC-04G	5.0.2	LRX22G.SC04GOMU1AOE1
GS6 Edge	SC-04G	5.0.2	LRX22G.SC04GOMU1AOH4
GS6 Edge	SC-04G	5.0.2	LRX22G.SC04GOMU1AOG3

APPENDIX 2

The *ServiceModeAppBroadcastReceiver* being declared twice from a Samsung S6 Edge running 5.1.1 with a build number of *LMY47X.G925AUCU3BOJ7*.

```
    E: receiver (line=161)
      A: android:name(0x01010003)=".ServiceModeAppBroadcastReceiver" (Raw:
".ServiceModeAppBroadcastReceiver")
      A: android:permission(0x01010006)="com.sec.android.app.servicemodeapp.permission.KEYSTRING" (Raw:
"com.sec.android.app.servicemodeapp.permission.KEYSTRING")
      E: intent-filter (line=164)
        E: action (line=165)
          A: android:name(0x01010003)="android.provider.Telephony.SECRET_CODE" (Raw:
"android.provider.Telephony.SECRET_CODE")
        E: data (line=167)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="22558463" (Raw: "22558463")
        E: data (line=170)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="RTN" (Raw: "RTN")
        E: data (line=173)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="3214789650" (Raw: "3214789650")
        E: data (line=176)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="9900" (Raw: "9900")
        E: data (line=179)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="0514" (Raw: "0514")
        E: data (line=182)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="66336" (Raw: "66336")
        E: data (line=185)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="66336324" (Raw: "66336324")
        E: data (line=188)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="99007788" (Raw: "99007788")
        E: data (line=191)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="638732" (Raw: "638732")
        E: data (line=195)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="7284" (Raw: "7284")
        E: data (line=198)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="06" (Raw: "06")
        E: data (line=202)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="272886" (Raw: "272886")
        E: data (line=205)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
          A: android:host(0x01010028)="773738" (Raw: "773738")
        E: data (line=209)
          A: android:scheme(0x01010027)="android_secret_code" (Raw: "android_secret_code")
```

A: android:host(0x01010028)="0808" (Raw: "0808")
E: intent-filter (line=214)
E: action (line=215)
A: android:name(0x01010003)="com.samsung.intent.action.SEC_FACTORY_RESET_WITHOUT_FACTORY_UI" (Raw: "com.samsung.intent.action.SEC_FACTORY_RESET_WITHOUT_FACTORY_UI")
E: receiver (line=218)
A: android:name(0x01010003)=".ServiceModeAppBroadcastReceiver" (Raw: ".ServiceModeAppBroadcastReceiver")
E: intent-filter (line=219)
E: action (line=220)
A: android:name(0x01010003)="android.intent.action.BOOT_COMPLETED" (Raw: "android.intent.action.BOOT_COMPLETED")
E: action (line=221)
A: android:name(0x01010003)="com.sec.android.app.servicemodeapp.NOTIDUMP_OFF" (Raw: "com.sec.android.app.servicemodeapp.NOTIDUMP_OFF")

APPENDIX 3

Models and Android 5.0.2 builds that can introduce the non-existent notification listener vulnerability.

Model	Build Number
SM-G9250	LRX22G.G9250ZTU1AODC
SM-G9250	LRX22G.G9250ZCU1AOE4
SM-G9250	LRX22G.G9250ZCU1AOE8
SM-G9250	LRX22G.G9250ZCU1AOE7
SM-G9250	LRX22G.G9250ZTU1AOEA
SM-G9250	LRX22G.G9250ZCU1AOF8
SM-G9250	LRX22G.G9250ZTU1AOF1
SM-G925A	LRX22G.G925AUCU1AOCE
SM-G925F	LRX22G.G925FXXU1AOCV
SM-G925F	LRX22G.G925FXXU1AOCZ
SM-G925I	LRX22G.G925IDVU1AOC4
SM-G925I	LRX22G.G925IDVU1AOE2
SM-G925K	LRX22G.G925KKKU1AOD8
SM-G925K	LRX22G.SC04GOMU1AOD2
SM-G925K	LRX22G.G925KKKU1AODC
SM-G925K	LRX22G.G925KKKU1AOE6
SM-G925K	LRX22G.G925KKKU1AOF6
SM-G925L	LRX22G.G925LKLU1AOD8
SM-G925L	LRX22G.G925LKLU1AODC
SM-G925L	LRX22G.G925LKLU1AOE6
SM-G925P	LRX22G.G925PVPU1AOCF
SM-G925P	LRX22G.G925PVPU1AOE2
SM-G925R4	LRX22G.G925R4TYU1AOD3
SM-G925R4	LRX22G.G925R4TYU1AOE2
SM-G925S	LRX22G.G925SKSU1AOD5
SM-G925S	LRX22G.G925SKSU1AOD8
SM-G925S	LRX22G.G925SKSU1AODC
SM-G925S	LRX22G.G925SKSU1AOE6
SM-G925S	LRX22G.G925SKSU1AOF3
SM-G925T	LRX22G.G925TTMB1AOCG
SM-G925V	LRX22G.G925VVRU1AOC3
SM-G925V	LRX22G.G925VVRU1AOE2
SM-G925V	LRX22G.G925VVRU2AOF1
SM-G925W8	LRX22G.G925W8VLU1AOCG
SM-G925W8	LRX22G.G925W8VLU1AOE1
SM-G925W8	LRX22G.G925W8VLU2AOG2
SM-G925X	LRX22G.G925XXXU1AOC6_LLK
SC-04G	LRX22G.SC04GOMU1AOD2
SC-04G	LRX22G.SC04GOMU1AOE1
SC-04G	LRX22G.SC04GOMU1AOH4
SC-04G	LRX22G.SC04GOMU1APA5
SC-04G	LRX22G.SC04GOMU1AOG3

REFERENCES

ArrayMap. (n.d.). Retrieved from https://android.googlesource.com/platform/frameworks/base/+/android-6.0.0_r1/core/java/android/util/ArrayMap.java.

Bhattacharya, P., Yang, L., Guo, M., Qian, K., & Yang, M. (2014). Learning mobile security with Labware. *IEEE Security & Privacy*, *12*(1), 69-72.

Bagheri, H., Sadeghi, A., Garcia, J., & Malek, S. (2015). Covert: Compositional analysis of android inter-app permission leakage. *IEEE Transactions on Software Engineering*, *41*(9), 866-886.

Chin, E., Felt, A. P., Greenwood, K., & Wagner, D. (2011). Analyzing inter-application communication in Android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (pp. 239-252). ACM.

Fedler, R., Schütte, J., & Kulicke, M. (2013). On the effectiveness of malware protection on android. *Fraunhofer AISEC*, *45*.

Grace, M. C., Zhou, Y., Wang, Z., & Jiang, X. (2012). Systematic Detection of Capability Leaks in Stock Android Smartphones. In *NDSS* (Vol. 14, p. 19).

IntentResolver. (n.d.). Retrieved from https://android.googlesource.com/platform/frameworks/base/+/android-6.0.0_r1/services/core/java/com/android/server/IntentResolver.java.

NotificationListenerService. (n.d.) Retreived from http://developer.android.com/reference/android/service/notification/NotificationListenerService.html.

Octeau, D., McDaniel, P., Jha, S., Bartel, A., Bodden, E., Klein, J., & Le Traon, Y. (2013). Effective inter-component communication mapping in android: An essential step towards holistic security analysis. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)* (pp. 543-558).

OTA Updates. (n.d.). Retrieved from https://source.android.com/devices/tech/ota/.

PackageManagerService. (n.d.). Retrieved from https://android.googlesource.com/platform/frameworks/base/+/android-6.0.0_r1/services/core/java/com/android/server/pm/PackageManagerService.java.

PackageParser. (n.d.). Retrieved from https://android.googlesource.com/platform/frameworks/base/+/56a2301/core/java/android/content/pm/PackageParser.java.

Potharaju, R., Newell, A., Nita-Rotaru, C., & Zhang, X. (2012). Plagiarizing smartphone applications: attack strategies and defense techniques. In *International Symposium on Engineering Secure Software and Systems* (pp. 106-120). Springer Berlin Heidelberg.

Settings. (n.d.). Retrieved from https://android.googlesource.com/platform/frameworks/base/+/android-6.0.0_r1/core/java/android/provider/Settings.java.

Settings.Secure. (n.d.). Retrieved from http://developer.android.com/reference/android/provider/Settings.Secure.html.

Vidas, T., & Christin, N. (2013). Sweetening android lemon markets: measuring and combating malware in application marketplaces. In *Proceedings of the third ACM conference on Data and application security and privacy* (pp. 197-208). ACM.

Zhou, Y., & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy* (pp. 95-109). IEEE.