Introduction

How to get set up / getting started

About This Course

- This is an Android course focused on teaching application logic vulnerabilities
- 3 primary attack perspectives:
  - Untrusted third party apps on device
  - Phishing users to tap an attacker controlled link
  - Scanning potentially dangerous NFC tags

- The topics discussed in these lessons should only be used for educational purposes only. Malicious Erection LLC does not condone the use of any material presented here to be used for any other reason.

- This course is catered toward students who have experience in reverse engineering Android applications
- The following topics will not be covered:
  - Rooting and/or modifying the security model of the Android OS
  - Hooking into Android applications
  - Setup a hooking and/or reverse engineering environment
  - How application sandboxing works in Android
  - What the External Storage is and how it works
  - The different types of IPCs that are used to share data between Android applications
- It is HIGHLY recommended that you are comfortable with the points listed above before proceeding with this course

Requirements – Android Device/Emulator

- To complete this course, you will need either a physical Android device or an Android emulator. Please note, for the module on NFC tags a physical device is required.

Physical device requirements

- Rooted device running at least Android 12
  - Google Pixel device is preferred
- A USB cable to connect your device to your workstation
- OPTIONAL - A writable NFC tag
  - Tags which are ISO 14443A certified will work

Emulator requirements

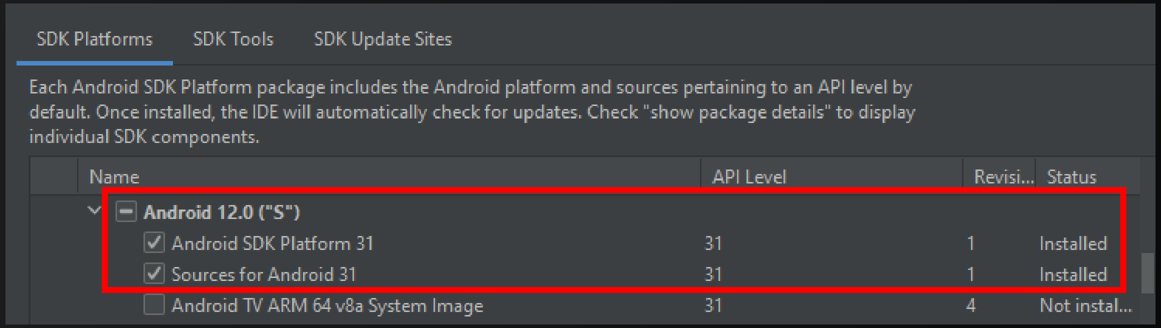
- Model: Any
- ROM: Android 12.0 (Google APIs)
  - Choose the processor that corresponds to your host machine
- RAM: minimum 1536 MB
- VM heap: minimum 228 MB
- Internal Storage: minimum 2048 MB
- SD card: minimum 512 MB

MILICIOUS ERECTION LLC

Requirements – Software

- Favorite Android application decompiler
- "Axolotl" application
  - The `.apk` file is bundled with this course
  - Install the `.apk` file onto your Android device

- Android Studio
  - SDK Platforms
    - Android SDK Platform 31
    - Sources for Android 31
  - SDK Tools
    - Android SDK Build-Tools – 30.0.0 or above
    - Latest version of Android SDK Command-line Tools



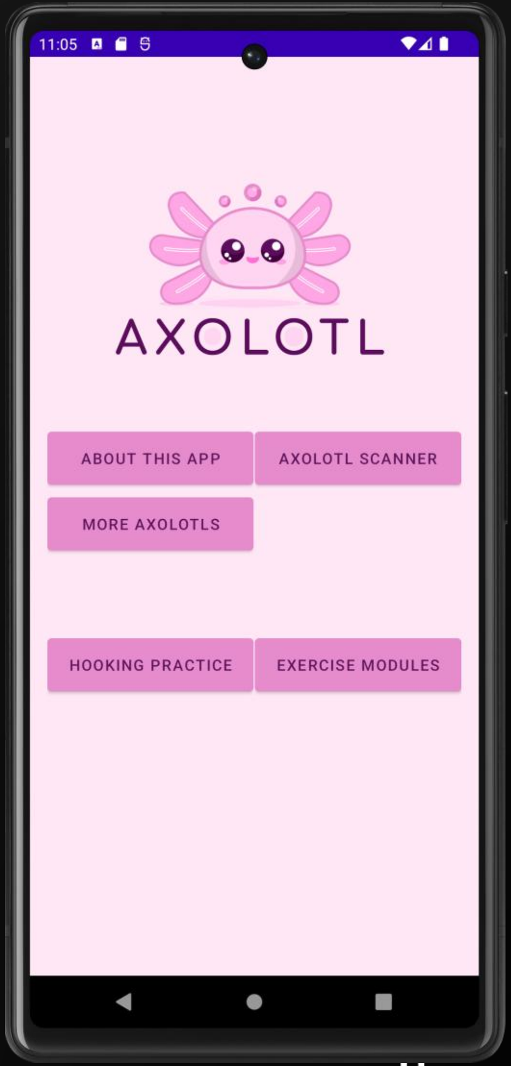
Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, the IDE will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revisi...	Status
Android 12.0 ("S")			
✓ Android SDK Platform 31	31	1	Installed
✓ Sources for Android 31	31	1	Installed
Android TV ARM 64 v8a System Image	31	4	Not instal...

Android Studio SDK Manager showing Android SDK Platform 31 being installed

Axolotl – Companion Application

- The goal is to retrieve all the flags scattered throughout the application
- Getting all the flags demonstrates an ability to:
    - Decompile Android applications
    - Read and understand Android source code
    - Create custom applications to exploit Android vulnerabilities
- Install this application on your test device



MILLICIOUS ERECTION LLC

Module 0 – Tooling, Setup, and Hooking

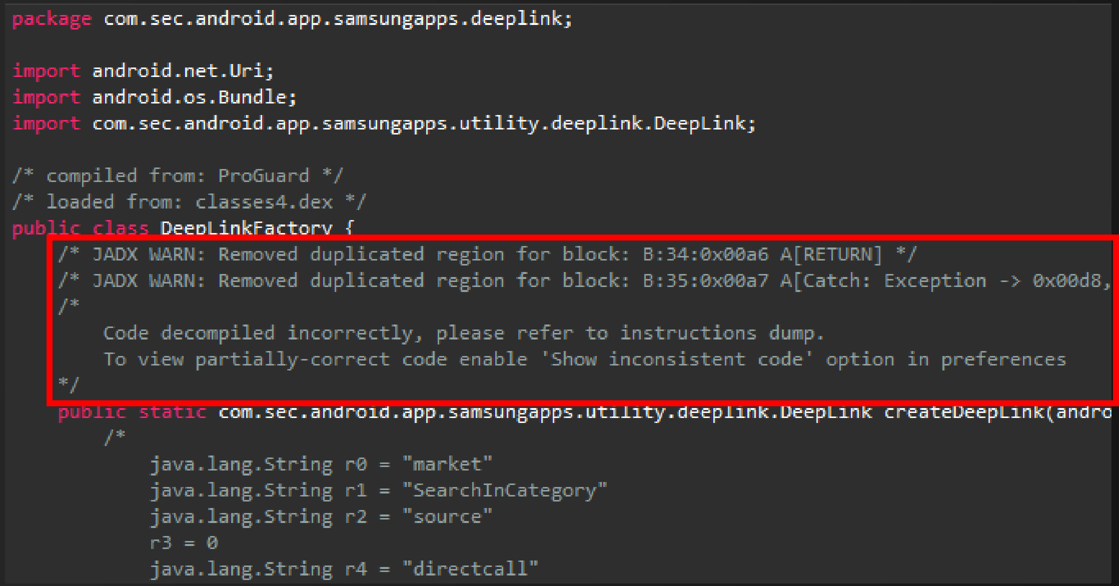
MILICIOUS
ERECTION LLC

Tooling - Decompilers

- JADX - https://github.com/skylot/jadx/releases
  - Usually the “go to” decompiler

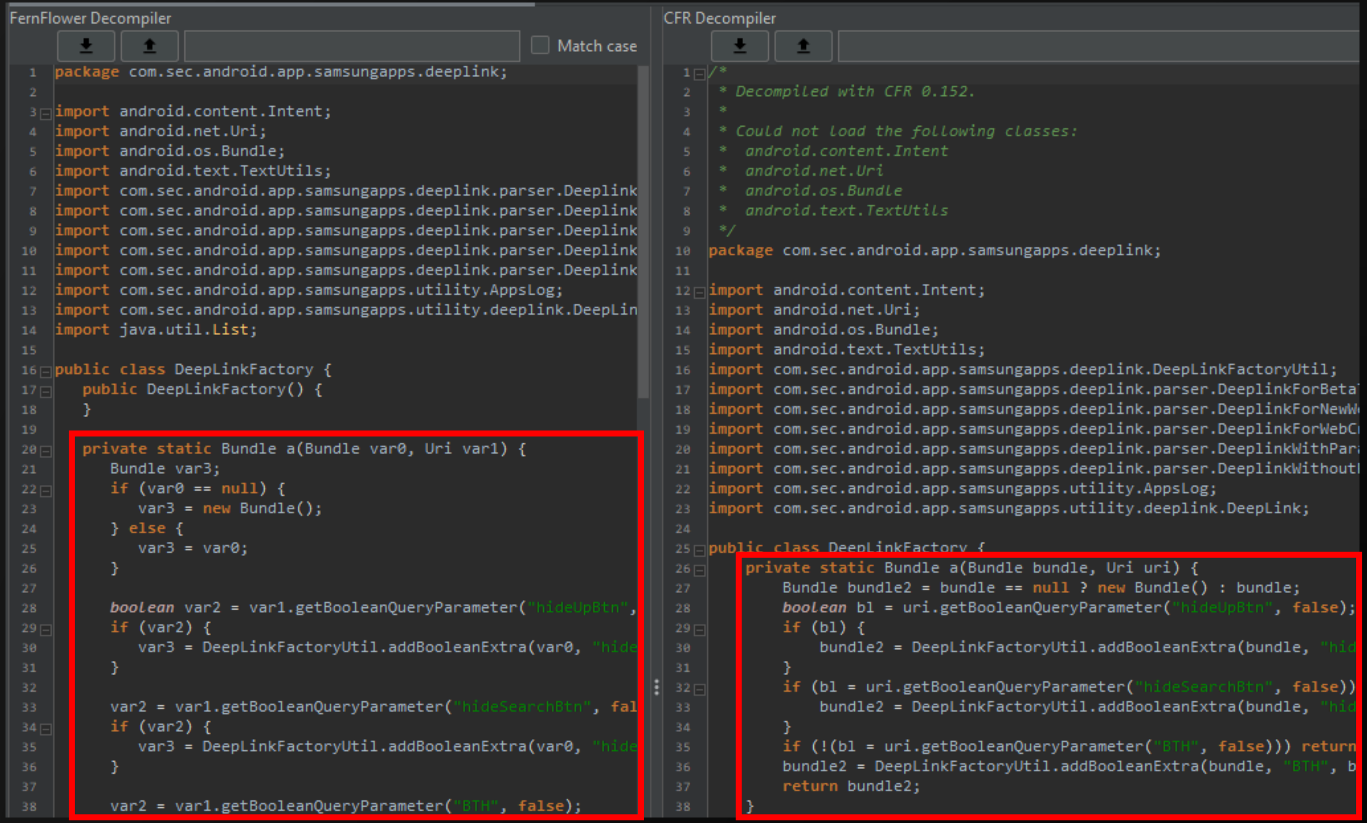
- ByteCode Viewer - https://github.com/Konloch/bytecode-viewer/releases
  - Combines different decompilers into one executable
    - JD-Gui/Core
    - Procyon
    - CFR
    - Fernflower
    - Krakatau
    - JADX-Core
  - The decompilers that are shipped with ByteCode Viewer might be outdated

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 */
/* loaded from: classes4.dex */
public class DeepLinkFactory {
    /* JADX WARN: Removed duplicated region for block: B:34:0x00a6 A[RETURN] */
    /* JADX WARN: Removed duplicated region for block: B:35:0x00a7 A[Catch: Exception -> 0x00d8,
    /*
        Code decompiled incorrectly, please refer to instructions dump.
        To view partially-correct code enable 'Show inconsistent code' option in preferences
    */
    public static com.sec.android.app.samsungapps.utility.deeplink.DeepLink createDeepLink(andro
        /*
            java.lang.String r0 = "market"
            java.lang.String r1 = "SearchInCategory"
            java.lang.String r2 = "source"
            r3 = 0
            java.lang.String r4 = "directcall"
```

JADX failing to fully decompile a Java class



FernFlower Decompiler

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

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import com.sec.android.app.samsungapps.deeplink.parser.Deeplink
import com.sec.android.app.samsungapps.deeplink.parser.Deeplink
import com.sec.android.app.samsungapps.deeplink.parser.Deeplink
import com.sec.android.app.samsungapps.deeplink.parser.Deeplink
import com.sec.android.app.samsungapps.deeplink.parser.Deeplink
import com.sec.android.app.samsungapps.utility.AppsLog;
import com.sec.android.app.samsungapps.utility.deeplink.DeepLin
import java.util.List;

public class DeepLinkFactory {
   public DeepLinkFactory() {
   }

   private static Bundle a(Bundle var0, Uri var1) {
      Bundle var3;
      if (var0 == null) {
         var3 = new Bundle();
      } else {
         var3 = var0;
      }

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

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

      var2 = var1.getBooleanQueryParameter("BTH", false);
```

CFR Decompiler

```
/*
 * Decompiled with CFR 0.152.
 *
 * Could not load the following classes:
 *  android.content.Intent
 *  android.net.Uri
 *  android.os.Bundle
 *  android.text.TextUtils
 */
package com.sec.android.app.samsungapps.deeplink;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import com.sec.android.app.samsungapps.deeplink.DeepLinkFactoryUtil;
import com.sec.android.app.samsungapps.deeplink.parser.DeeplinkForBeta
import com.sec.android.app.samsungapps.deeplink.parser.DeeplinkForNewW
import com.sec.android.app.samsungapps.deeplink.parser.DeeplinkForWebC
import com.sec.android.app.samsungapps.deeplink.parser.DeeplinkWithPar
import com.sec.android.app.samsungapps.deeplink.parser.DeeplinkWithout
import com.sec.android.app.samsungapps.utility.AppsLog;
import com.sec.android.app.samsungapps.utility.deeplink.DeepLink;

public class DeepLinkFactory {
    private static Bundle a(Bundle bundle, Uri uri) {
        Bundle bundle2 = bundle == null ? new Bundle() : bundle;
        boolean bl = uri.getBooleanQueryParameter("hideUpBtn", false);
        if (bl) {
            bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, "hid
        }
        if (bl = uri.getBooleanQueryParameter("hideSearchBtn", false))
            bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, "hid
        }
        if (!(bl = uri.getBooleanQueryParameter("BTH", false))) return
        bundle2 = DeepLinkFactoryUtil.addBooleanExtra(bundle, "BTH", b
        return bundle2;
    }
```

ByteCode Viewer using Fernflower and CFR Decompiler to decompile the same class

Tooling – Hooking Frameworks

- Frida - https://github.com/frida/frida/releases
  - Frida wrapper Objection - https://github.com/sensepost/objection
- LSPosed (modern Xposed) - https://github.com/LSPosed/LSPosed/releases
  - Sometimes to evade hooking detection methods, you will need to hook Android applications at the Zygote level; this is exactly what LSPosed does
  - LSPosed also comes in handy if the Android application forks itself into separate processes; since LSPosed is injected at the Zygote level, the LSPosed code will be injected into all forked processes
  - To install LSPosed, Magisk is required to be installed on the device, along with Zygisk being enabled

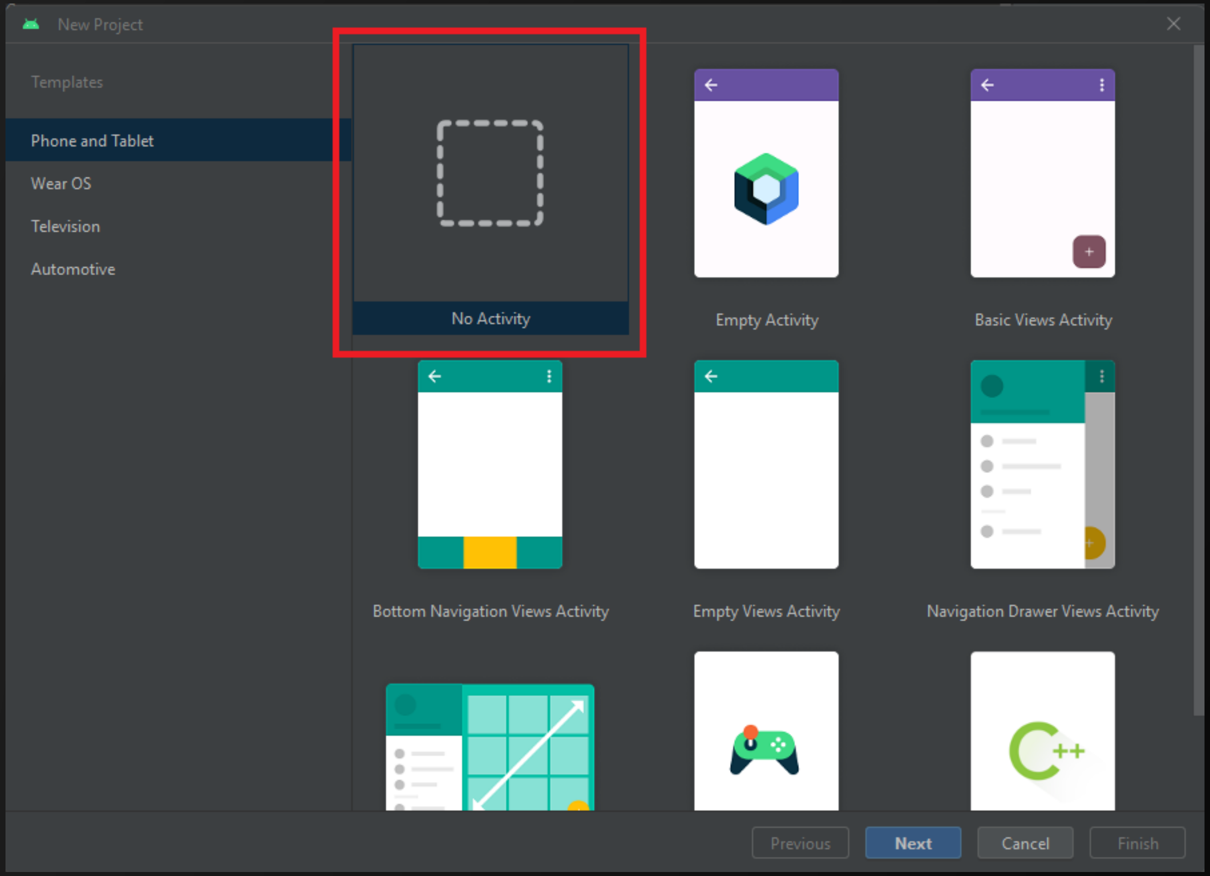
Tooling – Android Studio and Example Exploit

- To demonstrate some of the vulnerabilities outlined in this course, you will be programming an Android application which will act as a “malicious third party app”
- This application will be installed on your test device
  - All programming will be done in Android Studio
  - Java will be the primary programming language for this application
- From here on out, this application will be referred to as “Example Exploit”
  - The next few slides will go over how to setup and configure the Android Studio project for “Example Exploit”

Note that the screenshot(s) in this presentation were taken from Android Studio version 2022.3.1 Patch 2. Your version of Android Studio may look slightly different.

Tooling – Android Studio and Example Exploit

- In Android Studio, create a new project
  - Phone and Tablet project with “No Activity”

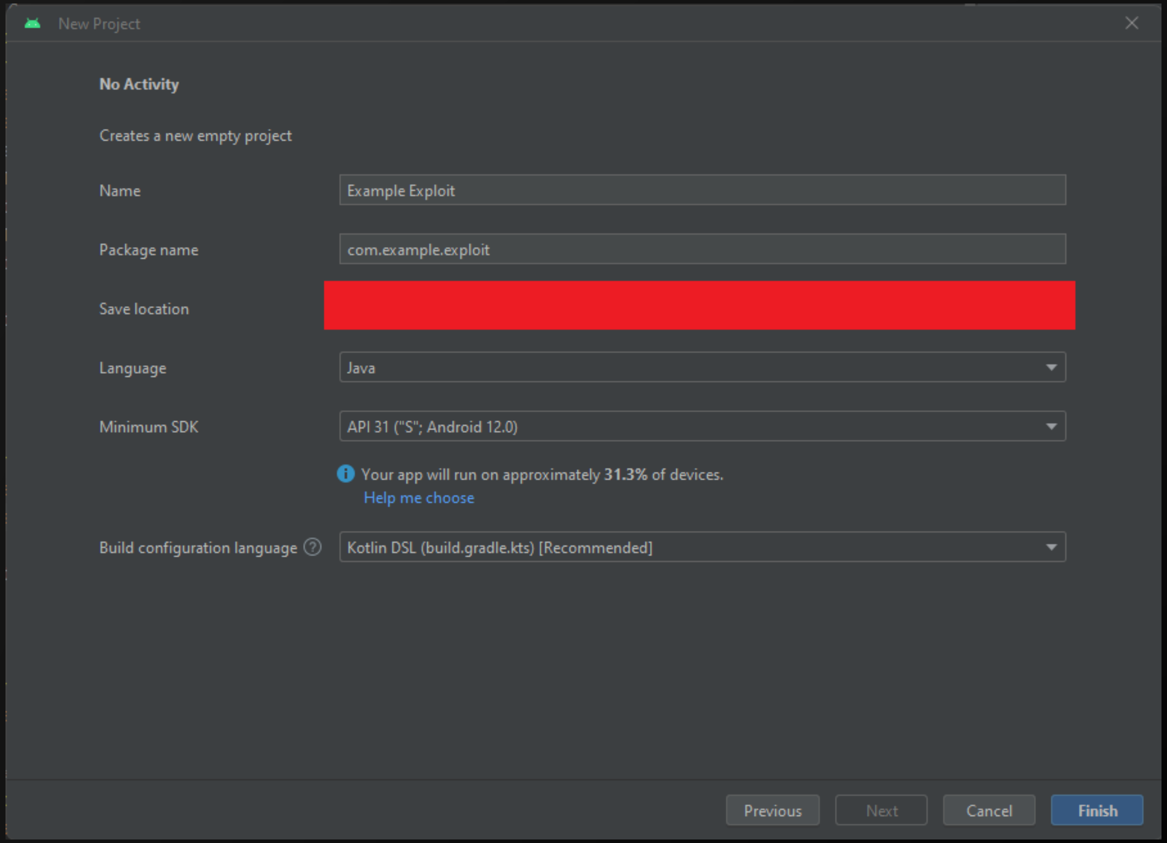


Android Studio – selecting “No Activity”

MILICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

- Configure the project with the following settings:
  - Name: **Example Exploit**
  - Package Name: **`com.example.exploit`**
  - Language: **Java**
  - Minimum SDK: **31**
  - Build Configuration Language: **default value**


New Project

No Activity

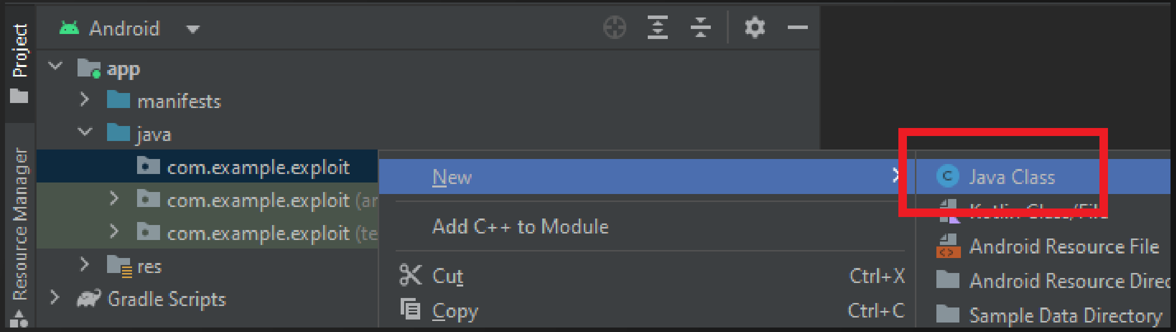
Creates a new empty project

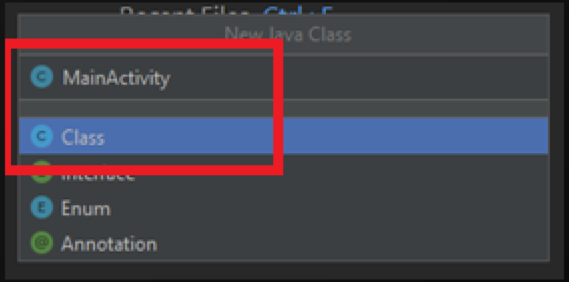
Name	Example Exploit
Package name	com.example.exploit
Save location	[redacted]
Language	Java
Minimum SDK	API 31 ("S"; Android 12.0)
	Your app will run on approximately **31.3%** of devices. Help me choose
Build configuration language	Kotlin DSL (build.gradle.kts) [Recommended]

Android Studio – configuring the package

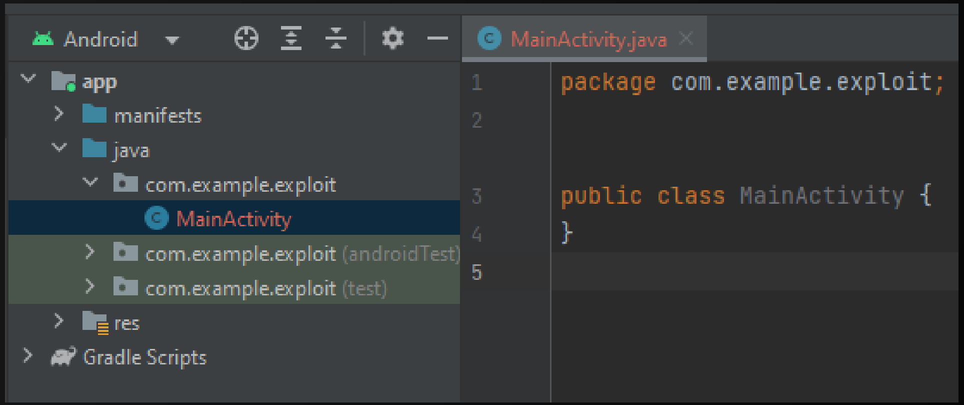
Tooling – Android Studio and Example Exploit

- After the project is created, create a new Java Class called `MainActivity`
- You should then be presented with an empty Java class





Android Studio – adding a Java Class

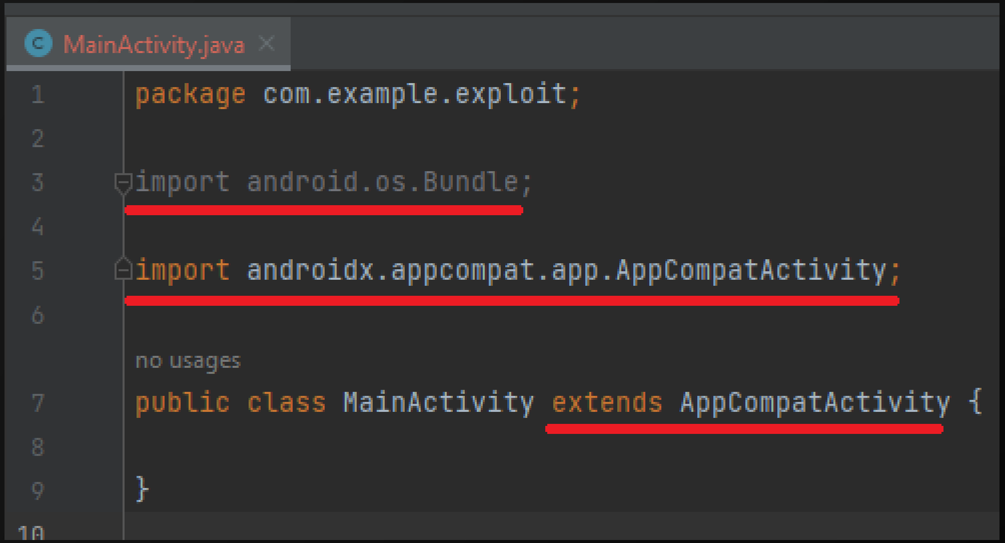


```java
package com.example.exploit;

public class MainActivity {
}
```

Note that the screenshot(s) in this presentation were taken from Android Studio version 2022.3.1 Patch 2. Your project configuration options may look different.

Tooling – Android Studio and Example Exploit

- Add the following code to the `MainActivity` class:
  - Import the additional libraries
    - `android.os.Bundle`
    - `androidx.appcompat.app.AppCompatActivity`
- Extend the Java class with `AppCompatActivity`



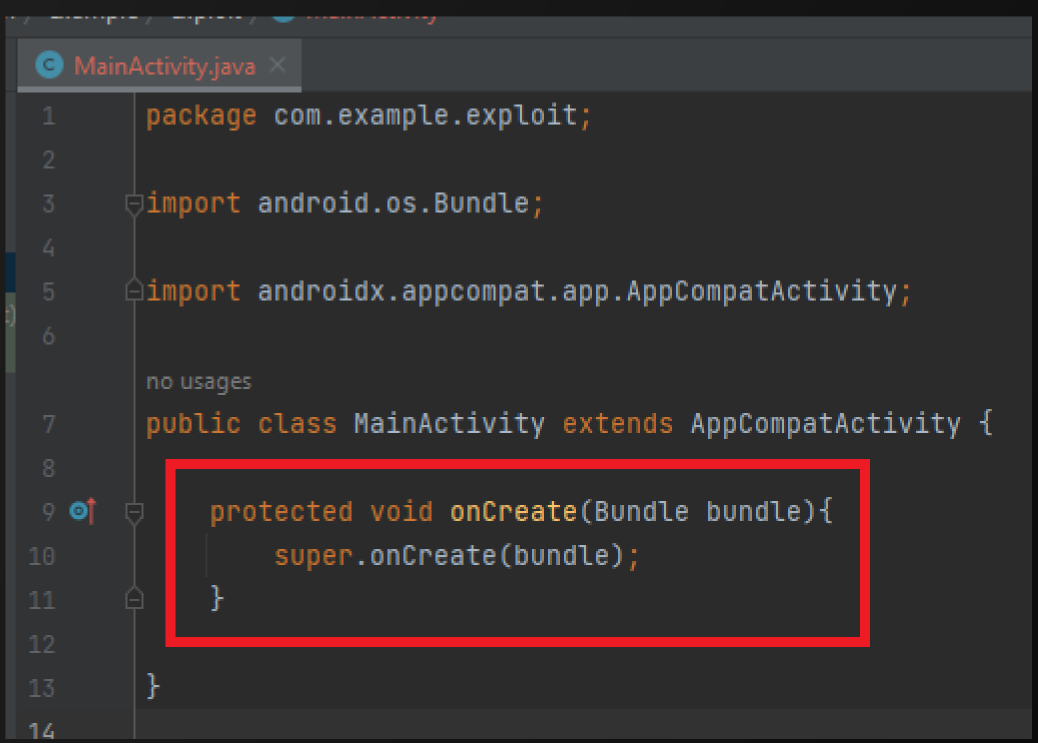
MainActivity.java

```
package com.example.exploit;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

no usages
public class MainActivity extends AppCompatActivity {

}
```

MILICIOUS
ERECTION LLC

Tooling – Android Studio and Example Exploit

- Create a new `protected void` method `onCreate(Bundle)` and add the appropriate `super` code



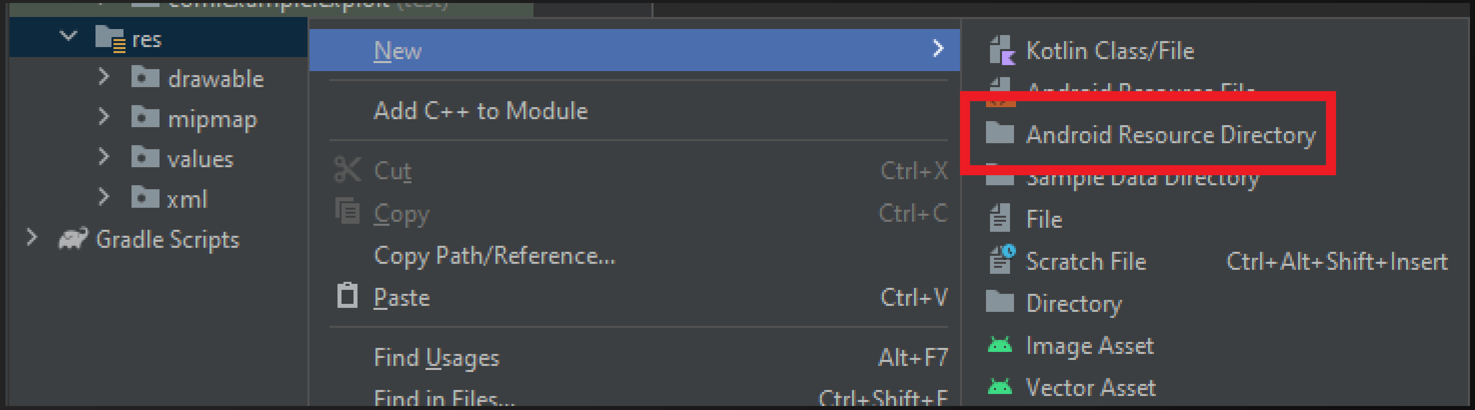
MainActivity.java

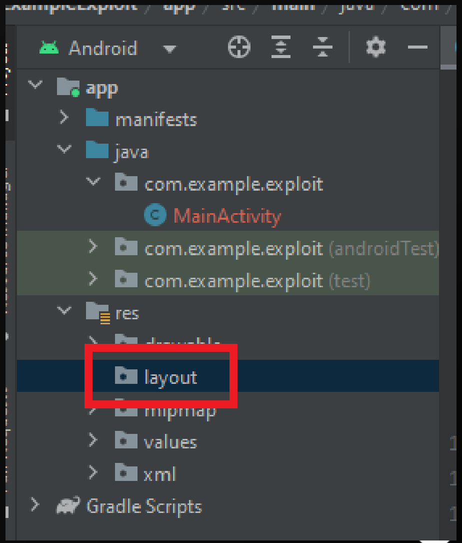
```java
package com.example.exploit;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

no usages
public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle bundle){
        super.onCreate(bundle);
    }

}
```

MILICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

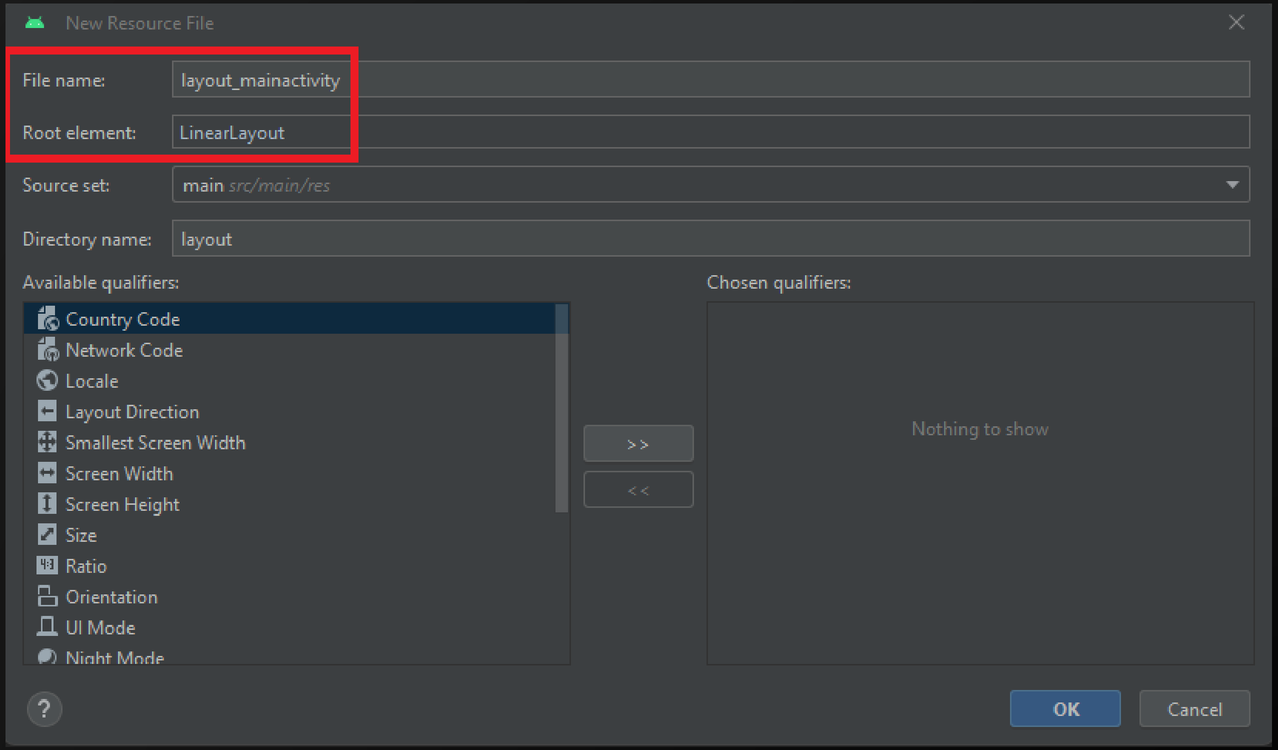
- In the project structure, right click “res” and create an Android Resource Directory called “layout”



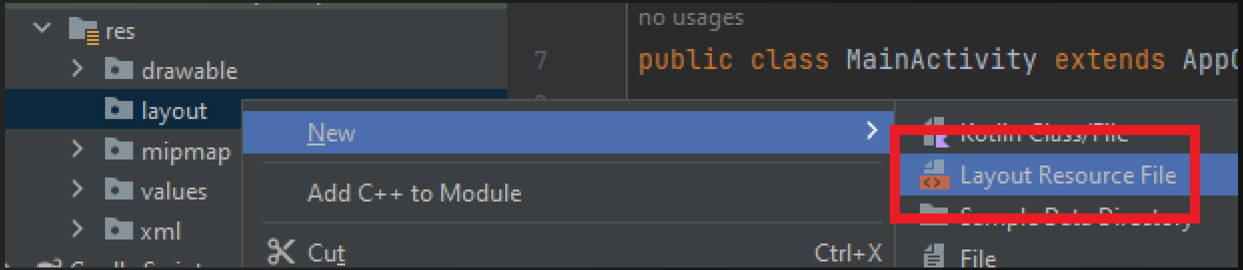


Tooling – Android Studio and Example Exploit

- Right click on "layout" and create a "Layout Resource File" called `layout_mainactivity`
- Configure the "Root Element" value to be `LinearLayout`


New Resource File

File name: layout_mainactivity
Root element: LinearLayout
Source set: main src/main/res
Directory name: layout

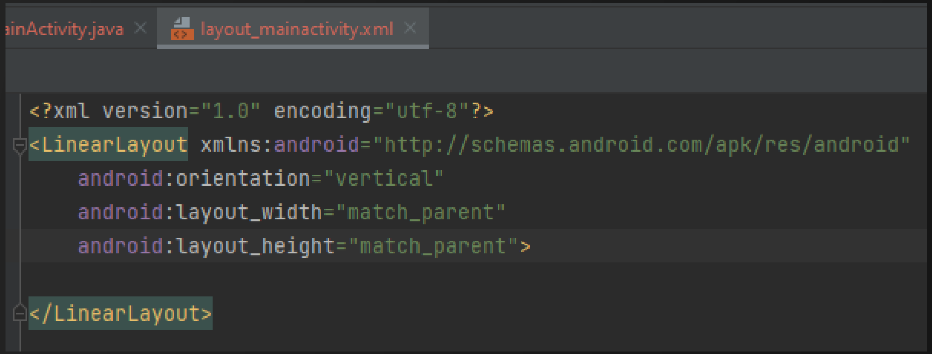


```
no usages
public class MainActivity extends App(
```

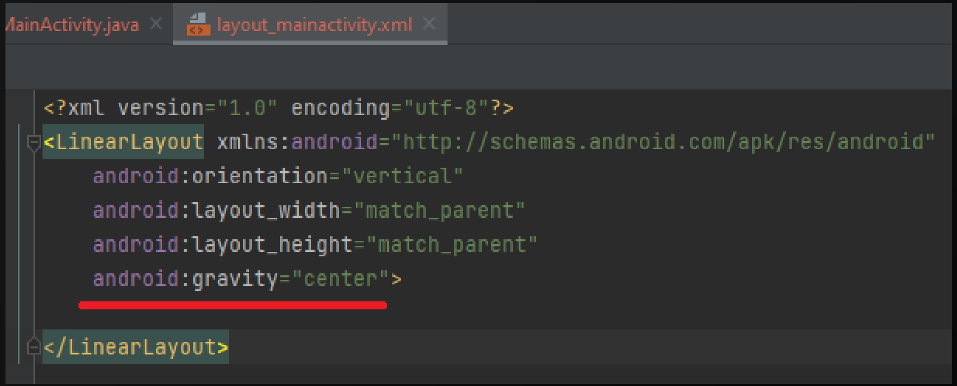
MILICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

- When the new layout is created, make sure the view is set to either “Split” or “Code”
- The default “LinearLayout” code should be pre-populated
- Add the following property value:
  - `android:gravity=“center”`



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</LinearLayout>
```

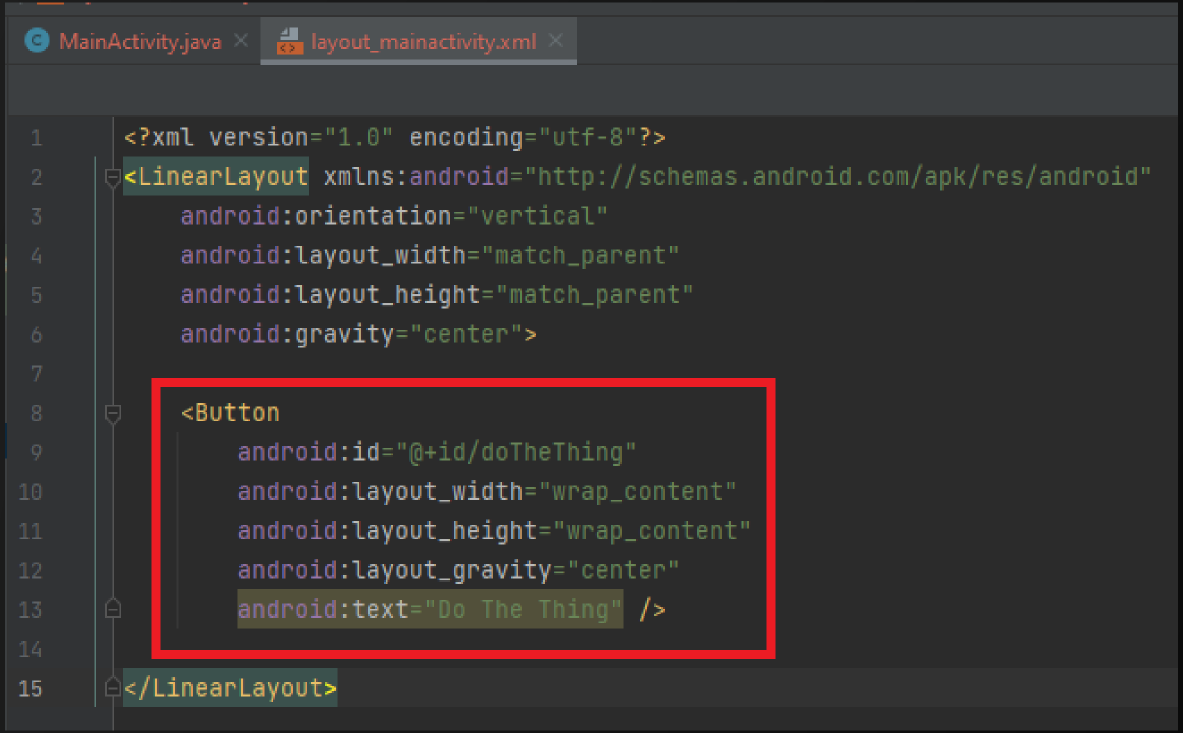


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

</LinearLayout>
```

MILICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

- Add a button to the layout with the ID `doTheThing` and add a text value "Do The Thing"

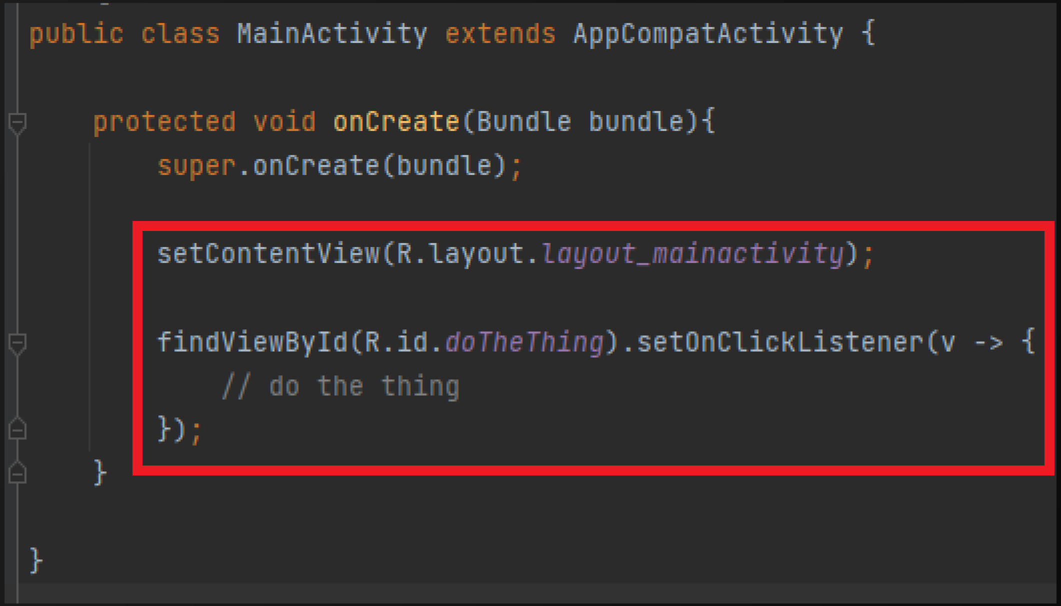


```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

    <Button
        android:id="@+id/doTheThing"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Do The Thing" />

</LinearLayout>
```

MILICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

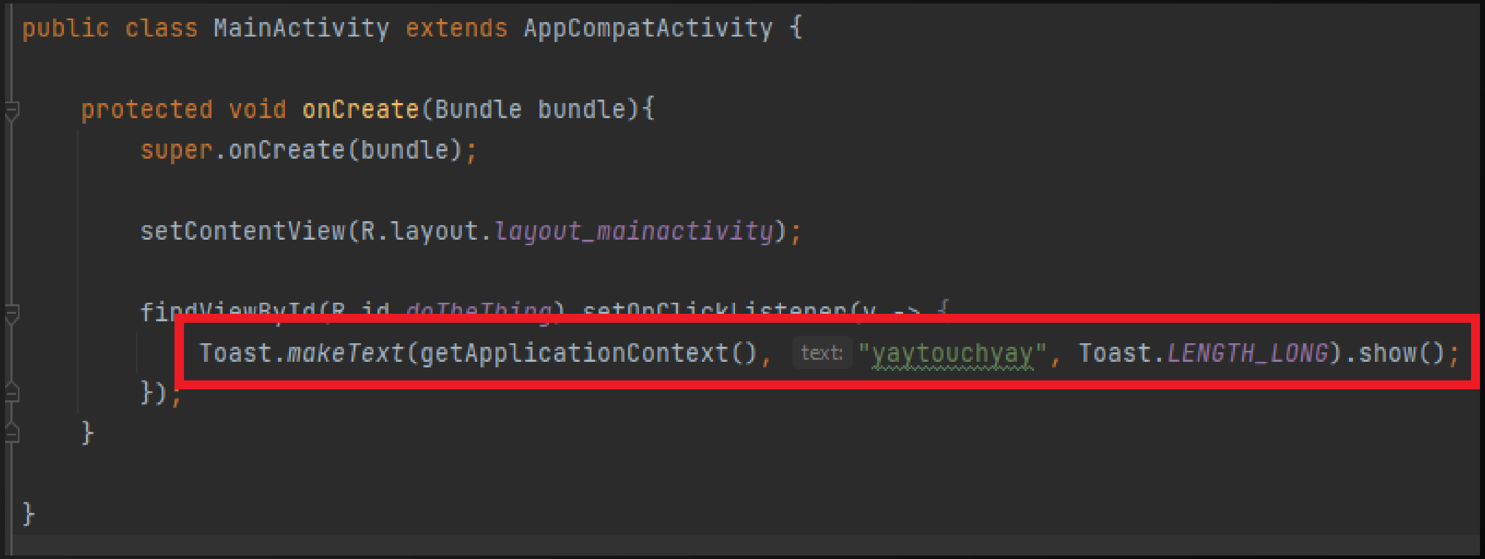
- Going back to the `MainActivity` class, add code to the `onCreate(Bundle)` so that:
    - The layout of `MainActivity` is set to the layout file you just created
    - A listener is created to execute code when the button `doTheThing` is tapped



```
public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle bundle){
        super.onCreate(bundle);

        setContentView(R.layout.layout_mainactivity);

        findViewById(R.id.doTheThing).setOnClickListener(v -> {
            // do the thing
        });
    }

}
```

Tooling – Android Studio and Example Exploit

- Add code so that when the button is tapped, a Toast message appears with a custom message
  - The code below will cause a Toast message to appear with the text "yaytouchyay" whenever the button is tapped

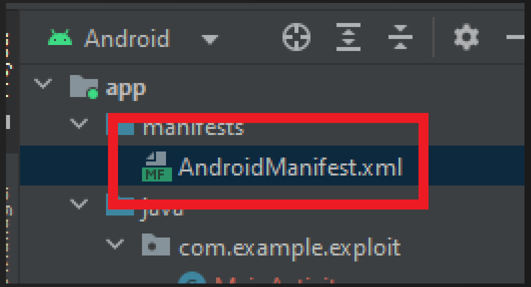


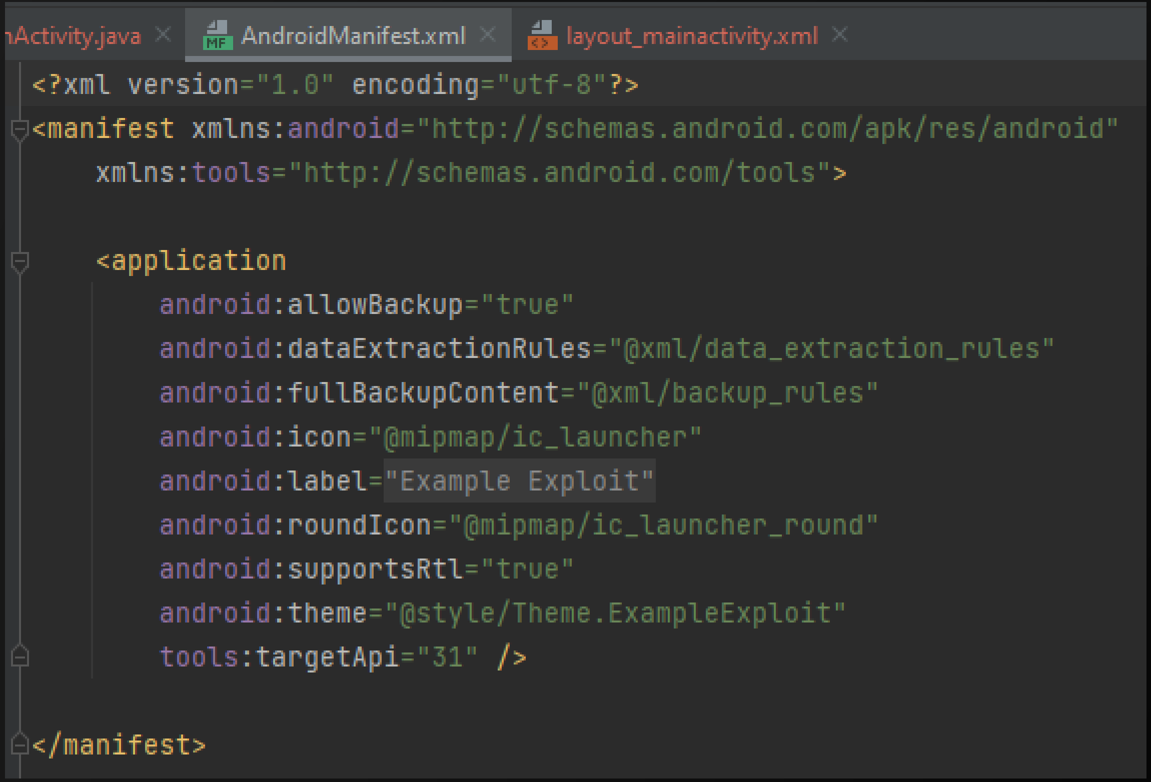
```
public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle bundle){
        super.onCreate(bundle);

        setContentView(R.layout.layout_mainactivity);

        findViewById(R.id.doTheThing).setOnClickListener(v -> {
            Toast.makeText(getApplicationContext(), text: "yaytouchyay", Toast.LENGTH_LONG).show();
        });
    }

}
```

MILICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

- In the project view panel, open the `Manifest.xml` file
- You should be presented with the contents of `Manifest.xml`



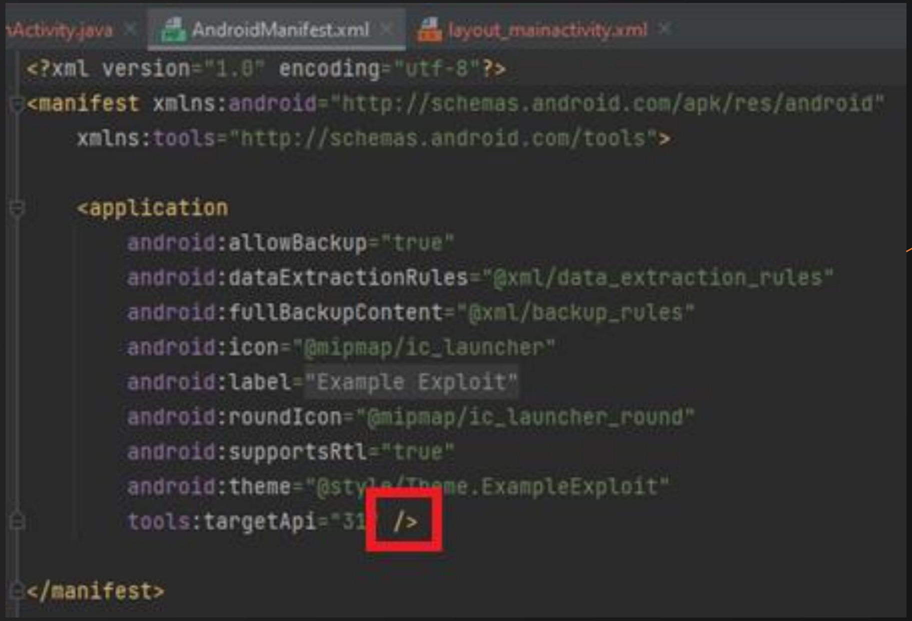


```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Example Exploit"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExampleExploit"
        tools:targetApi="31" />

</manifest>
```

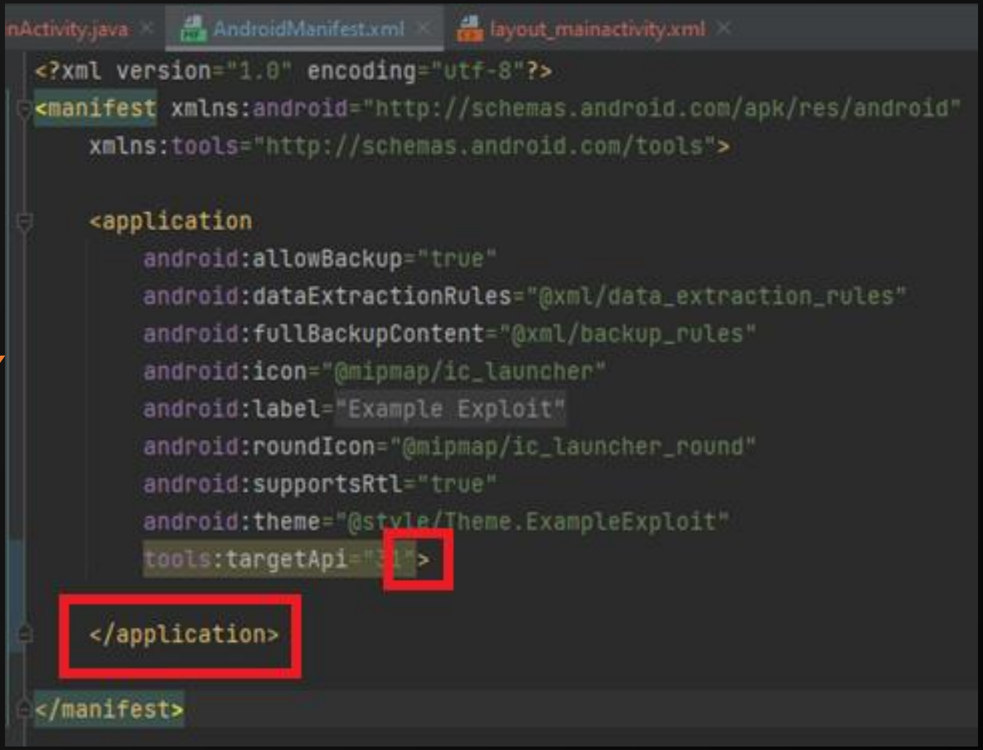
MALICIOUS ERECTION LLC

Tooling – Android Studio and Example Exploit

- Adjust the `Manifest.xml` file so that the "application" XML tag ends with `</application>`



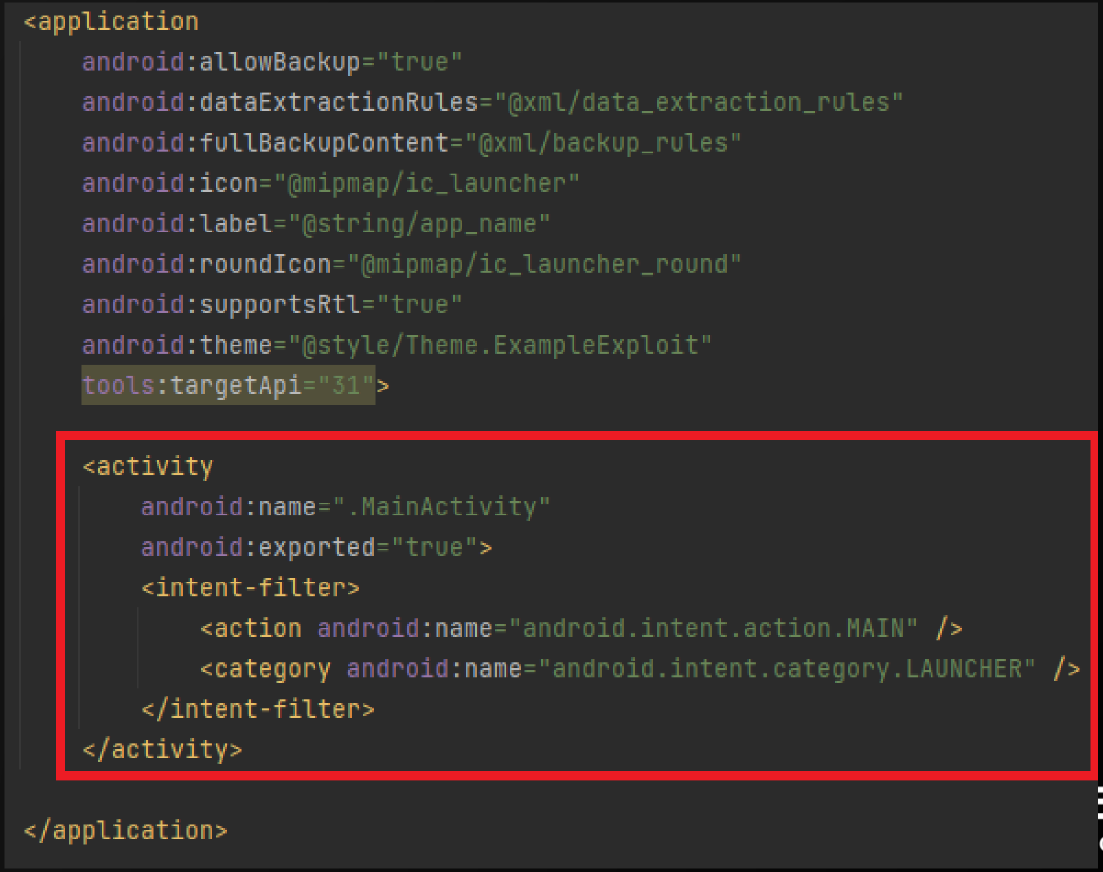
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Example Exploit"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExampleExploit"
        tools:targetApi="31" />

</manifest>
```



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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Example Exploit"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExampleExploit"
        tools:targetApi="31">

    </application>

</manifest>
```

Tooling – Android Studio and Example Exploit

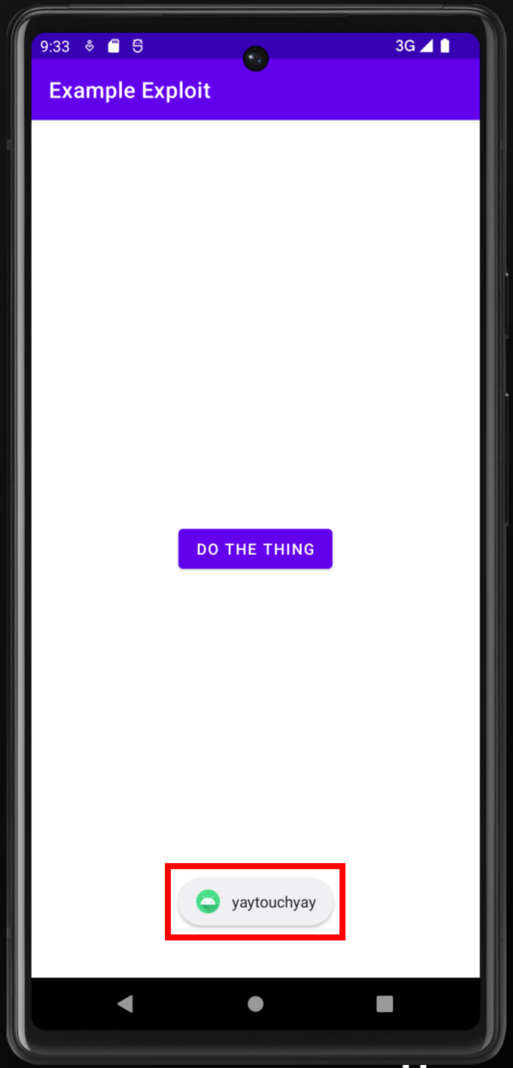
- Under the "application" XML tag, add an "activity" tag with the proper intent filter so that the `MainActivity` Java class is the "launcher" activity for the application
- Make sure `MainActivity` is exported



```xml
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.ExampleExploit"
    tools:targetApi="31">

    <activity
        android:name=".MainActivity"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

</application>
```

Tooling – Android Studio and Example Exploit

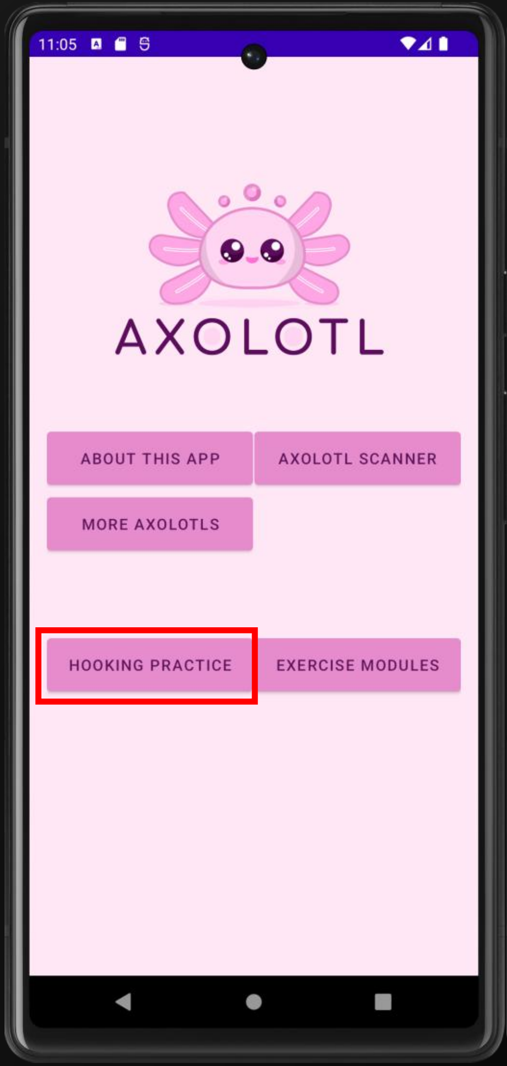
- To test and make sure Example Exploit launches correctly, compile and launch it
- Once the application is launched, touch the button and make sure the Toast message appears

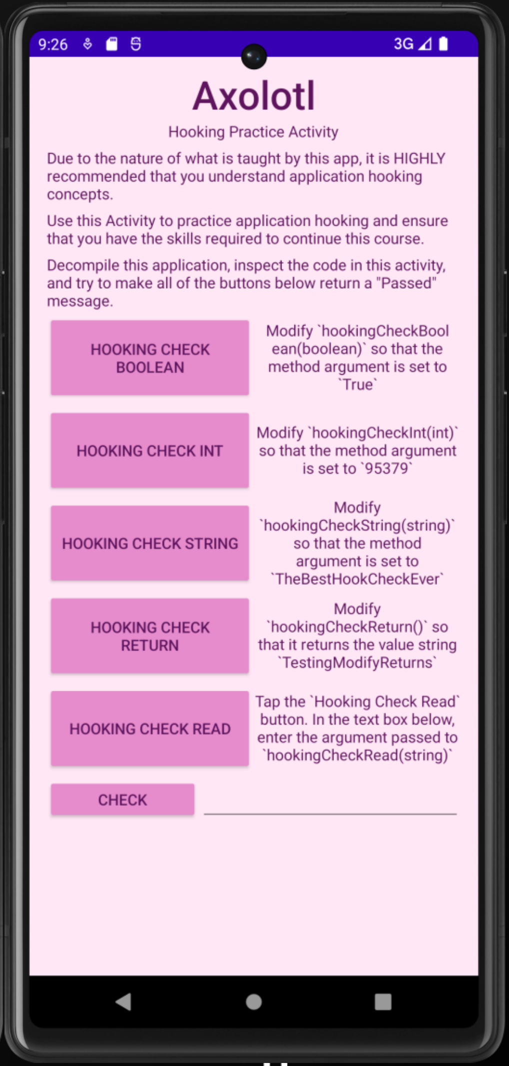


MILLICIOUS ERECTION LLC

Exercise 0 – Application Hooking

- Lets make sure you can reverse engineer and hook into Android applications
- Axolotl contains an area where you can practice your hooking techniques
  - Main menu screen -> Tap “Hooking Practice”
- The goal is to:
  - Reverse engineer the Activity `com.maliciouserection.axolotl.activity.hooking_check`
  - Hook into the appropriate methods
  - Modify the data that is processed in each method




9:26

Axolotl

Hooking Practice Activity

Due to the nature of what is taught by this app, it is HIGHLY recommended that you understand application hooking concepts.

Use this Activity to practice application hooking and ensure that you have the skills required to continue this course.

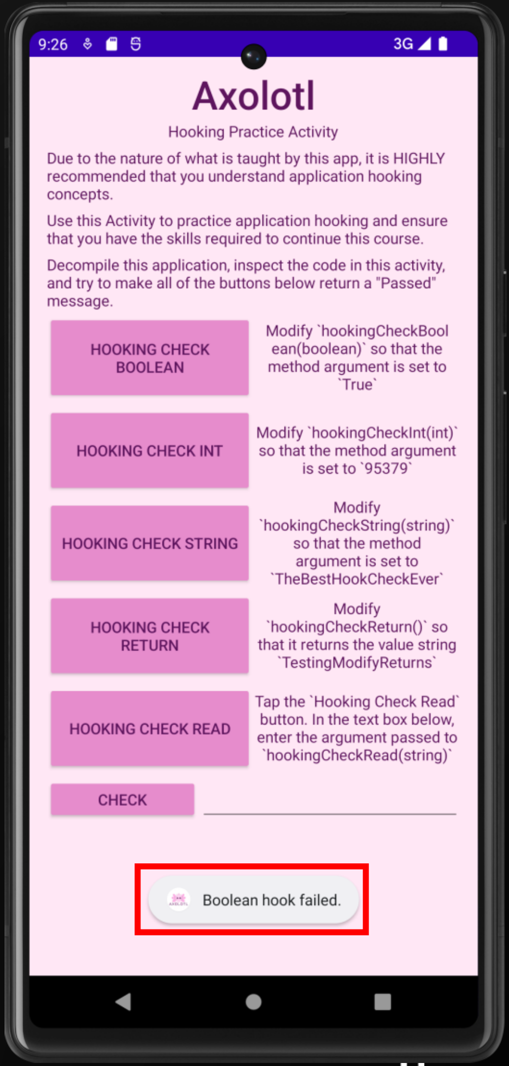
Decompile this application, inspect the code in this activity, and try to make all of the buttons below return a "Passed" message.

HOOKING CHECK BOOLEAN	Modify `hookingCheckBoolean(boolean)` so that the method argument is set to `True`
HOOKING CHECK INT	Modify `hookingCheckInt(int)` so that the method argument is set to `95379`
HOOKING CHECK STRING	Modify `hookingCheckString(string)` so that the method argument is set to `TheBestHookCheckEver`
HOOKING CHECK RETURN	Modify `hookingCheckReturn()` so that it returns the value string `TestingModifyReturns`
HOOKING CHECK READ	Tap the `Hooking Check Read` button. In the text box below, enter the argument passed to `hookingCheckRead(string)`

CHECK

Exercise 0 – Application Hooking

- When you tap each button, a Toast message will appear
  - This indicates if a method modification was successful
- Tasks:
  - Modify `hookingCheckBoolean(boolean)` so that the method argument is set to `True`
  - Modify `hookingCheckInt(int)` so that the method argument is set to `95379`
  - Modify `hookingCheckString(String)` so that the method argument is set to `TheBestHookCheckEver`
  - Modify `hookingCheckReturn()` so that it returns the String value `TestingModifyReturns`
  - Tap the “Hooking Check Read” button; read the argument which is passed to the method `hookingCheckRead(String)` and enter it in the text box at the bottom of the Activity, then tap “Check”
- Once completed, proceed through the rest of the course


9:26 3G

Axolotl

Hooking Practice Activity

Due to the nature of what is taught by this app, it is HIGHLY recommended that you understand application hooking concepts.

Use this Activity to practice application hooking and ensure that you have the skills required to continue this course.

Decompile this application, inspect the code in this activity, and try to make all of the buttons below return a "Passed" message.

HOOKING CHECK BOOLEAN	Modify `hookingCheckBoolean(boolean)` so that the method argument is set to `True`
HOOKING CHECK INT	Modify `hookingCheckInt(int)` so that the method argument is set to `95379`
HOOKING CHECK STRING	Modify `hookingCheckString(string)` so that the method argument is set to `TheBestHookCheckEver`
HOOKING CHECK RETURN	Modify `hookingCheckReturn()` so that it returns the value string `TestingModifyReturns`
HOOKING CHECK READ	Tap the `Hooking Check Read` button. In the text box below, enter the argument passed to `hookingCheckRead(string)`

CHECK

Boolean hook failed.

MILLICIOUS ERECTION LLC