
RANDORISEC

# Mobile Hacking Workshop – Android

Or how to resolve the OWASP Android Crackmes

*May 28th 2020*

# Objectives

- Learn how to perform basic Android testing using
  - apktool
  - JADX
  - Frida
  - Ghidra
  - Objection
  - Mobexler

# Targets

- Examples based on OWASP Crackmes
  - Android UnCrackable Level 1
    - https://github.com/OWASP/owasp-mstg/raw/master/Crackmes/Android/Level_01/UnCrackable-Level1.apk
  - Android UnCrackable Level 2
    - https://github.com/OWASP/owasp-mstg/raw/master/Crackmes/Android/Level_02/UnCrackable-Level2.apk
  - Android UnCrackable Level 3
    - https://github.com/OWASP/owasp-mstg/raw/master/Crackmes/Android/Level_03/UnCrackable-Level3.apk

# GitHub

- All needed or helpful files are available here
  - https://github.com/randorisec/workshops/
- Please clone this repository
  - `git clone https://github.com/randorisec/workshops.git`
- For this workshop, only use the files inside the **Android** folder

# Setup

# Setup – Intro

- For this workshop, we are going to use
  - Mobexler Lite virtual machine
    - https://mobexler.com/download.htm
  - Android Studio
    - With an emulator (later on this workshop)
    - Or a rooted device eventually if you want!
  - Apktool
  - JADX
  - Frida
  - Ghidra
  - Objection

# Setup – Mobexler

**Default password: 12345**

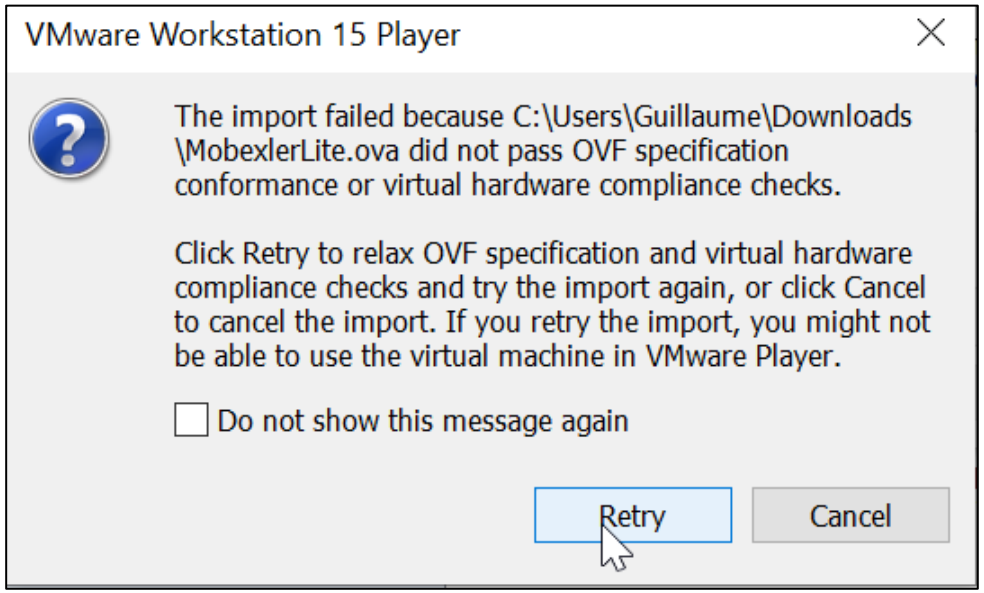
- “A Mobile Application Penetration Testing Platform”
- Similar to a Kali Linux machine but focused on mobile apps
  - Android and iOS
- Setup Guide
  - https://mobexler.com/setup.htm
- This machine includes most common tools for testing Android apps
  - Frida, JADX, JD-GUI, Android Studio, Burp Suite, MobSF, Objection, Ghidra, etc.
  - https://mobexler.com/tools.htm

# Setup – Mobexler

- Mobexler and VMWare
  - For this workshop, use only the Mobexler Lite version
    - https://drive.google.com/file/d/1r2exB-4OyBW0vRKFAPY7ls7ezX2ioXtl/view?usp=sharing
  - **Don’t use Virtual Box!!! Use VMWare Player or Workstation**
    - https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html
    - Note: Virtualization feature inside the VM is not available with VirtualBox if you use an Intel CPU…

# Setup – Mobexler

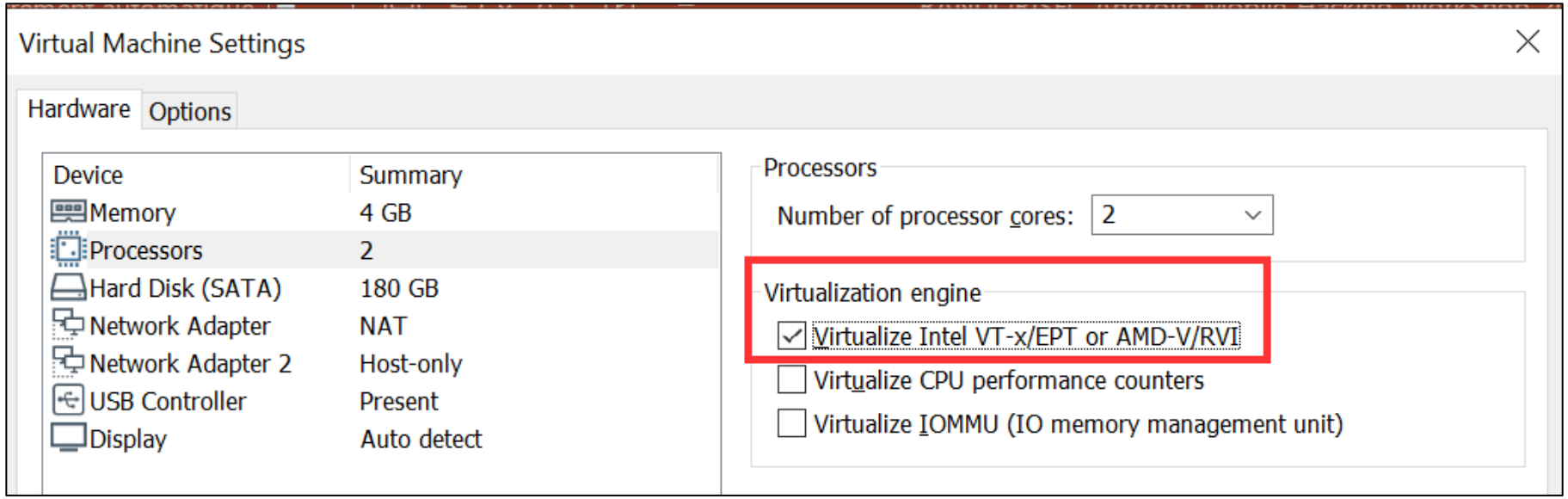
- Import the OVA file using VMWare
  - If an error message appears saying the OVA file didn't pass the OVF specifications, please click retry. It should work ☺



# Setup – Mobexler

## Before to launch the VM

- Edit the Virtual Machine hardware settings
  - Memory: At least 4 Go
  - Processors: Enable "Virtualize Intel VT-x/EPT or AMD-V/RVI“



# Setup – Mobexler

- VMWare Tools (Optional)
  - Install the VMWare tools
    - https://linuxconfig.org/install-vmware-tools-on-ubuntu-18-04-bionic-beaver-linux
    - `sudo apt install open-vm-tools-desktop`

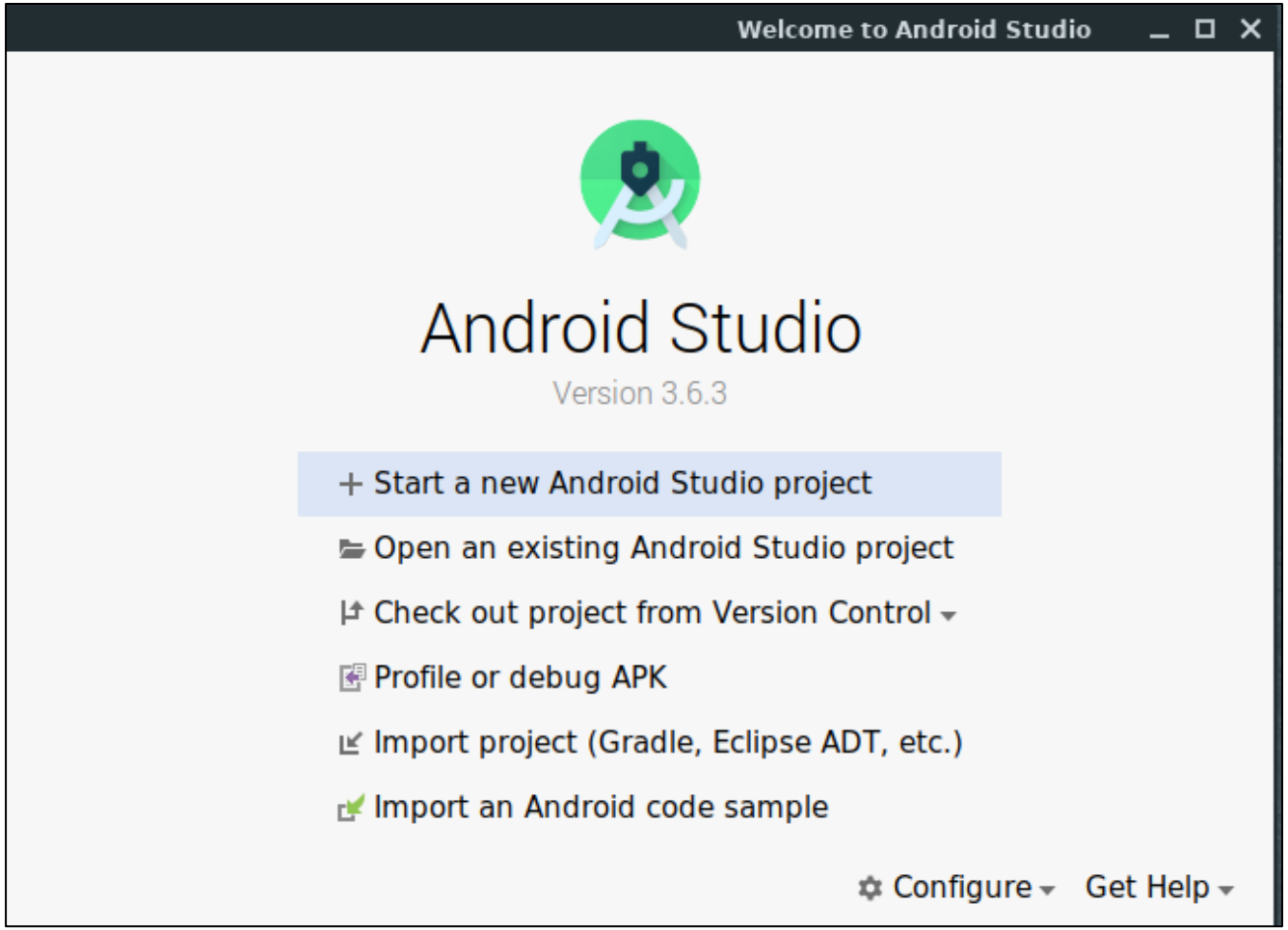
- **/!\ Enable KVM permissions for Android Studio /!**

```
1. sudo apt install qemu-kvm
2. sudo adduser MobexlerLite kvm
3. reboot
```

# Setup – Android Studio

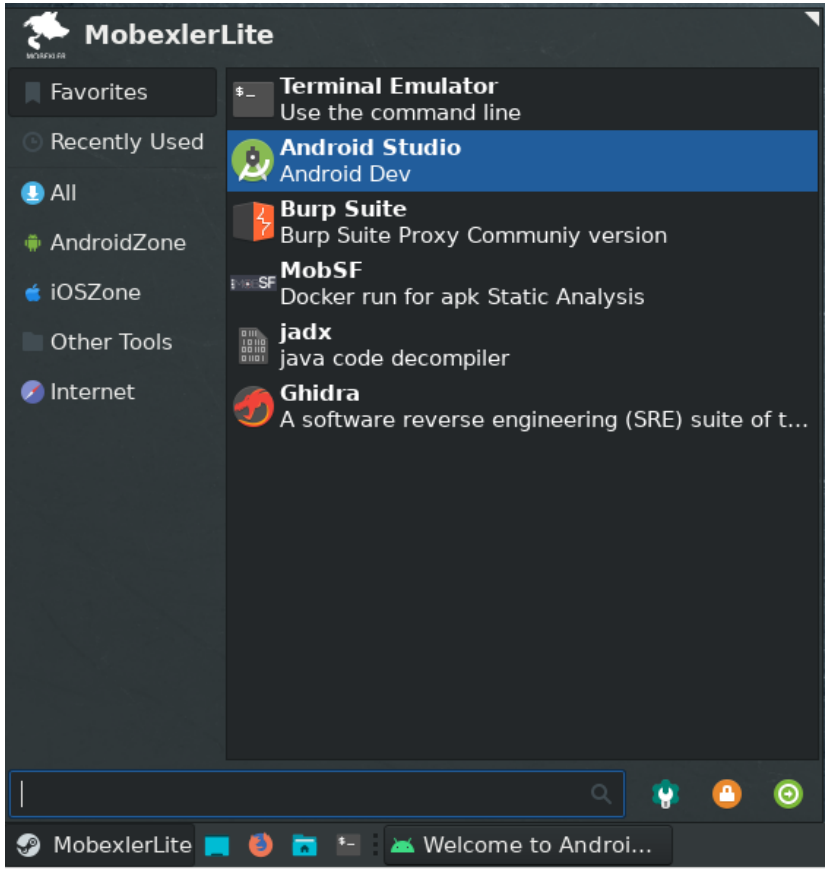
https://developer.android.com/studio/

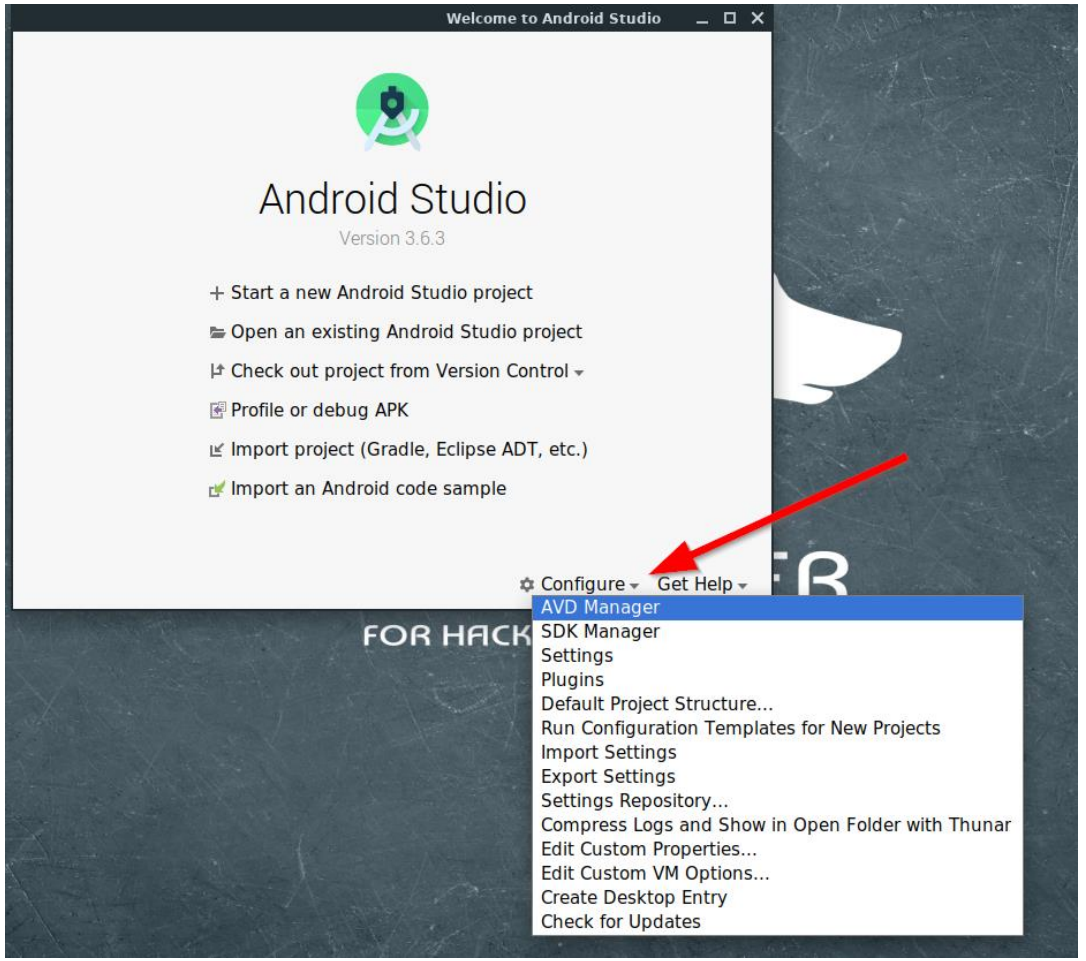
- Default IDE
  - Create Android apps
  - Debug apps
  - Logcat
  - Manage emulators
  - …



# Setup – Android Studio

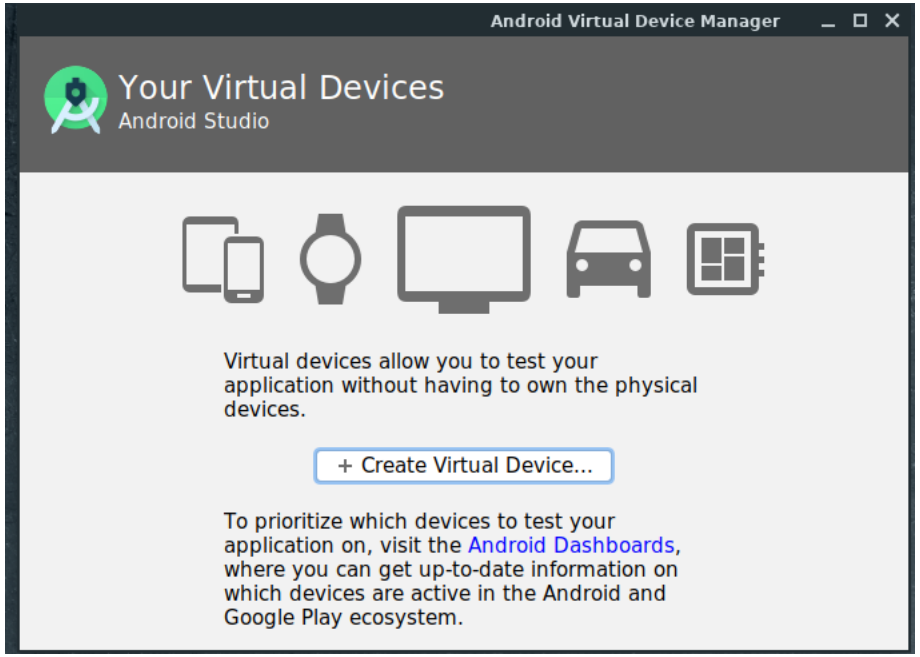
- Let's create an emulator!

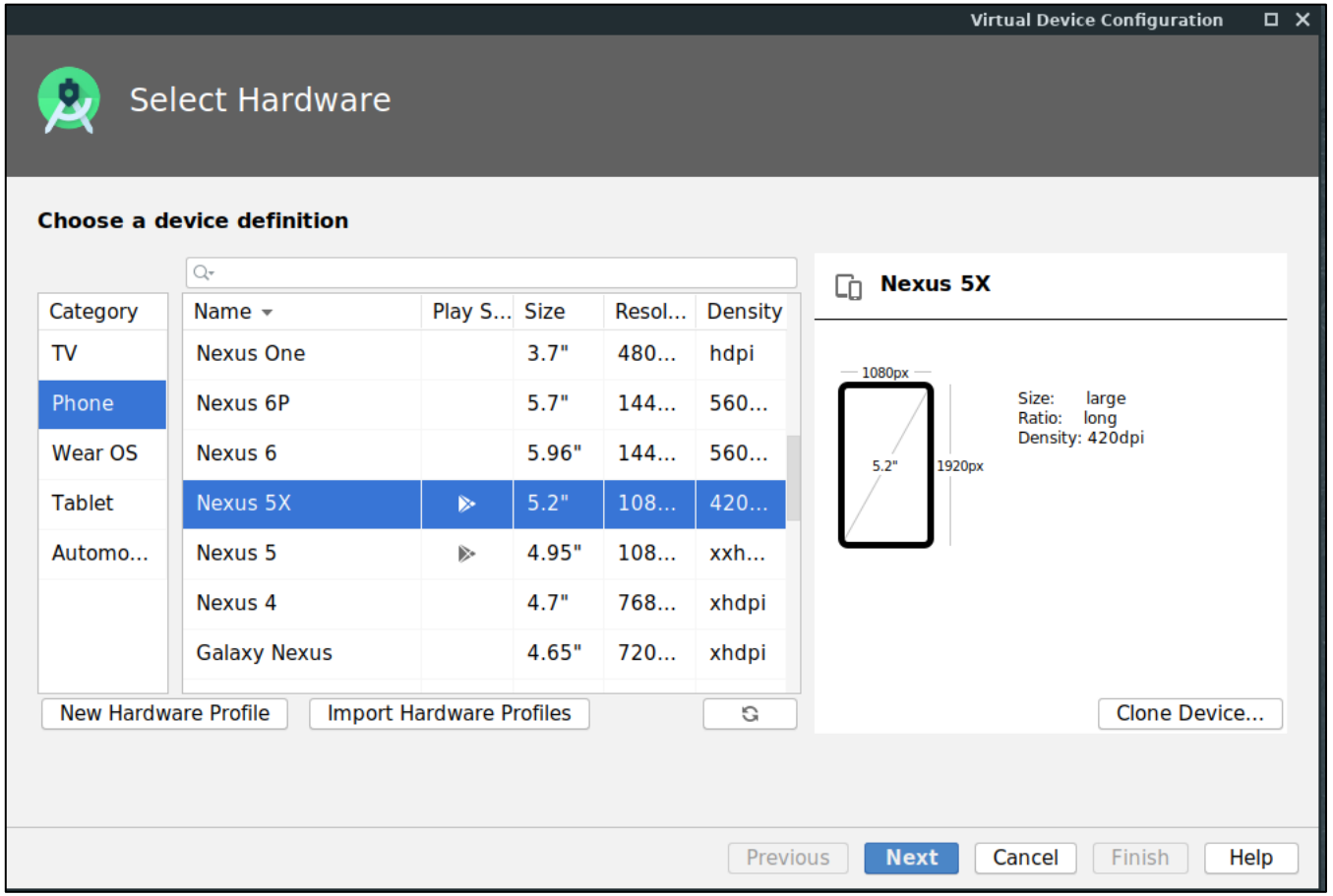


# Setup – Android Studio

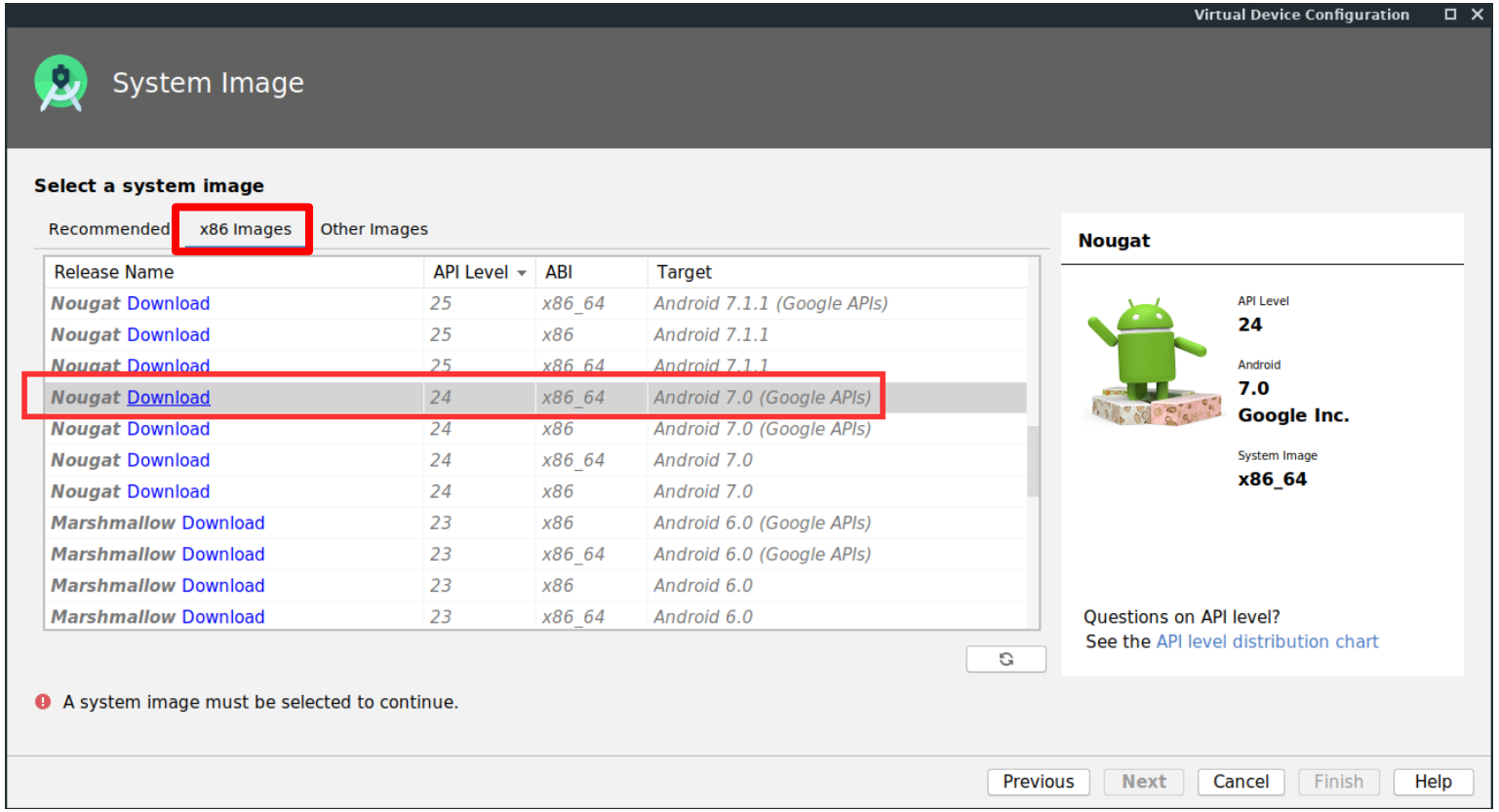
- Create a virtual device
  - Select “Nexus 5X”



# Setup – Android Studio

- Select a system image
  - **x86 images**
  - Release Name
    - Nougat
  - API Level
    - 24
  - ABI
    - X86_64
  - Target
    - Android 7.0 (**Google APIs**)



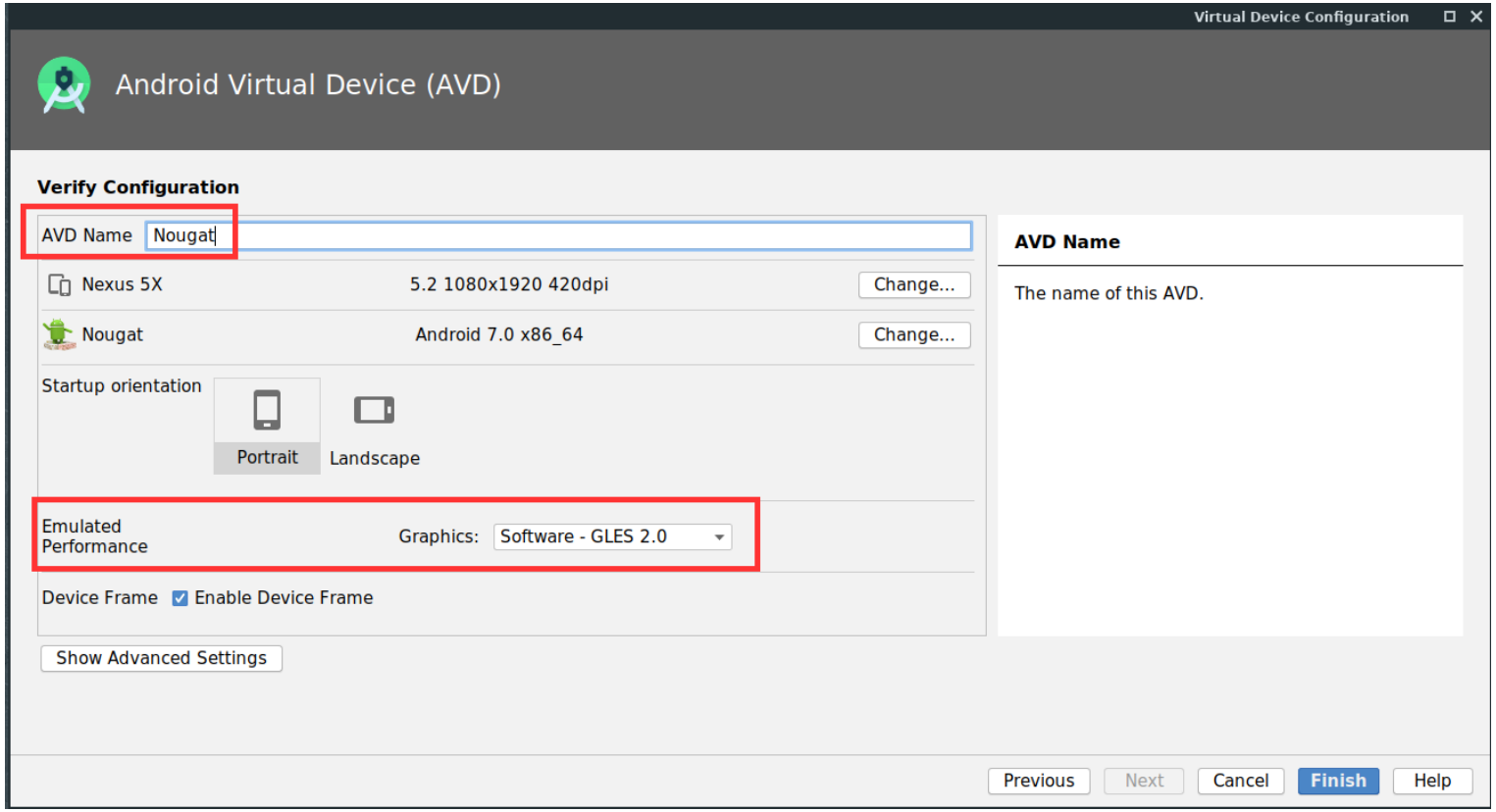
# Setup – Android Studio

- Download the system image
  - Click Download
- After the download, you can click Next ☺

- Tip: Google Play images are **NOT rooted**
  - Those are production releases
  - You need to root them if you want root privileges
- Tip2: Select Google APIs images if you want root privileges

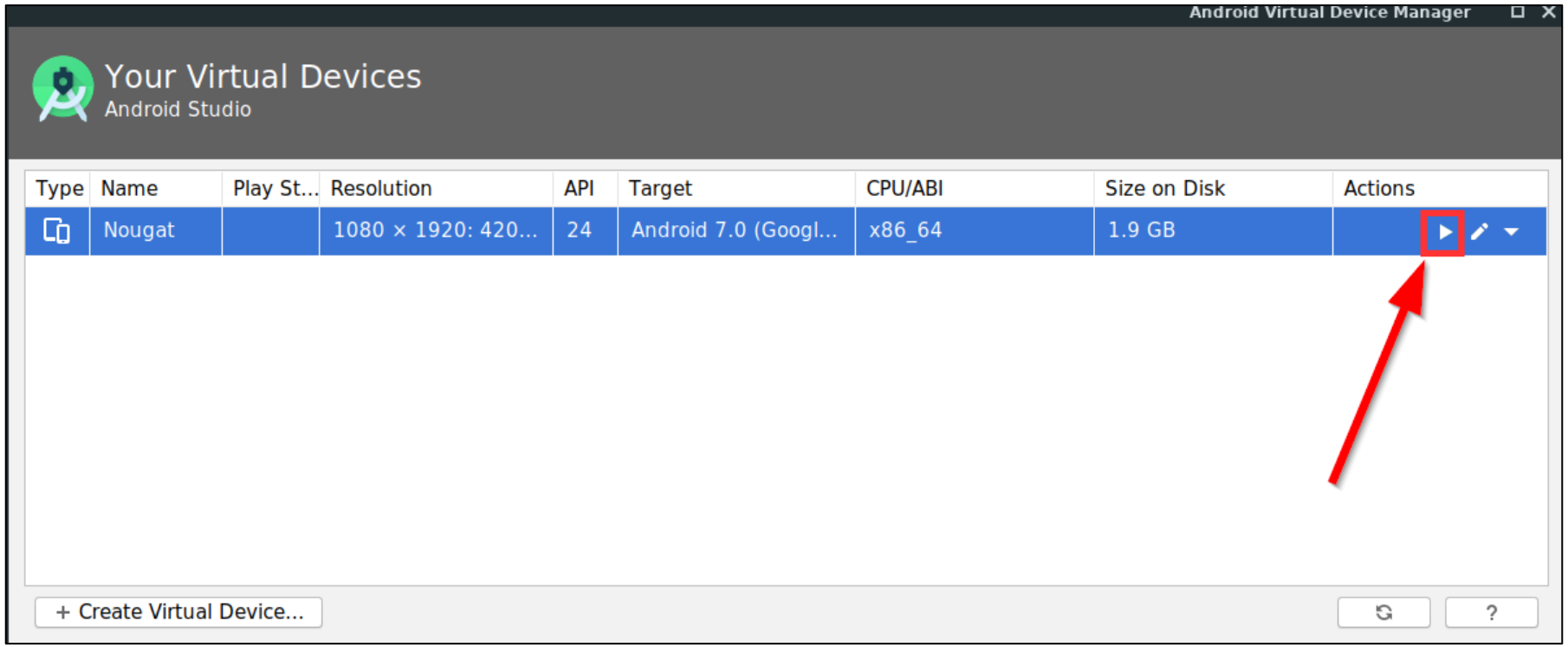
# Setup – Android Studio

- Set-up the AVD Name
  - **Nougat**
- Modify the Graphics performance
  - **Software - GLES 2.0**



# Setup – Android Studio

- Your emulator is ready!
  - You can launch it using the play button



# Setup – adb

- Command line tool to interact with an Android device (virtual or physical)
  - Included in the Android Sdk
- Get a shell
  - `adb shell`
- Run a command
  - `adb shell [cmd]`
- Restart adb with root privileges
  - `adb root`
- List Android devices connected
  - `adb devices`
- Connect through USB
  - `adb -d shell`
- Connect through TCP/IP
  - `adb -e shell`
- Copy local file to device
  - `adb push [local] [device]`
- Copy file from remote device
  - `adb pull [remote] [local]`

# Setup – adb

- Examples

```
MobexlerLite Terminal
File  Edit  View  Terminal  Tabs  Help
MobexlerLite ~ adb devices
List of devices attached
emulator-5554   device

MobexlerLite ~ adb shell
generic_x86_64:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),
3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc) context=u:r:shell:s0
generic_x86_64:/ $ exit
MobexlerLite ~ adb root
restarting adbd as root
MobexlerLite ~ adb shell id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_
admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc) context=u:r:su:s0
MobexlerLite ~ adb install UnCrackable-Level1.apk
Performing Streamed Install
Success
MobexlerLite ~
```

# Setup – apktool

https://ibotpeaches.github.io/Apktool

- "A tool for reverse engineering Android APK files"
  - Decode resource files (XMLs, DEX files, etc.)
  - Rebuild APK

- The version used by Mobexler is not up-to-date
  - Please run the following script to update apktool (Github repo)

```
bash update-apktool-2.4.1.sh
```

    - https://ibotpeaches.github.io/Apktool/install/

# Setup – apktool

- Tampering or modifying your app could be useful to bypass some security mechanisms
- Decompile and recompile an app with apktool (1/2)
  1. Decode the app with apktool

```
apktool d myapp.apk -o myapp
```

  2. Modify the smali code inside myapp/smali and/or modify the resources files (e.g. AndroidManifest.xml)

# Setup – apktool

https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

- Example of smali code

```
.class public Lsg/vantagepoint/a/b;
.super Ljava/lang/Object;

.method public static a(Landroid/content/Context;)Z
.locals 0

invoke-virtual {p0}, Landroid/content/Context;-
>getApplicationContext()Landroid/content/Context;
move-result-object p0

if-eqz p0, :cond_0
const/4 p0, 0x1
return p0
:cond_0
const/4 p0, 0x0
return p0
.end method
```

# Setup – apktool

- Decompile and recompile an app with apktool (2/2)
  3. Recompile the app

```
apktool b myapp -o mynewapp.apk
```

  4. Sign your app with a valid certificate using **jarsigner**
     - Use the debug keystore provided by the Android SDK tools

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg
SHA1 -keystore $HOME/.android/debug.keystore -
storepass android mynewapp.apk androiddebugkey
```

  5. (Optional) Use **zipalign** to provide optimization to the APK

```
zipalign -fv 4 <input APK> <output APK>
```

# Setup – apktool

https://developer.android.com/studio/publish/app-signing

- Create your own keystore using the command line
  - One-liner example

```
keytool -genkeypair -dname "cn=John Doe, ou=Security,
o=Randorisec, c=FR" -alias mykeystore -keystore
myandroid.keystore -storepass keystorepass -validity
20000 -keyalg RSA -keysize 2048  -sigalg SHA1withRSA
```

  - Note: Using this command, the keystore password and the key password will be the same
  - Note2: By default, SHA256withDSA is used but jarsigner is not able to handle SHA256 algorithm. It is better to use SHA1 in this case

# Setup – apktool

https://developer.android.com/studio/publish/app-signing

- For this workshop, we are going to create this specific keystore

```
keytool -genkeypair -dname "cn=Daniel Kloo, ou=Budapest,
o=Bsides, c=HU" -alias budkey -keystore bud.keystore -
storepass bsides -validity 2000 -keyalg RSA -keysize
2048  -sigalg SHA1withRSA
```

- So for signing your APKs, you can use the following command

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -
keystore bud.keystore -storepass bsides <APK_file>
budkey
```

# Setup – apktool

- The script “generate-keystore.sh” performed the following command

```
keytool -genkeypair -dname "cn=Daniel Kloo, ou=Budapest,
o=Bsides, c=HU" -alias budkey -keystore bud.keystore -
storepass bsides -validity 2000 -keyalg RSA -keysize
2048  -sigalg SHA1withRSA
```

- Please move or copy the “bud.keystore” file to your home directory
  - /home/mobexlerlite

# Setup – apktool

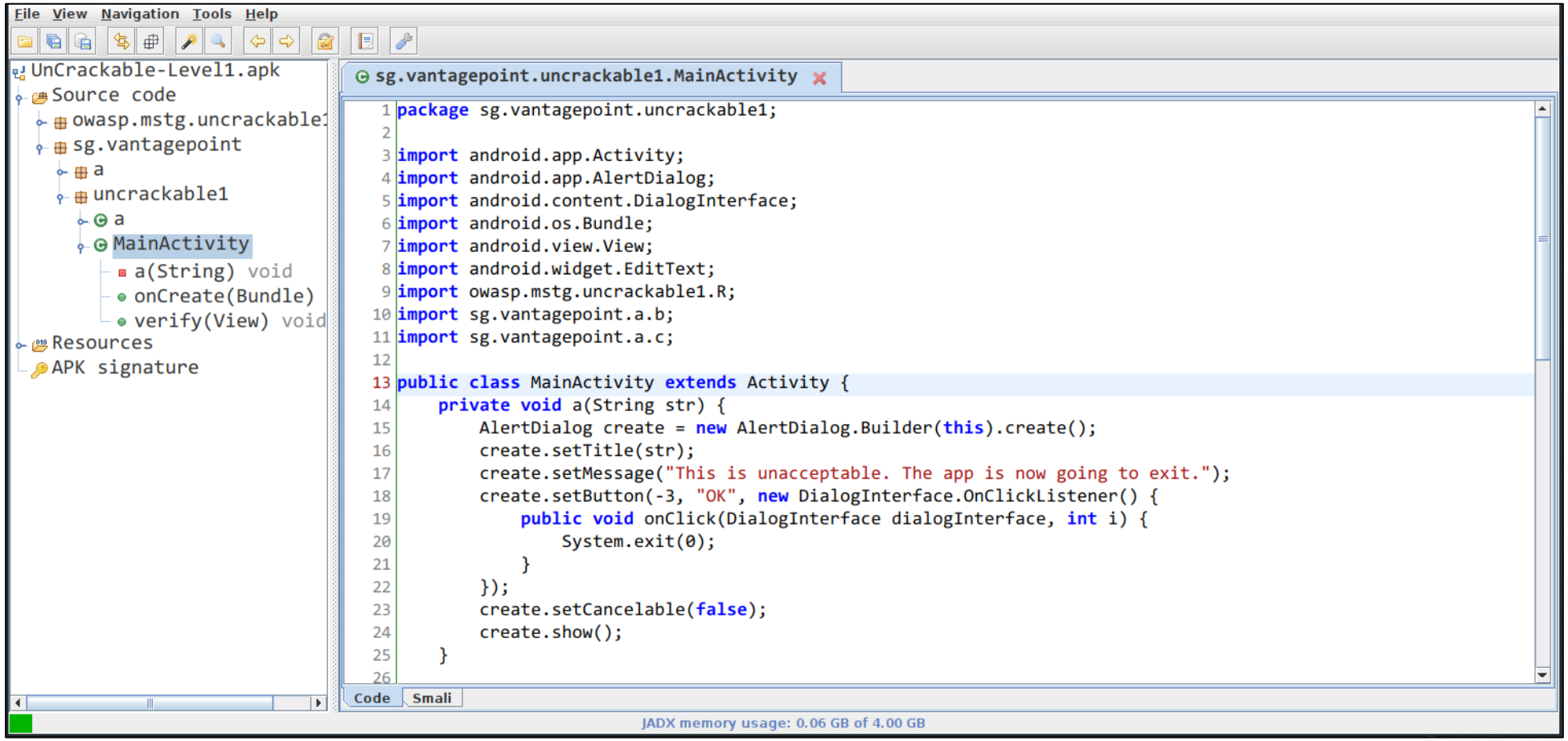
- The script “sign-apk.sh” sign an APK using the previously created keystore

```
MobexlerLite ~ | owasp bash sign-apk.sh newUl1.apk
 updating: META-INF/BUDKEY.SF
 updating: META-INF/BUDKEY.RSA
  signing: AndroidManifest.xml
  signing: classes.dex
  signing: res/layout/activity_main.xml
  signing: res/menu/menu_main.xml
  signing: res/mipmap-hdpi/ic_launcher.png
  signing: res/mipmap-mdpi/ic_launcher.png
  signing: res/mipmap-xhdpi/ic_launcher.png
  signing: res/mipmap-xxhdpi/ic_launcher.png
  signing: res/mipmap-xxxhdpi/ic_launcher.png
  signing: resources.arsc

>>> Signer
    X.509, CN=Daniel Kloo, OU=Budapest, O=Bsides, C=HU
    [trusted certificate]

jar signed.

Warning:
The signer's certificate is self-signed.
```

# Setup – JADX

https://github.com/skylot/jadx

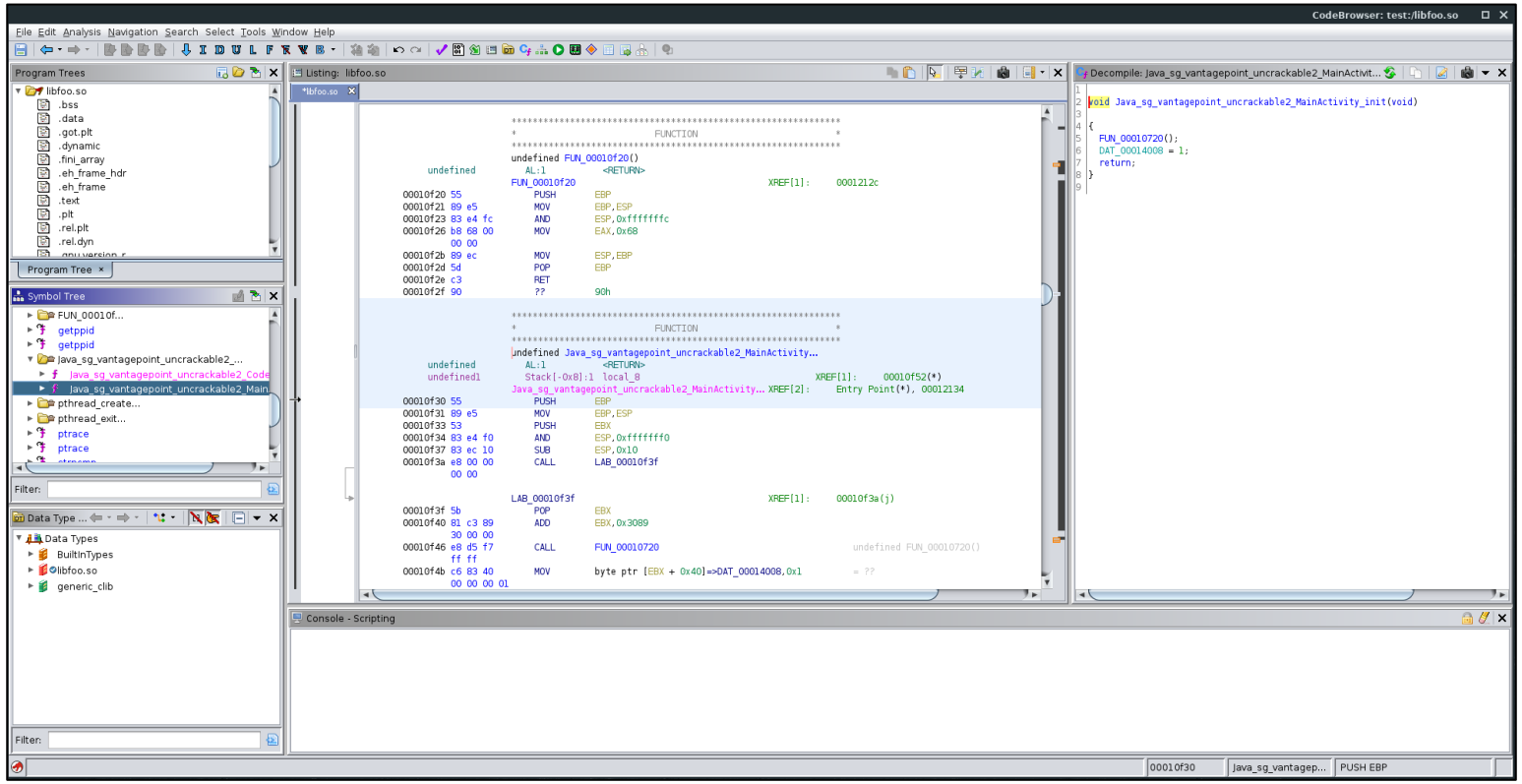
- Dex and APK files decompiler to Java



# Setup – Ghidra

https://ghidra-sre.org/

- "A software reverse engineering (SRE) suite of tools developed by NSA"
  - Useful to reverse engineer native libraries



# Setup – Native Code

https://developer.android.com/ndk/guides

- An Android app can embed native code
  - Usually written in C/C++
  - Available as a shared library (.so)
  - Lot of information on the Native Development Kit (NDK)

- Java Native Interface (aka JNI)
  - Provides a way for the bytecode to interact with native code
  - From Java/Kotlin, it is possible to call C/C++ methods
  - But it is also possible to do the same from C/C++ to call Java/Kotlin methods

# Setup – Native Code

- Example of Java code
  - Just define your native functions with the `native` keyword

```
// First, you need to load your library
static {
    System.loadLibrary("native-lib");
}

// Then define your native functions
public native String myNativeFunction();

public native void launchSometing();
```

# Setup – Native Code

- Example of native code
  - Native method should respect a specific naming

```
#include <jni.h>
#include <string>
extern "C"
JNIEXPORT jstring JNICALL
Java_my_package_name_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF(hello.c_str());
}
```

# Setup – Frida

https://www.frida.re/docs/home/

- Frida
  - It's a dynamic code instrumentation toolkit
  - "It lets you inject snippets of JavaScript or your own library into native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX"
- Rooted device
  - Need to install a Frida server on the Android device
- Non rooted device
  - Need to repackage the targeted app with the frida-gadget module

# Setup – Frida

- Install Frida on your system
  - Easy way with pip

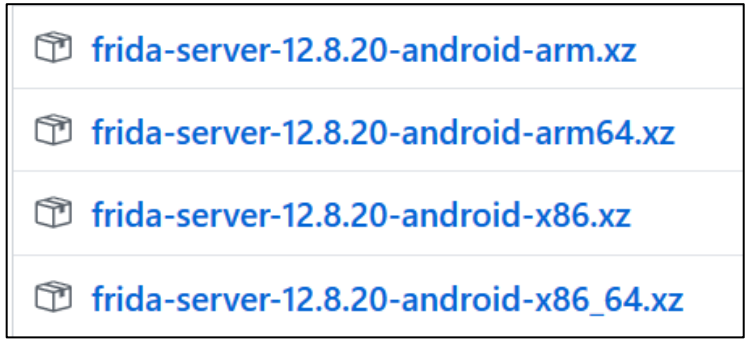
```
pip install frida frida-tools (Python bindings)
pip install –upgradable frida (obtain the last version)
```

  - Or compile from the sources
    - https://github.com/frida/frida
- Then, install Frida binary on the Android device (or emulator)
  - If you are not sure which version needed

```
$ adb shell getprop ro.product.cpu.abi
x86_64
```

# Setup – Frida

- Download frida-server for the correct architecture
  - https://github.com/frida/frida/releases




```
wget https://github.com/frida/frida/releases/download/12.8.20/frida-server-12.8.20-android-x86_64.xz
xz -d frida-server-12.8.20-android-x86_64.xz
```

- Then, upload the binary file and execute it

```
adb push frida-server-12.8.20-android-x86_64 /data/local/tmp/frida-server
adb shell "chmod 755 /data/local/tmp/frida-server"
adb shell "/data/local/tmp/frida-server"
```

- Note: adb needs to run with root privileges!

# Setup – Frida Tools

- frida-ps
  - Get the list of running processes

    -U option for USB devices or emulators

```
$ frida-ps -U
 PID  Name
----  ------------------------------------------------
4527  adbd
4248  android.process.acore
1375  audioserver
1376  cameraserver
4625  com.android.defcontainer
4654  com.android.gallery3d
1745  com.android.inputmethod.latin
2512  com.android.launcher3
1916  com.android.phone
```

# Setup – Frida Tools

- frida-ps
  - Get the list of installed applications

```
$ frida-ps -U -i
 PID  Name                           Identifier
----  -----------------------------  ------------------------------------
1745  Android Keyboard (AOSP)        com.android.inputmethod.latin
2301  Android Services Library       com.google.android.ext.services
1625  Android System                 android
1625  Call Management                com.android.server.telecom
5516  Chrome                         com.android.chrome
5240  ConfigUpdater                  com.google.android.configupdater
5215  Download Manager               com.android.providers.downloads
1625  Fused Location                 com.android.location.fused
2371  Google App                     com.google.android.googlequicksearchbox
5479  Google Partner Setup           com.google.android.partnersetup
1762  Google Play services           com.google.android.gms
```

# Setup – Frida Tools

- By default, Frida tries to attach to an existing process
  - `frida -U com.android.chrome`
- To spawn an application, use the -f option as follow
  - `frida -U -f com.android.chrome`
    - By default, the process is paused
  - `frida -U -f com.android.chrome --no-pause`

# Setup – Frida Scripting

- Python bindings are provided with Frida
  - However, the hooks need to be written in JavaScript ☹
- Here are the basics Frida functions
  - **`Java.perform`**`(function() {//your code here});`
    - Function allowing to perform actions on the Java code
  - **`Java.use`**`(class_name)`
    - Function allowing to use a specific Java class
  - **`overload`**
    - Overload a specific method
  - **`implementation`**
    - In order to modify the implementation of the method

# Setup – Frida Scripting

- The most convenient way is to use scripts
  - `frida -U -l myscript.js com.android.chrome`
  - Here is an example allowing to overwrite the `onResume` function

```
// hook a function
Java.perform(function () {
    //Declare the Activity class as a variable
    var Activity = Java.use("android.app.Activity");
    //Re-implement the onResume function
    Activity.onResume.implementation = function () {
        console.log("[*] onResume() got called!");
        this.onResume();
    };
});
```

# Setup

DEMO

# UnCrackable Level 1

# UnCrackable Level 1

https://github.com/OWASP/owasp-mstg/tree/master/Crackmes#uncrackable-app-for-android-level-1

- Description
  - “This app holds a secret inside. Can you find it?”
  - Sensitive part of the code is obfuscated

- We are going to use different ways to solve this crackme
  1. Only by tampering the app
  2. Using Frida with a rooted device
  3. Using Frida with the lib-gadget

# UnCrackable Level 1

- Let's have a look at the code
  - Using jadx, let's analyze the decompiled Java code
    - `jadx-gui Uncrackable-level1.apk`

- First, let's have a look at the AndroidManifest.xml file
  - The idea is to find the Main activity

```
4     <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
5         <activity android:label="@string/app_name" android:name="sg.vantagepoint.uncrackable1.MainActivity">
6             <intent-filter>
7                 <action android:name="android.intent.action.MAIN"/>
8                 <category android:name="android.intent.category.LAUNCHER"/>
9             </intent-filter>
10        </activity>
11    </application>
12</manifest>
```

# UnCrackable Level 1 – Root/Debug Detection

- The app implements different protections to avoid running
  - If the device is rooted
  - If a debugger is attached to the app

```
27    /* access modifiers changed from: protected */
28    public void onCreate(Bundle bundle) {
29        if (c.a() || c.b() || c.c()) {
30            a("Root detected!");
31        }
32        if (b.a(getApplicationContext())) {
33            a("App is debuggable!");
34        }
35        super.onCreate(bundle);
36        setContentView(R.layout.activity_main);
37    }
```

- We can see those checks on
  - Class: **sg.vantagepoint.uncrackable1.MainActivity**
  - Function: **OnCreate()**

# UnCrackable Level 1 – Root/Debug Detection

- To identify if the device is rooted, the app checks
  - If the su binary exists
  - If the build contains “test-keys”
  - If common files used for rooting purposes exist on the device

- Class: **sg.vantagepoint.a.c**
- Functions: **a(), b() and c()**

```
1  package sg.vantagepoint.a;
2
3  import android.os.Build;
4  import java.io.File;
5
6  public class c {
7      public static boolean a() {
8          for (String file : System.getenv("PATH").split(":")) {
9              if (new File(file, "su").exists()) {
10                 return true;
11             }
12         }
13         return false;
14     }
15
16     public static boolean b() {
17         String str = Build.TAGS;
18         return str != null && str.contains("test-keys");
19     }
20
21     public static boolean c() {
22         for (String file : new String[]{"/system/app/Superuser.apk",
23             if (new File(file).exists()) {
24                 return true;
25             }
26         }
27         return false;
28     }
29 }
```

# UnCrackable Level 1 – Root/Debug Detection

- If the device is rooted or debugged, the app displays a dialog box and exits
- The same function is called on both cases
  - Class: **sg.vantagepoint.uncrackable1.MainActivity**
  - Function: **a()**

```
13 public class MainActivity extends Activity {
14     private void a(String str) {
15         AlertDialog create = new AlertDialog.Builder(this).create();
16         create.setTitle(str);
17         create.setMessage("This is unacceptable. The app is now going to exit.");
18         create.setButton(-3, "OK", new DialogInterface.OnClickListener() {
19             public void onClick(DialogInterface dialogInterface, int i) {
20                 System.exit(0);
21             }
22         });
23         create.setCancelable(false);
24         create.show();
25     }
```

# UnCrackable Level 1 – Root/Debug Detection

- Solutions to bypass root detection
  1. Easy way remove or hook the System.exit() call in order to avoid the app to exit
  2. Modify or hook the a() function on sg.vantagepoint.uncrackable1 MainActivity class in order to do nothing
  3. Modify or hook the a(),b() and c() from sg.vantagepoint.c class to return false

- Note: Same principle can be applied to bypass debug detection

# UnCrackable Level 1 – Hidden Secret

- A verify() function is called to check the user input
  - Class: **sg.vantagepoint.uncrackable1.MainActivity**
  - Function: **verify()**

```
39    public void verify(View view) {
40        String str;
41        String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
42        AlertDialog create = new AlertDialog.Builder(this).create();
43        if (a.a(obj)) {
44            create.setTitle("Success!");
45            str = "This is the correct secret.";
46        } else {
47            create.setTitle("Nope...");
48            str = "That's not it. Try again.";
49        }
50        create.setMessage(str);
51        create.setButton(-3, "OK", new DialogInterface.OnClickListener() {
52            public void onClick(DialogInterface dialogInterface, int i) {
53                dialogInterface.dismiss();
54            }
55        });
56        create.show();
57    }
```

# UnCrackable Level 1 – Hidden Secret

- Function containing the key and the passphrase encrypted
  - Class: **sg.vantagepoint.uncrackable1.a**
  - Function: **a()**

```
6  public class a {
7      public static boolean a(String str) {
8          byte[] bArr;
9          byte[] bArr2 = new byte[0];
10         try {
11             bArr = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"), Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=", 0));
12         } catch (Exception e) {
13             Log.d("CodeCheck", "AES error:" + e.getMessage());
14             bArr = bArr2;
15         }
16         return str.equals(new String(bArr));
17     }
18
19     public static byte[] b(String str) {
20         int length = str.length();
21         byte[] bArr = new byte[(length / 2)];
22         for (int i = 0; i < length; i += 2) {
23             bArr[i / 2] = (byte) ((Character.digit(str.charAt(i), 16) << 4) + Character.digit(str.charAt(i + 1), 16));
24         }
25         return bArr;
26     }
27 }
```

# UnCrackable Level 1 – Hidden Secret

- The secret passphrase is encrypted using AES
  - Function to decrypt the secret
    - Class: **sg.vantagepoint.a.a**
    - Function: **a()**

```
 6 public class a {
 7     public static byte[] a(byte[] bArr, byte[] bArr2) {
 8         SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
 9         Cipher instance = Cipher.getInstance("AES");
10         instance.init(2, secretKeySpec);
11         return instance.doFinal(bArr2);
12     }
13 }
```

# UnCrackable Level 1 – Hidden Secret

- Solutions to retrieve the passphrase in cleartext
  1. Tamper the app in order to display the secret after the decryption call
  2. Hook the decryption function to retrieve the return value

# UnCrackable Level 1 – Code Tampering

## Bypass the root detection

1. Decompile the app using apktool

```
apktool d UnCrackable-Level1.apk
```

2. Edit the smali code in order to remove the System.exit() call

```
nano UnCrackable-
Level1/smali/sg/vantagepoint/uncrackable1/MainActivity\$1.smali
```

3. Remove the following lines

```
 const/4 p1, 0x0

 invoke-static {p1}, Ljava/lang/System;->exit(I)V
```

# UnCrackable Level 1 – Code Tampering

## Display the secret

4. Edit the smali code in order to display the secret using logcat

```
nano UnCrackable-Level1/smali/sg/vantagepoint/uncrackable1/a.smali
```

5. Modify the a() function by adding those 2 lines

```
const-string v2, "RESULT"

invoke-static {v2, v1}, Landroid/util/Log;-
>d(Ljava/lang/String;Ljava/lang/String;)I
```

6. Those 2 lines need to be added before this line

```
invoke-virtual {p0, v1}, Ljava/lang/String;-
>equals(Ljava/lang/Object;)Z
```

# UnCrackable Level 1 – Code Tampering

- The objective is to display the secret inside the logcat messages
- Here is the result of the modified smali code

```
 GNU nano 2.9.3                                    smali/sg/vantagepoint/uncrackable1/a.smali

    invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v0

    invoke-static {v1, v0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    move-object v0, v2

    :goto_0
    new-instance v1, Ljava/lang/String;

    invoke-direct {v1, v0}, Ljava/lang/String;-><init>([B)V

    const-string v2, "RESULT"

    invoke-static {v2, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I


    invoke-virtual {p0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result p0

    return p0
.end method
```

# UnCrackable Level 1 – Code Tampering

7. Build the new APK with apktool

```
apktool b UnCrackable-Level1 -o newUl1.apk
```

8. Sign the new APK with jarsigner

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -
keystore bud.keystore -storepass bsides newUl1.apk
budkey
```

9. Uninstall the original app

```
adb uninstall owasp.mstg.uncrackable1
```

10. Install the new APK

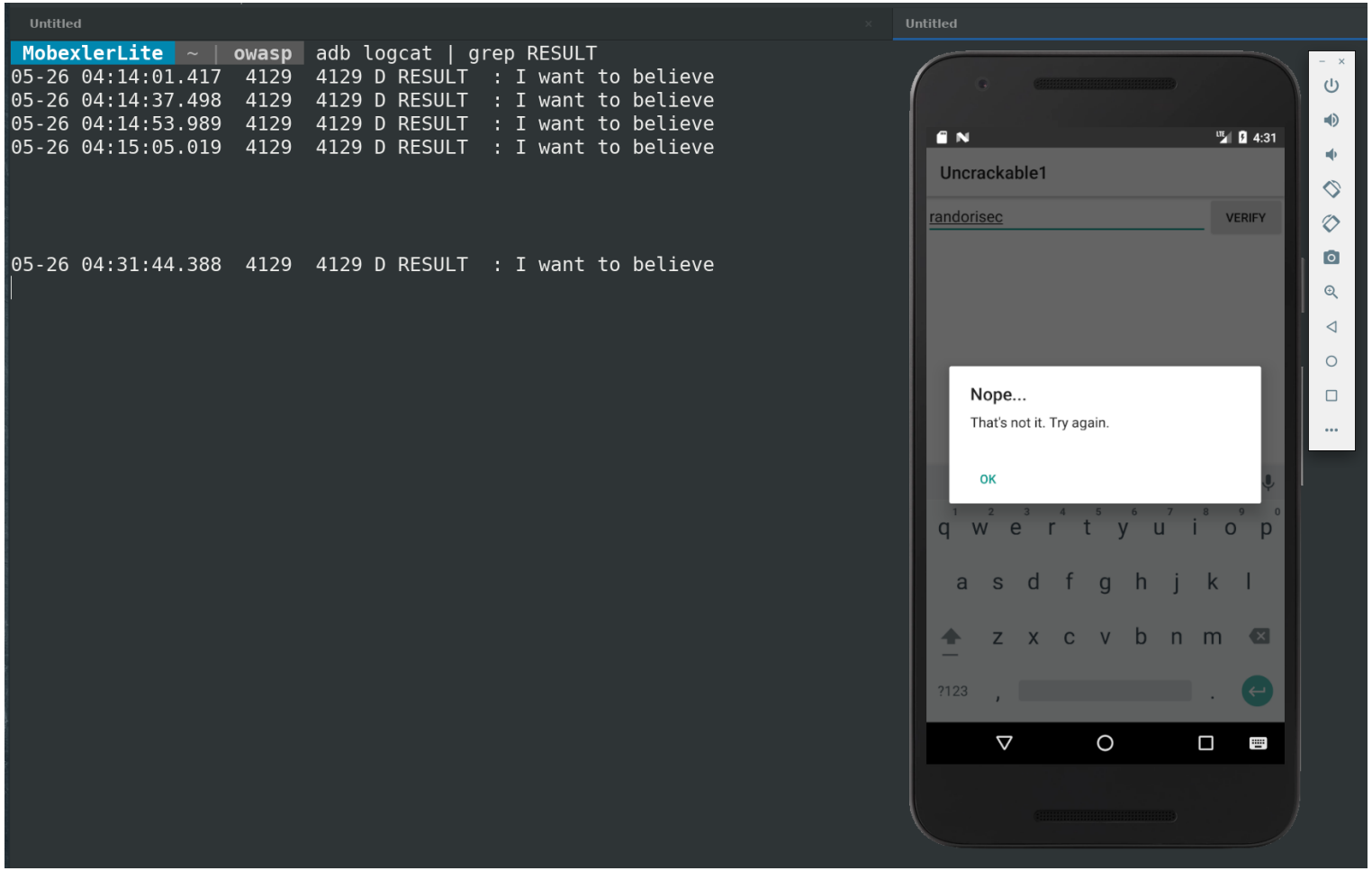
```
adb install newUl1.apk
```

# UnCrackable Level 1 – Code Tampering

- Launch logcat and grep the RESULT tag

```
adb logcat | grep RESULT
```

- Finally, launch the app and enter a wrong password



# UnCrackable Level 1 – Code Tampering

DEMO

# UnCrackable Level 1 – Frida (Rooted device)

- Bypass the root detection
  - A solution is to hook the System.exit() function to do nothing

**bypass-root-with-exit-ul1.js**

```
Java.perform(function () {
  console.log("Starting uncrackable1...");

  var sysexit = Java.use("java.lang.System");
  sysexit.exit.implementation = function() {
    // We avoid exiting the application
    console.log("System.exit() function was called!");
  };
});
```

# UnCrackable Level 1 – Frida (Rooted device)

- Another solution is to hook a(), b() and c() functions to return false

**bypass-root-ul1.js**

```
Java.perform(function () {
  console.log("Starting uncrackable1...");
  var root_class = Java.use("sg.vantagepoint.a.c");
  root_class.a.implementation = function() {
    console.log("a() function was called!");
    return false; };
  root_class.b.implementation = function() {
    console.log("b() function was called!");
    return false; };
  root_class.c.implementation = function() {
    console.log("c() function was called!");
    return false; };
});
```

# UnCrackable Level 1 – Frida (Rooted device)

- To retrieve the secret, we just need to hook the decryption function

```
Java.perform(function () {
    var aaClass = Java.use("sg.vantagepoint.a.a");
    // We modify the code in order to execute the method
    aaClass.a.implementation = function(arg1, arg2) {
        // Call the original function
        var retval = this.a(arg1, arg2);
        // Then , we just translate the byte array in string
        var password = '';
        for(var i = 0; i < retval.length; i++) {
            password += String.fromCharCode(retval[i]);
        }
        console.log("[*] Decrypted: " + password);
        return retval; };
});
```

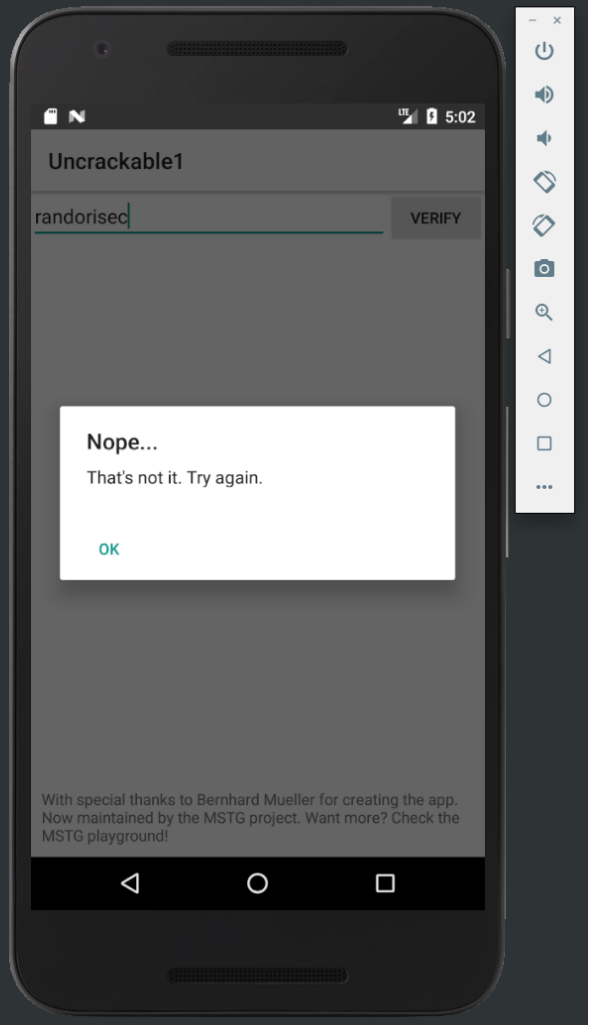
# UnCrackable Level 1 – Frida (Rooted device)

- By combining, the 2 previous techniques it is possible to retrieve the hidden secret by providing a wrong password
  - Use the solution-ul1.js Frida script

```
frida -U -l solution-ul1.js -f owasp.mstg.uncrackable1
--no-pause
```

# UnCrackable Level 1 – Frida (Rooted device)

```
MobexlerLite ~ | owasp frida -U -l solution-ul1.js -f owasp.mstg.uncrackable1 --no-pause
    ____
   / _  |   Frida 12.8.20 - A world-class dynamic instrumentation toolkit
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at https://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable1`. Resuming main thread!
[Android Emulator 5554::owasp.mstg.uncrackable1]-> Starting uncrackable1...
a() function was called!
b() function was called!
c() function was called!
[*] Decrypted: I want to believe
```



# UnCrackable Level 1 – Frida (Rooted device)

DEMO

# UnCrackable Level 1 – Frida (lib-gadget)

- It is possible to use Frida against a non rooted device!
  - It is very cool if you don’t want or can’t root the device
  - The app needs to be rebuilt with the lib-gadget library
- In short, the steps are
  1. Decode the app with apktool
  2. Add the lib-gadget library inside the app
  3. Modify the smali code to load the lib-gadget
  4. Add the INTERNET permission to the app
  5. Rebuild and sign the app
  6. PROFIT!

# UnCrackable Level 1 – Frida (lib-gadget)

- To resolve the crackme
  - We are just going to tamper the app in order to use lib-gadget
  - Then, we can use our previous Frida scripts

- The Hard way
  - Modify the APK by hand in order to inject the Frida gadget library
- The Easy way
  - Use objection tool to tamper the APK

# UnCrackable Level 1 – Frida (lib-gadget)

1. Decompile the app using apktool

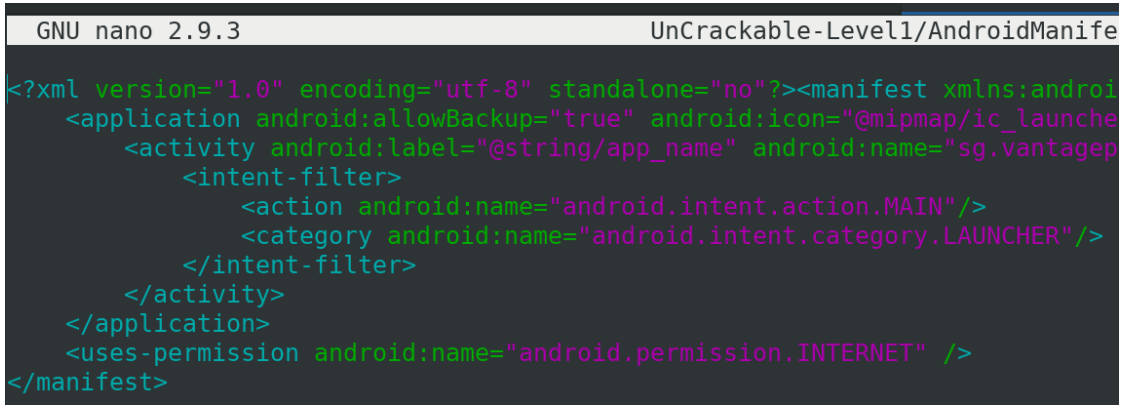
```
apktool d UnCrackable-Level1.apk
```

2. Edit the AndroidManifest.xml file

```
nano UnCrackable-Level1/AndroidManifest.xml
```

3. Add the INTERNET permission

```
<uses-permission android:name="android.permission.INTERNET" />
```



# UnCrackable Level 1 – Frida (lib-gadget)

4. Download the Frida gadget library
   - Check the architecture and Frida version

```
wget
https://github.com/frida/frida/releases/download/12.8.20/frida-
gadget-12.8.20-android-x86_64.so.xz

xz -d frida-gadget-12.8.20-android-x86_64.so.xz
```

5. Add the library inside the app

```
mkdir -p UnCrackable-Level1/lib/x86_64/

cp frida-gadget-12.8.20-android-x86_64.so UnCrackable-
Level1/lib/x86_64/libfrida-gadget.so
```

# UnCrackable Level 1 – Frida (lib-gadget)

6. Modify the smali code in order to load the Frida gadget library
   - Try to load the library very early
   - Usually, add the System.loadLibrary() call on the Main Activity

```
nano UnCrackable-
Level1/smali/sg/vantagepoint/uncrackable1/MainActivity.smali
```

   - Add the following lines inside the onCreate() function

```
const-string v0, "frida-gadget"
invoke-static {v0}, Ljava/lang/System;-
>loadLibrary(Ljava/lang/String;)V
```

# UnCrackable Level 1 – Frida (lib-gadget)

```
 GNU nano 2.9.3          UnCrackable-Level1/smali/sg/vantagepoint/uncrackable1/MainActivity.smali


    invoke-virtual {v0, v2, p1, v1}, Landroid/app/AlertDialog;->setButton(ILjava/lang/CharSequence;La

    const/4 p1, 0x0

    invoke-virtual {v0, p1}, Landroid/app/AlertDialog;->setCancelable(Z)V

    invoke-virtual {v0}, Landroid/app/AlertDialog;->show()V

    return-void
.end method


# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 1
    const-string v0, "frida-gadget"
    invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V

    invoke-static {}, Lsg/vantagepoint/a/c;->a()Z

    move-result v0

    if-nez v0, :cond_0

    invoke-static {}, Lsg/vantagepoint/a/c;->b()Z

    move-result v0
```

# UnCrackable Level 1 – Frida (lib-gadget)

7. Build the new APK with apktool

```
apktool b UnCrackable-Level1 -o gadgetUl1.apk
```

8. Sign the new APK with jarsigner

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -
keystore bud.keystore -storepass bsides gadgetUl1.apk
budkey
```

9. Remove the previous UnCrackable1

```
adb uninstall owasp.mstg.uncrackable1
```

10. Install the newly created APK

```
adb install gadgetUl1.apk
```

# UnCrackable Level 1 – Frida (lib-gadget)

- You can now launch the app
  - The app seems to be stuck
  - But it’s normal

- The Frida gadget is waiting for a connection

```
MobexlerLite ~ | owasp adb logcat | grep Frida
05-26 19:34:08.525  5127  5142 I Frida   : Listening on 127.0.0.1 TCP port 27042
05-26 19:36:41.532  5157  5172 I Frida   : Listening on 127.0.0.1 TCP port 27042
05-26 19:37:15.953  5191  5207 I Frida   : Listening on 127.0.0.1 TCP port 27042
05-26 19:38:13.614  5218  5233 I Frida   : Listening on 127.0.0.1 TCP port 27042
05-26 19:42:56.812  5380  5395 I Frida   : Listening on 127.0.0.1 TCP port 27042
```

- Note: If you don’t get this message on Logcat something went wrong!

# UnCrackable Level 1 – Frida (lib-gadget)

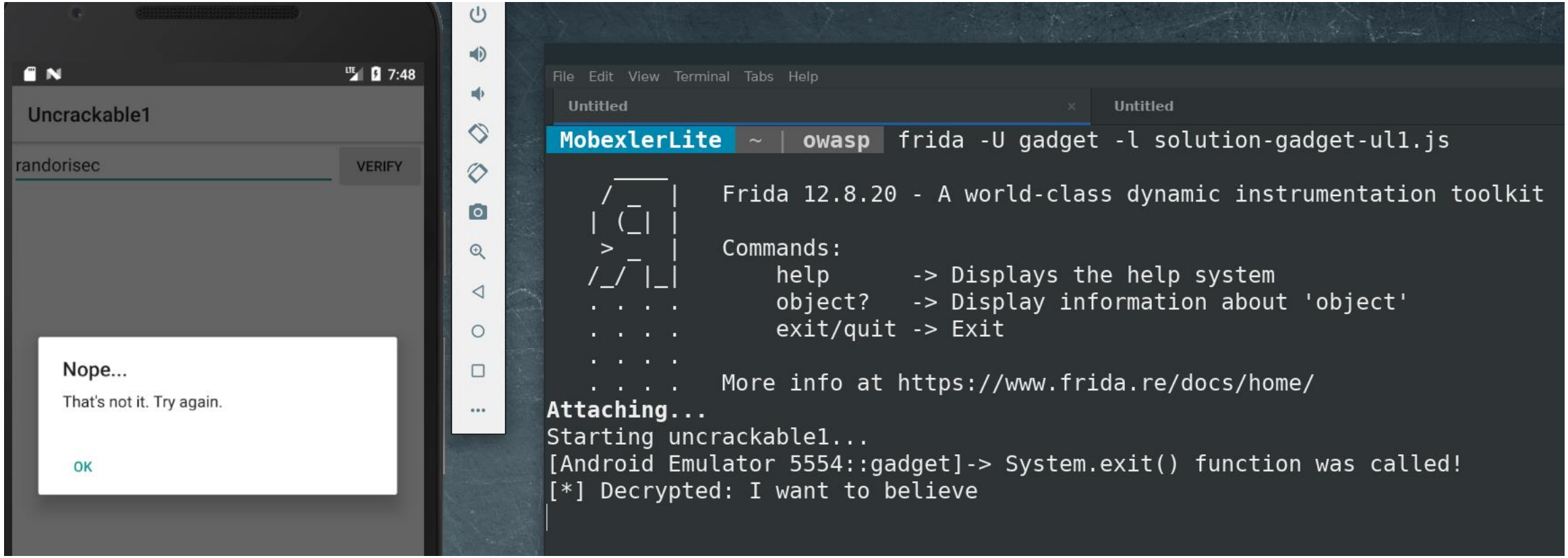
- Finally, just launch Frida using the following script

**solution-gadget-ul1.js**

```
Java.perform(function () {
  console.log("Starting uncrackable1...");
  var sysexit = Java.use("java.lang.System");
  sysexit.exit.implementation = function() {
  console.log("System.exit() function was called!"); };
  var aaClass = Java.use("sg.vantagepoint.a.a");
  aaClass.a.implementation = function(arg1, arg2) {
      var retval = this.a(arg1, arg2);
      var password = '';
      for(var i = 0; i < retval.length; i++) {
          password += String.fromCharCode(retval[i]);
      }
      console.log("[*] Decrypted: " + password);
      return retval; };
});
```

# UnCrackable Level 1 – Frida (lib-gadget)

```
frida -U gadget -l solution-gadget-ul1.js
```



# UnCrackable Level 1 – Frida (lib-gadget)

https://github.com/sensepost/objection/wiki/Patching-Android-Applications

- Tamper the app with objection
  1. Install the following packages needed by objection

```
sudo apt install aapt zipalign
```

  2. Execute objection by specifying the Frida version

```
objection patchapk --source UnCrackable-Level1.apk -V
12.8.20
```

- Note: If the emulator is not running, you need to specify the targeted architecture

```
objection patchapk --source UnCrackable-Level1.apk -V
12.8.20 --architecture x86_64
```

# UnCrackable Level 1 – Frida (lib-gadget)

DEMO

# UnCrackable Level 2

# UnCrackable Level 2 – Description

https://github.com/OWASP/owasp-mstg/tree/master/Crackmes#uncrackable-app-for-android-level-2

- Description
  - “This app holds a secret inside. May include traces of native code.”
  - Sensitive part of the code is inside a shared library (native code)
- This app is pretty similar to Level 1
  - The Root/Debug detection routine is the same
- However
  - The secret is hidden inside a shared library

# UnCrackable Level 2 – Analysis

- Looking at the code using jadx

```
jadx-gui Uncrackable-level2.apk
```

- A shared  library named “foo” is loaded
  - Class: **sg.vantagepoint.uncrackable2.MainActivity**
  - Function: **onCreate()**

```
16 public class MainActivity extends c {
17     private CodeCheck m;
18
19     static {
20         System.loadLibrary("foo");
21     }
22
```

# UnCrackable Level 2 – Analysis

- 2 native functions are defined inside the Java code

Class:
**sg.vantagepoint.uncrackable2.MainActivity**
Function:
**init()**

```
sg.vantagepoint.uncrackable2.MainActivity
36
37    private native void init();
38
39    /* access modifiers changed from: protected */
40    public void onCreate(Bundle bundle) {
41        init();
42        if (b.a() || b.b() || b.c()) {
43            a("Root detected!");
44        }
45        if (a.a(getApplicationContext())) {
46            a("App is debuggable!");
47        }
```

Class:
**sg.vantagepoint.uncrackable2.CodeCheck**
Function:
**bar()**

```
sg.vantagepoint.uncrackable2.CodeCheck
1 package sg.vantagepoint.uncrackable2;
2
3 public class CodeCheck {
4     private native boolean bar(byte[] bArr);
5
6     public boolean a(String str) {
7         return bar(str.getBytes());
8     }
9 }
```

# UnCrackable Level 2 – Analysis

- The bar() function is used to check the string provided by the user
  - The call is initiated from the verify() function from the Main Activity
    - Class: **sg.vantagepoint.uncrackable2.MainActivity**
    - Function: **verify()**

```
63          } execute(new Void[]{null, null, null});
64          this.m = new CodeCheck();
65          super.onCreate(bundle);
66          setContentView((int) R.layout.activity_main);
67      }
68
69      public void verify(View view) {
70          String str;
71          String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
72          AlertDialog create = new AlertDialog.Builder(this).create();
73          if (this.m.a(obj)) {
74              create.setTitle("Success!");
75              str = "This is the correct secret.";
76          } else {
```

sg.vantagepoint.uncrackable2.CodeCheck

```
1  package sg.vantagepoint.uncrackable2;
2
3  public class CodeCheck {
4      private native boolean bar(byte[] bArr);
5
6      public boolean a(String str) {
7          return bar(str.getBytes());
8      }
9  }
```

# UnCrackable Level 2 – Reverse

- It’s time to use apktool to decode the APK

```
apktool d Uncrackable-level2.apk
```

- And then analyse the “libfoo” library

```
MobexlerLite ~ | owasp find UnCrackable-Level2/lib/ -type f
UnCrackable-Level2/lib/armeabi-v7a/libfoo.so
UnCrackable-Level2/lib/x86/libfoo.so
UnCrackable-Level2/lib/x86_64/libfoo.so
UnCrackable-Level2/lib/arm64-v8a/libfoo.so
MobexlerLite ~ | owasp
```

- In this case, different architectures are available
  - For this workshop, we are going to focus on x86 ☺

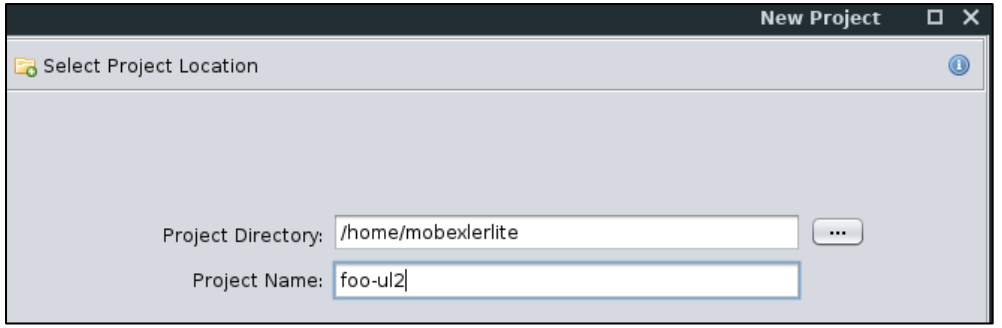
# UnCrackable Level 2 – Reverse

- We are going to use Ghidra to reverse the “foo” library

```
ghidraRun
```

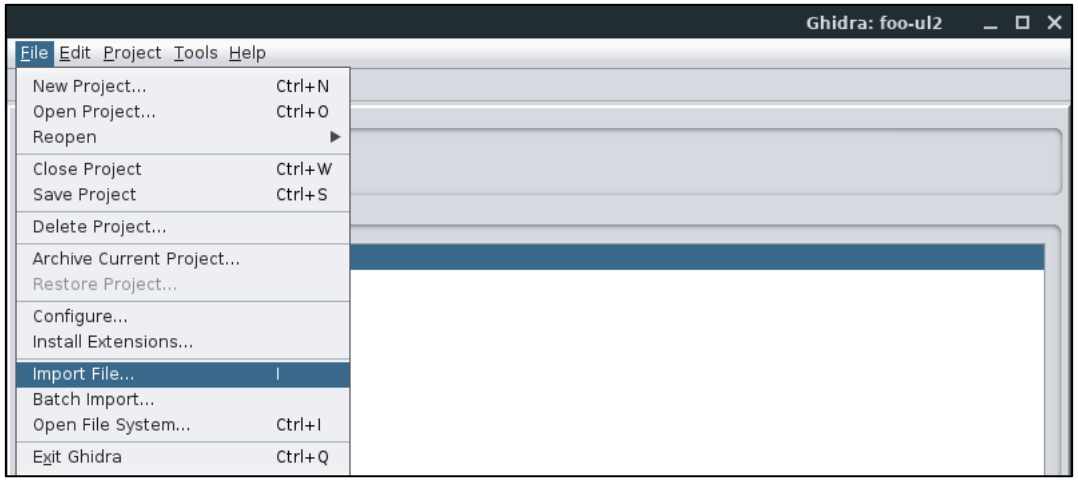
1. Start a “New Project…”


2. Select “Non-Shared Project”
3. Set the “Project Name”
   - **foo-ul2**
4. Finish

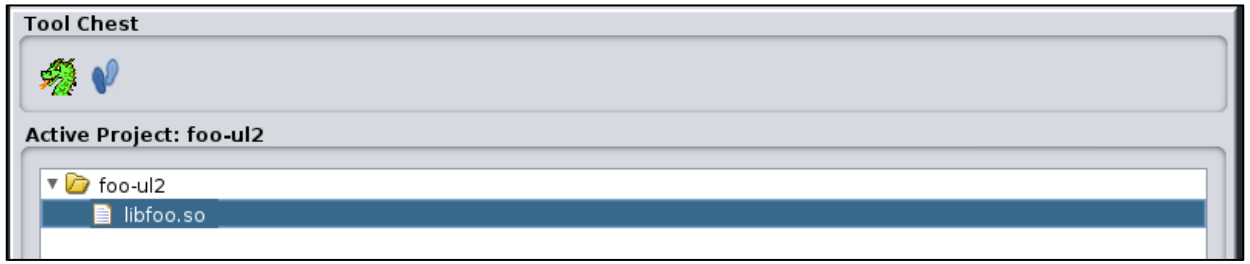


# UnCrackable Level 2 – Reverse

- Now, you can import the “libfoo” library”

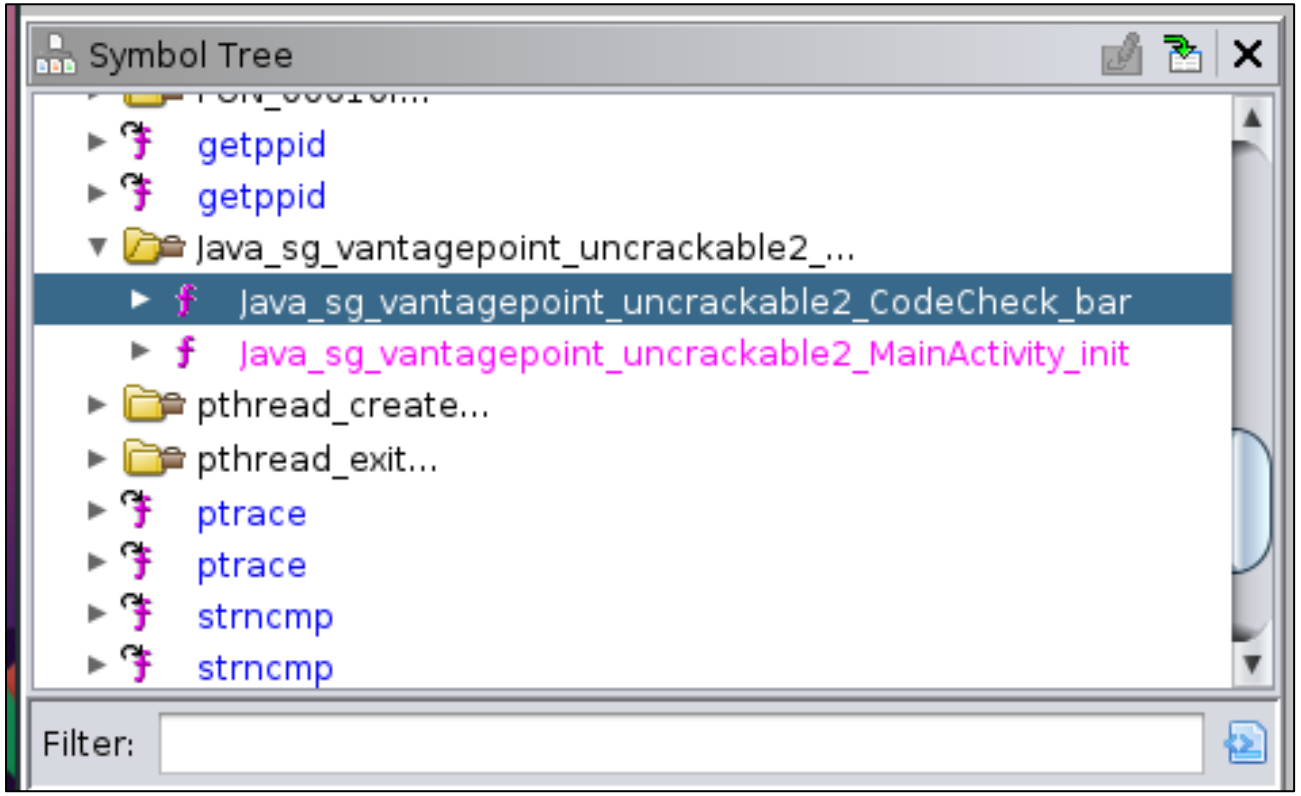



- Then, double-click on the “libfoo.so” file



# UnCrackable Level 2 – Reverse

- When looking at the functions provided by the “libfoo” library
  - 2 functions used by the APK are identified



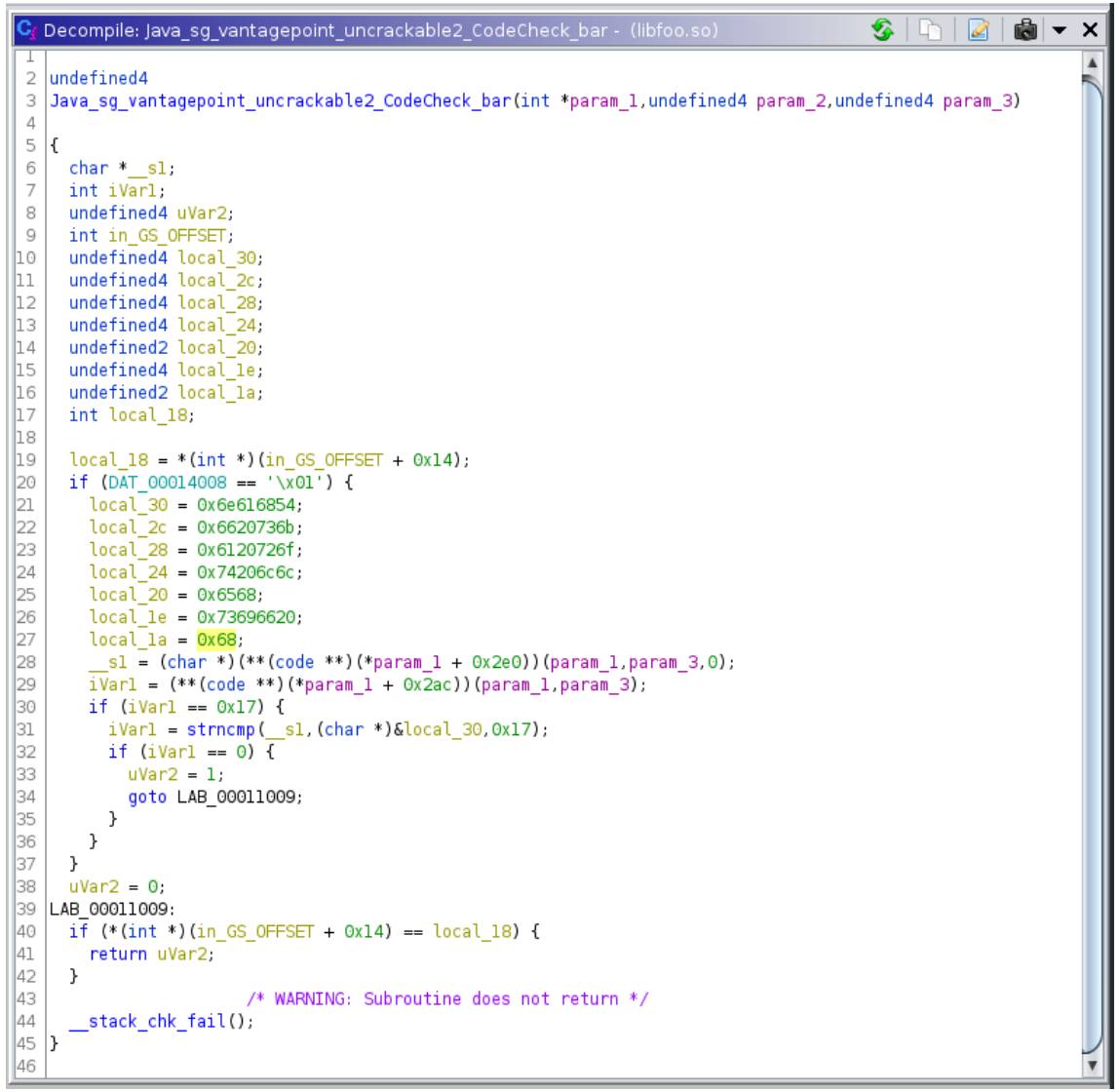
# UnCrackable Level 2 – Reverse

Analyze **Java_sg_vantagepoint_uncrackable2_CodeCheck_bar** function

- Using the decompiler provided by Ghidra

In this function, we can identify that

- Line 28: the byte array (param3) is casted in char * (__s1 variable)
- Line 30: the length of the secret should be 23 (0x17)
- Line 31: strncmp() function is used to check the user input against the hardcoded secret



# UnCrackable Level 2 – Frida (Native Code)

- Solution
  1. Bypass root detection using the same techniques used for UnCrackable Level 1
  2. Hook strncmp() function to display the arguments provided

- Frida allows to hook native code in the Android app
  - Interceptor (https://www.frida.re/docs/javascript-api/#interceptor )

# UnCrackable Level 2 – Frida (Native Code)

- **`Interceptor.attach(target, callbacks[,data])`**
  - Intercept calls and allows you to set callbacks when the function is called and when the function returns

```
Interceptor.attach (Module.findExportByName ( "libc.so", "read"), {
            onEnter: function (args) {
            send (args[1].readUtf8String ());
            },
            onLeave: function (retval) {
            }
});
```

# UnCrackable Level 2 – Frida (Native Code)

- **`Interceptor.replace(target, replacement[,data])`**
  - Replace the implementation of a function with **`replacement`**
  - Use **`NativeCallback`** to implement a **`replacement`** in JavaScript
  - NativeFunction allow you to call a native method

```
var openPtr = Module.getExportByName('libc.so', 'open');
var open = new NativeFunction(openPtr, 'int', ['pointer', 'int']);
Interceptor.replace(openPtr, new NativeCallback(function (pathPtr, flags) {
  var path = pathPtr.readUtf8String();
  log('Opening "' + path + '"');
  var fd = open(pathPtr, flags);
  log('Got fd: ' + fd);
  return fd;
}, 'int', ['pointer', 'int']));
```

# UnCrackable Level 2 – Frida (Native Code)

- In order to avoid displaying parameters for all strncmp() call
  - Implement a test to check if the 1st parameter starts with a specific pattern
    - For example: randorisec
- Respect the length of the expected user input
  - 23
- strncmp() is an external function provided by the libc library
  - `Module.findExportByName("libc.so", "strncmp")`

# UnCrackable Level 2 – Frida (Native Code)

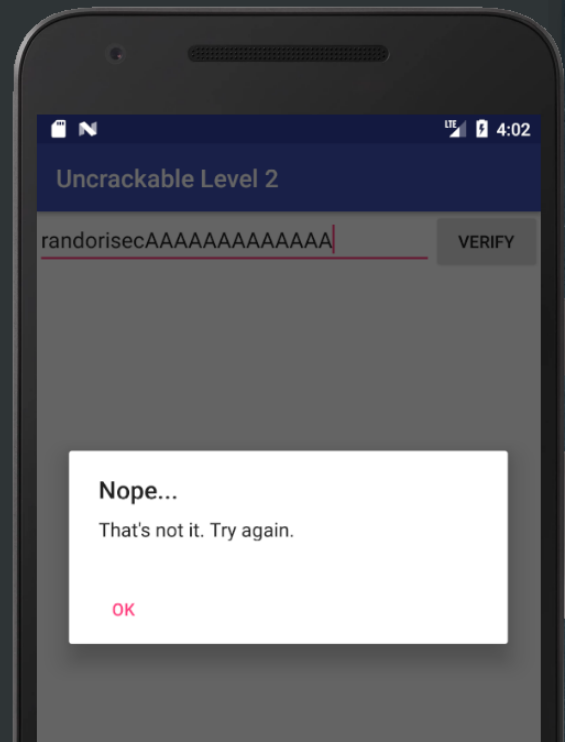
- Here is the final Frida script

```
Interceptor.attach (Module.findExportByName ( "libc.so", "strncmp"), {
    onEnter: function (args) {
        var param1 = Memory.readUtf8String(args[0]);
        var param2 = Memory.readUtf8String(args[1]);
        if (param1.startsWith("randorisec")) {
                console.log("Secret is: " +  param2);
        }
    } });
Java.perform(function () {
  console.log("Starting uncrackable2...");
  var sysexit = Java.use("java.lang.System");
  sysexit.exit.implementation = function() {
  console.log("System.exit() function was called!");
  }; });
```

# UnCrackable Level 2 – Frida (Native Code)

```
frida -U -l solution-ul2.js -f owasp.mstg.uncrackable2
--no-pause
```

```
MobexlerLite ~ | owasp frida -U -l solution-ul2.js -f owasp.mstg.uncrackable2 --no-pause
     ____
    / _  |   Frida 12.8.20 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable2`. Resuming main thread!
[Android Emulator 5554::owasp.mstg.uncrackable2]-> Starting uncrackable2...
System.exit() function was called!
Secret is: Thanks for all the fish
```



# UnCrackable Level 2

DEMO

# UnCrackable Level 3

# UnCrackable Level 3 – Description

https://github.com/OWASP/owasp-mstg/tree/master/Crackmes#uncrackable-app-for-android-level-3

- Description
  - “A secret string is hidden somewhere in this app. Find a way to extract it.”
  - Sensitive part of the code is inside a shared library (native code)
  - Additional anti-hooking and anti-tampering protections were implemented

- Same techniques from Level 2
  - The Root/Debug detection routine
  - The secret is hidden inside a shared library

# UnCrackable Level 3 – Anti-Tampering

- Checks are performed inside the Java code
  - Verify the CRC of all the libraries (/lib/ folder)
  - Verify the CRC of the classes.dex file
- If one of those files is modified
  - The variable “tampered” is set to 31337
  - The CRC values for the libraries are stored in the “strings.xml” file
  - The CRC value of the classes.dex is hardcoded inside the native code

```
43    <string name="app_name">Uncrackable Level 3</string>
44    <string name="arm64_v8a">1608485481</string>
45    <string name="armeabi">1054637268</string>
46    <string name="armeabi_v7a">881998371</string>
47    <string name="button_verify">Verify</string>
48    <string name="edit_text">Enter the Secret String</string>
49    <string name="foo_crc">1234</string>
50    <string name="mips">3104746423</string>
51    <string name="mips64">1319538057</string>
```

```
36 public class MainActivity extends AppCompatActivity {
       private static final String TAG = "UnCrackable3";
       static int tampered = 0;
       private static final String xorkey = "pizzapizzapizzapizzapizz";
       private CodeCheck check;
       Map<String, Long> crc;

       private native long baz();

       private native void init(byte[] bArr);
```

# UnCrackable Level 3 – Anti-Tampering

- Class: **sg.vantagepoint.uncrackable3.MainActivity**
- Function: **verifyLibs()**

```
54    private void verifyLibs() {
55        this.crc = new HashMap();
59        this.crc.put("armeabi-v7a", Long.valueOf(Long.parseLong(getResources().getString(R.string.armeabi_v7a))));
60        this.crc.put("arm64-v8a", Long.valueOf(Long.parseLong(getResources().getString(R.string.arm64_v8a))));
62        this.crc.put("x86", Long.valueOf(Long.parseLong(getResources().getString(R.string.x86))));
63        this.crc.put("x86_64", Long.valueOf(Long.parseLong(getResources().getString(R.string.x86_64))));
          try {
66            ZipFile zipFile = new ZipFile(getPackageCodePath());
              for (Map.Entry next : this.crc.entrySet()) {
                  String str = "lib/" + ((String) next.getKey()) + "/libfoo.so";
72                ZipEntry entry = zipFile.getEntry(str);
74                Log.v(TAG, "CRC[" + str + "] = " + entry.getCrc());
76                if (entry.getCrc() != ((Long) next.getValue()).longValue()) {
77                    tampered = 31337;
78                    Log.v(TAG, str + ": Invalid checksum = " + entry.getCrc() + ", supposed to be " + next.getValue());
                  }
              }
85            ZipEntry entry2 = zipFile.getEntry("classes.dex");
87            Log.v(TAG, "CRC[" + "classes.dex" + "] = " + entry2.getCrc());
89            if (entry2.getCrc() != baz()) {
90                tampered = 31337;
91                Log.v(TAG, "classes.dex" + ": crc = " + entry2.getCrc() + ", supposed to be " + baz());
              }
          } catch (IOException unused) {
```

# UnCrackable Level 3 – Anti-Hooking

- Checks are performed inside the Native code
  - If Frida or Xposed are detected, the app craches!

```
05-27 20:59:09.593  1360  1413 W audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
05-27 20:59:11.030  1360  1414 W audio_hw_generic: Not supplying enough data to HAL, expected position 17955544 , only wrote 17601120
05-27 20:59:24.318  6820  6844 V UnCrackable3: Tampering detected! Terminating...
05-27 20:59:24.318  6820  6844 F libc    : Fatal signal 6 (SIGABRT), code -6 in tid 6844 (tg.uncrackable3)
05-27 20:59:24.320  1255  1255 W         : debuggerd: handling request: pid=6820 uid=10083 gid=10083 tid=6844
05-27 20:59:24.322  6852  6852 E         : debuggerd: ptrace attach to 6820 failed: Operation not permitted
05-27 20:59:24.336  6852  6852 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
05-27 20:59:24.336  6852  6852 F DEBUG   : Build fingerprint: 'Android/sdk_google_phone_x86_64/generic_x86_64:7.0/NYC/6400688:userdebug/dev-keys'
05-27 20:59:24.336  6852  6852 F DEBUG   : Revision: '0'
05-27 20:59:24.336  6852  6852 F DEBUG   : ABI: 'x86_64'
05-27 20:59:24.336  6852  6852 F DEBUG   : pid: 6820, tid: 6844, name: tg.uncrackable3  >>> owasp.mstg.uncrackable3 <<<
05-27 20:59:24.336  6852  6852 F DEBUG   : signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr --------
05-27 20:59:24.336  6852  6852 F DEBUG   :     rax 0000000000000000  rbx 00007c3bf2adee30  rcx ffffffffffffffff  rdx 0000000000000006
05-27 20:59:24.336  6852  6852 F DEBUG   :     rsi 0000000000001abc  rdi 0000000000001aa4
05-27 20:59:24.336  6852  6852 F DEBUG   :     r8  00007c3bf263e090  r9  00007c3bf2adeb80  r10 00007c3bf2adee30  r11 0000000000000246
05-27 20:59:24.336  6852  6852 F DEBUG   :     r12 0000000000001abc  r13 0000000000000006  r14 00007c3bf2adee30  r15 00007c3bf2adeb00
05-27 20:59:24.336  6852  6852 F DEBUG   :     cs  0000000000000033  ss  000000000000002b
05-27 20:59:24.336  6852  6852 F DEBUG   :     rip 00007c3c12f833e8  rbp 0000000000000000  rsp 00007c3bf2adead8  eflags 0000000000000246
05-27 20:59:24.339  6852  6852 F DEBUG   :
05-27 20:59:24.339  6852  6852 F DEBUG   : backtrace:
05-27 20:59:24.339  6852  6852 F DEBUG   :     #00 pc 000000000007a3e8  /system/bin/linker64 (__dl_syscall+24)
05-27 20:59:24.339  6852  6852 F DEBUG   :     #01 pc 000000000000c03b  /system/bin/linker64 (__dl__ZL24debuggerd_signal_handleriP7siginfoPv+987)
05-27 20:59:24.339  6852  6852 F DEBUG   :     #02 pc 000000000020d641  /data/local/tmp/re.frida.server/frida-agent-64.so
05-27 20:59:24.339  6852  6852 F DEBUG   :     #03 pc 0000000000001a8f  /system/lib64/libc.so (offset 0x1b000)
05-27 20:59:24.545  6820  6844 F libc    : Fatal signal 6 (SIGABRT), code -6 in tid 6844 (tg.uncrackable3)
05-27 20:59:24.545  6820  6844 I libc    : Another thread contacted debuggerd first; not contacting debuggerd.
```

# UnCrackable Level 3 – Anti-Hooking

- Using Ghidra we can identify the function performing the check
  - x86 libfoo.so analyzed
- The strstr() function is used to search the strings “frida” or “xposed” inside /proc/self/maps
  - If this is the case the “goodbye” function is called
  - As the name implies, the app stops

Decompile: FUN_00013080 - (libfoo.so)

```
1
2  void FUN_00013080(void)
3
4  {
5    FILE *__stream;
6    char *pcVar1;
7    FILE *pFVar2;
8    int in_GS_OFFSET;
9    int iVar3;
10   undefined *puStack564;
11   char *pcStack548;
12   char *pcStack544;
13   char *pcStack540;
14   char *pcStack536;
15   char acStack532 [516];
16
17   puStack564 = &LAB_00013094;
18   pcStack544 = "r";
19   pcStack548 = "/proc/self/maps";
20   __stream = fopen("/proc/self/maps","r");
21   if (__stream != (FILE *)0x0) {
22     pcStack536 = "frida";
23     pcStack540 = "xposed";
24     do {
25       pcVar1 = fgets(acStack532,0x200,__stream);
26       if (pcVar1 == (char *)0x0) {
27         fclose(__stream);
28         usleep(500);
29         __stream = fopen(pcStack548,pcStack544);
30         pFVar2 = __stream;
31       }
32       else {
33         pcVar1 = strstr(acStack532,pcStack536);
34         if (pcVar1 != (char *)0x0) break;
35         pFVar2 = (FILE *)strstr(acStack532,pcStack540);
36       }
37     } while (pFVar2 == (FILE *)0x0);
38   }
39   __android_log_print();
40   puStack564 = (undefined *)0x1317a;
41   goodbye();
42   iVar3 = *(int *)(in_GS_OFFSET + 0x14);
```

# UnCrackable Level 3 – Hidden Secret

- To understand how the hidden secret is hardcoded let's have a look to the function
  - **Java_sg_vantagepoint_uncrackable3_CodeCheck_bar**

- 2 variables are XORed and compared to the user input
  - iVar1 is the user input
  - puVar5 is a secret from the Java code
  - local40 is obtained by calling
    - FUN_00010fa0

Decompile: Java_sg_vantagepoint_uncrackable3_CodeCheck_bar - (libfoo.so)

```
undefined4
Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(int *param_1,undefined4 param_2,undefined4 param_3)

{
  int iVar1;
  int iVar2;
  uint uVar3;
  undefined4 uVar4;
  undefined4 *puVar5;
  int in_GS_OFFSET;
  undefined4 local_40;
  undefined4 uStack60;
  undefined4 uStack56;
  undefined4 uStack52;
  undefined4 local_30;
  undefined4 local_2c;
  undefined local_28;
  int local_18;

  local_18 = *(int *)(in_GS_OFFSET + 0x14);
  local_40 = 0;
  uStack60 = 0;
  uStack56 = 0;
  uStack52 = 0;
  local_2c = 0;
  local_30 = 0;
  local_28 = 0;
  if (DAT_0001c020 == 0) {
    FUN_00010fa0(&local_40);
    iVar1 = (**(code **)(*param_1 + 0x2e0))(param_1,param_3,0);
    iVar2 = (**(code **)(*param_1 + 0x2ac))(param_1,param_3);
    if (iVar2 == 0x18) {
      uVar3 = 0;
      puVar5 = &DAT_0001601c;
      do {
        if (*(byte *)(iVar1 + uVar3) != (*(byte *)puVar5 ^ *(byte *)((int)&local_40 + uVar3)))
          goto LAB_00013456;
        uVar3 = uVar3 + 1;
        puVar5 = (undefined4 *)((int)puVar5 + 1);
      } while (uVar3 < 0x18);
      uVar4 = CONCAT31((int3)(uVar3 >> 8),1);
      if (uVar3 == 0x18) goto LAB_00013458;
    }
  }
LAB_00013456:
  uVar4 = 0;
LAB_00013458:
  if (*(int *)(in_GS_OFFSET + 0x14) == local_18) {
    return uVar4;
  }
```

# UnCrackable Level 3 – Hidden Secret

- The secret code from the Java code is
  - pizzapizzapizzapizzapizz
- Then, the init() function is called with this string
- Finally, this string is copied in an internal variable inside the native code
  - DAT_0001601c
- Let's call it the "xorkey"

```
36 public class MainActivity extends AppCompatActivity {
       private static final String TAG = "UnCrackable3";
       static int tampered = 0;
       private static final String xorkey = "pizzapizzapizzapizzapizz";
       private CodeCheck check;
       Map<String, Long> crc;

       private native long baz();

       private native void init(byte[] bArr);
```

```
sg.vantagepoint.uncrackable3.MainActivity
       /* access modifiers changed from: protected */
103    public void onCreate(Bundle bundle) {
104        verifyLibs();
106        init(xorkey.getBytes());
```

```
Decompile: Java_sg_vantagepoint_uncrackable3_MainActivity_init - (libfoo.so)
1
2  void Java_sg_vantagepoint_uncrackable3_MainActivity_init
3                 (int *param_1,undefined4 param_2,undefined4 param_3)
4
5  {
6    char *__src;
7
8    FUN_00013250();
9    __src = (char *)(**(code **)(*param_1 + 0x2e0))(param_1,param_3,0);
10   strncpy((char *)&DAT_0001601c,__src,0x18);
11   (**(code **)(*param_1 + 0x2e0))(param_1,param_3,__src,2);
12   DAT_00016038 = DAT_00016038 + 1;
13   return;
14 }
15
```

# UnCrackable Level 3 – Hidden Secret

- The second part of the secret is provided by another function
- At the end of the function, the parameter contains the second part of the secret
- Let's call it "secret"

# UnCrackable Level 3 – Hidden Secret

- To sum up, the following pseudo-code explains how the **bar()** function test the user input

```
function Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(JNIenv, jobject,
user_input) {
    iVar1 = user_input;
    iVar2 = length(user_input);
    FUN_NONAME(&secret); // Need to retrieve this secret
    if (iVar2 == 24){
        xorkey = &DAT_0001601c; //We already know the xorkey
        for(i=0; i<24; i++) {
            if(iVar1[i] != (secret[i] ^xorkey[i]) break;
        }
    }
}
```

# UnCrackable Level 3 – Solution

- The solution will be provided only using Frida!
  1. Hook the **System.exit()** function in order to avoid to exit the app (same technique as level 1 and 2)
  2. Hook the **strstr()** function in order to return false when Frida is detected
  3. Hook the unnamed function setting the “secret”
  4. We already have the “xorkey” inside the Java code (JADX)
  5. Finally, using a Python script, we are xoring the “secret” and the “xorkey” to obtain the hidden secret

- Bypass root and Frida detection
  - strstr() is from libc
  - If one of the parameters is “frida”, the return value is modified to return false

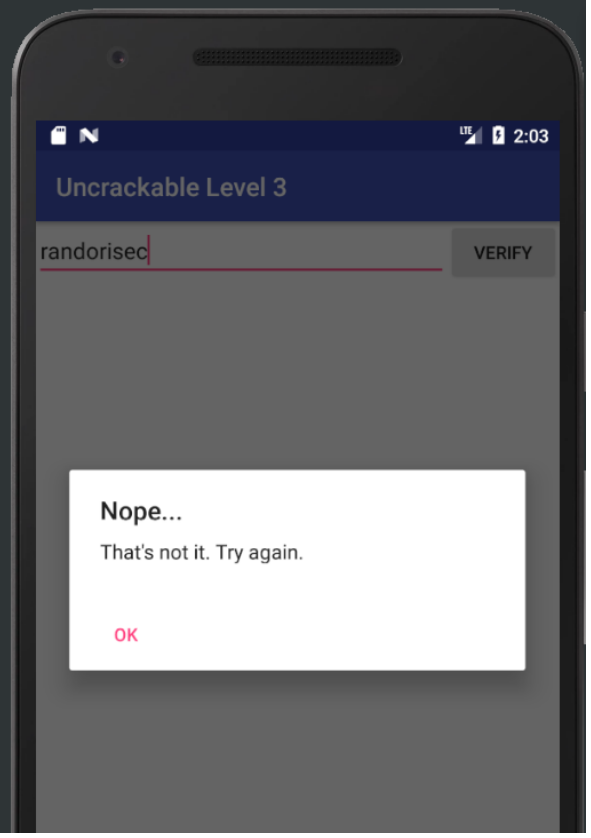
**bypass-anti-frida-ul3.js**

```
Java.perform(function () {
  console.log("Starting uncrackable3...");
  var root_class = Java.use("java.lang.System");
  root_class.exit.implementation = function() {
    console.log("System.exit() function was called!");
  }; });
Interceptor.attach(Module.findExportByName("libc.so", "strstr"),
{
    onEnter: function (args) {
        this.frida = Boolean(0);
       var haystack = Memory.readUtf8String(args[0]);
       var needle   = Memory.readUtf8String(args[1]);
    if ( haystack.indexOf("frida") != -1) {
            this.frida = Boolean(1);
     }
    },
    onLeave: function (retval) {
        if (this.frida) { retval.replace(0);}
        return retval;
    } });
```

# UnCrackable Level 3 – Solution

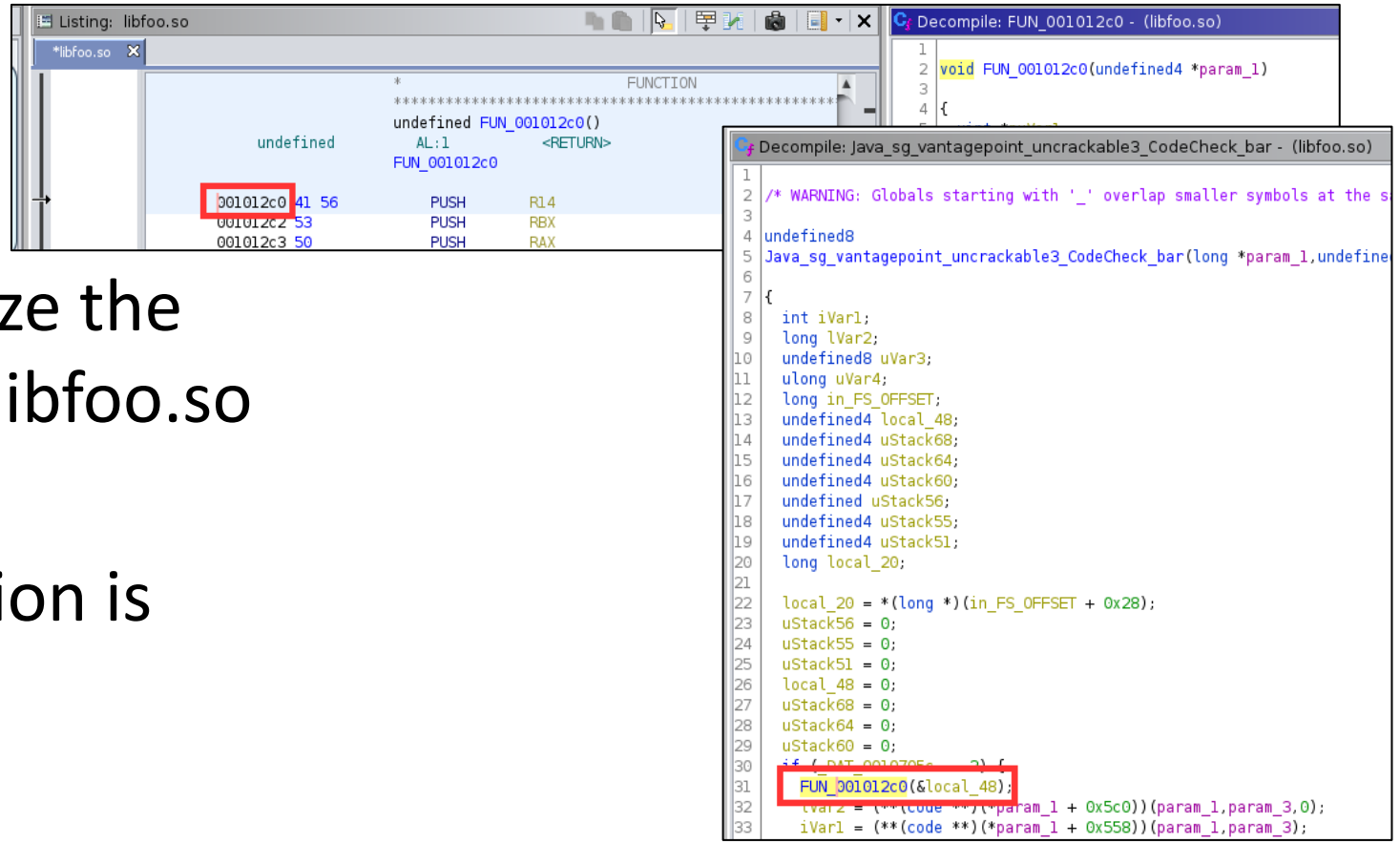
```
frida -U -l bypass-anti-frida-ul3.js -f
owasp.mstg.uncrackable3 --no-pause
```

```
MobexlerLite ~ | owasp frida -U -l bypass-anti-frida-ul3.js -f owasp.mstg.uncrackable3 --no-pause
     ____
    / _  |   Frida 12.8.20 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable3`. Resuming main thread!
[Android Emulator 5554::owasp.mstg.uncrackable3]-> Starting uncrackable3...
[Android Emulator 5554::owasp.mstg.uncrackable3]->
[Android Emulator 5554::owasp.mstg.uncrackable3]-> System.exit() function was called!
[Android Emulator 5554::owasp.mstg.uncrackable3]->
[Android Emulator 5554::owasp.mstg.uncrackable3]->
```



# UnCrackable Level 3 – Solution

- In order to hook the unnamed, we need to retrieve its address using Ghidra




  - Here we need to analyze the x86_64 version of the libfoo.so
  - Our device is x86_64
  - The offset of the function is
    - 0x000012c0

# UnCrackable Level 3 – Solution

- In this example
  - We retrieve the correct pointer for the function
  - Then, we attach to it
- onEnter
  - The pointer to the argument is saved
- onLeave
  - The content of the pointer is displayed

```
function hook_native_libs() {
  var offset_fun = 0x000012c0;
  var p_foo = Module.findBaseAddress('libfoo.so');
  var p_fun_secret = p_foo.add(offset_fun);

  Interceptor.attach( p_fun_secret, {
    onEnter: function (args) {
      this.secret = args[0];
      console.log("onEnter() p_fun_secret");
    },
    onLeave: function (retval) {
      console.log("onLeave() p_fun_secret");
      console.log(Memory.readByteArray(
                    this.secret, 24));
    }
  });
}
```

# UnCrackable Level 3 – Solution

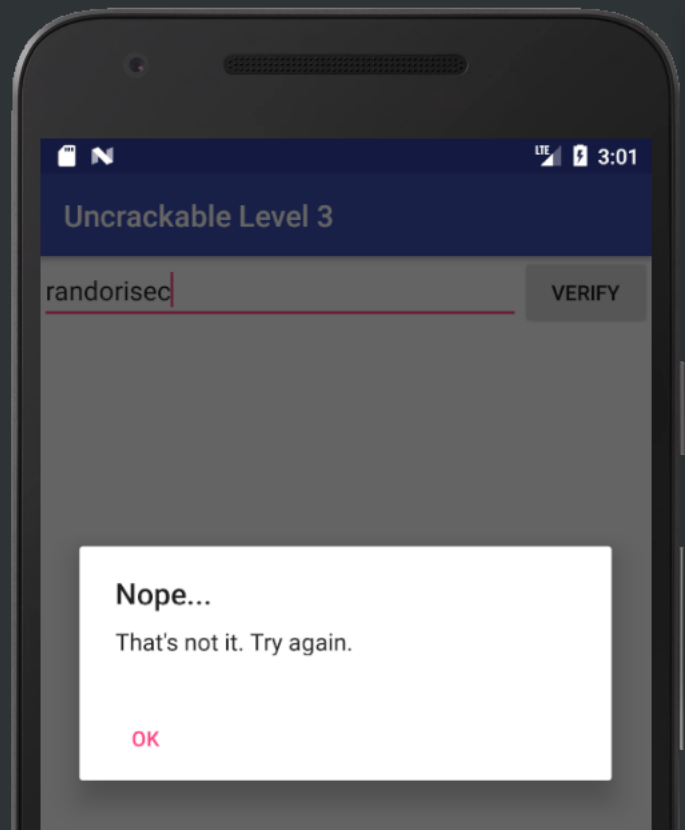
- In order to attach to the libfoo.so library, we need to wait the app to load the library
  - A small trick is to hook the onStart() function from the MainActivity
  - After that, we can try to attach to the libfoo.so

```
Java.perform(function () {
  console.log("Starting uncrackable3...");
  var root_class = Java.use("java.lang.System");
  root_class.exit.implementation = function() {
    console.log("System.exit() function was
called!");  };
  // this is just a placeholder to be sure the
libfoo.so was correctly loaded
  var mainactivity =
Java.use("sg.vantagepoint.uncrackable3.MainActivity
");
  mainactivity.onStart.overload().implementation =
function() {
     console.log("MainActivity was called!!!");
     var ret = this.onStart.overload().call(this);
  };
  // Now, we can attach to libfoo.so
  hook_native_libs(); });
```

# UnCrackable Level 3 – Solution

```
frida -U -l solution-ul3.js -f owasp.mstg.uncrackable3
--no-pause
```

```
MobexlerLite ~ | owasp frida -U -l solution-ul3.js -f owasp.mstg.uncrackable3 --no-pause
     ____
    / _  |   Frida 12.8.20 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable3`. Resuming main thread!
[Android Emulator 5554::owasp.mstg.uncrackable3]-> Starting uncrackable3...
MainActivity was called!!!
System.exit() function was called!
onEnter() p_fun_secret
onLeave() p_fun_secret
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  1d 08 11 13 0f 17 49 15 0d 00 03 19 5a 1d 13 15  ......I.....Z...
00000010  08 0e 5a 00 17 08 13 14                          ..Z.....
```



# UnCrackable Level 3 – Solution

- So now, we can just perform the xor operation against the
  - xorkey = “pizzapizzapizzapizzapizz”
  - secret = “1d0811130f1749150d0003195a1d1315080e5a0017081314”

```
MobexlerLite ~ | owasp cat xor-secret.py
#!/usr/bin/python3

secret = "1d0811130f1749150d0003195a1d1315080e5a0017081314".decode("hex")
xorkey = "pizzapizzapizzapizzapizz"

def xor_strings(xs, ys):
    return "".join(chr(ord(x) ^ ord(y)) for x, y in zip(xs, ys))

user_input = xor_strings(secret,xorkey)
print("The secret is: {0}".format(user_input))
MobexlerLite ~ | owasp python xor-secret.py
The secret is: making owasp great again
MobexlerLite ~ | owasp
```

# UnCrackable Level 3

DEMO

# Questions

# References

# References – General

- OWASP MSTG
  - https://github.com/OWASP/owasp-mstg/
- Android Applications Reversing 101
  - https://www.evilsocket.net/2017/04/27/Android-Applications-Reversing-101/
- Japan Smartphone Security Association (JSSEC) - Secure Coding Guide
  - https://www.jssec.org/dl/android_securecoding_en_20180901.pdf
- Mobile Hacking Cheat Sheet
  - https://github.com/randorisec/MobileHackingCheatSheet

# References – Frida

- Frida Javascript API
  - https://www.frida.re/docs/javascript-api
- Frida CodeShare
  - https://codeshare.frida.re
- Frida Snippets
  - https://github.com/iddoeldor/frida-snippets

# References – Frida

- Basic Dynamic Analysis With Frida
  - https://similarweb.engineering/basic-dynamic-analysis-with-frida/
- Frida Cheat Sheet
  - https://awakened1712.github.io/hacking/hacking-frida/
- Instrumenting Native Android Functions using Frida
  - https://www.notsosecure.com/instrumenting-native-android-functions-using-frida/
- Frida without root (or using lib-gadget)
  - https://koz.io/using-frida-on-android-without-root/

# References – Crackme Solutions

- UnCrackable Level 1
  - https://enovella.github.io/android/reverse/2017/05/18/android-owasp-crackmes-level-1.html
- UnCrackable Level 2
  - https://enovella.github.io/android/reverse/2017/05/20/android-owasp-crackmes-level-2.html
- UnCrackable Level 3
  - https://enovella.github.io/android/reverse/2017/05/20/android-owasp-crackmes-level-3.html