Android Mobile Hacking Workshop

March 27th, 2021

RANDORISEC

Objectives

Objectives

- Introduce common tools to assess Android apps
  - `apktool`
  - `jadx`
  - `Objection`
  - `Frida`
  - `Ghidra`
- Present different techniques to analyze an Android app
- Perform practical exercices

Targets

- Examples based on OWASP Crackmes (available on the Github repository)
  - Android UnCrackable Level 1
  - Android UnCrackable Level 2
  - Android UnCrackable Level 3

Prerequisites

Mobexler

- For this workshop, we are going to use Mobexler Virtual Machine
  - Download link: https://mobexler.com/download.htm
  - Credentials: **mobexler/12345**

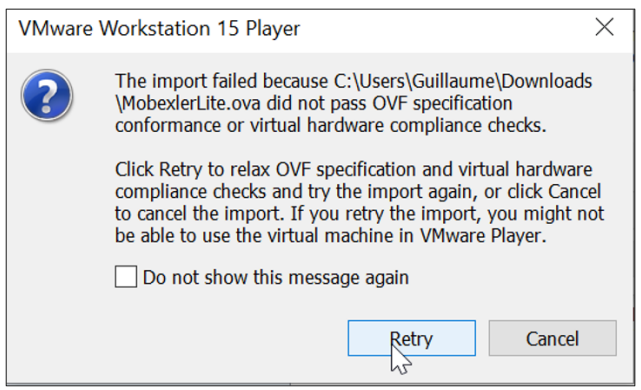
- Mobexler is a *Mobile Application Penetration Testing Platform*
  - Similar to a Kali Linux
  - Focused on mobiles apps: Android and iOS

- Setup Guide
  - https://www.randorisec.fr/setting-up-mobexler-vmware-android-studio/

Mobexler - VMWare Player

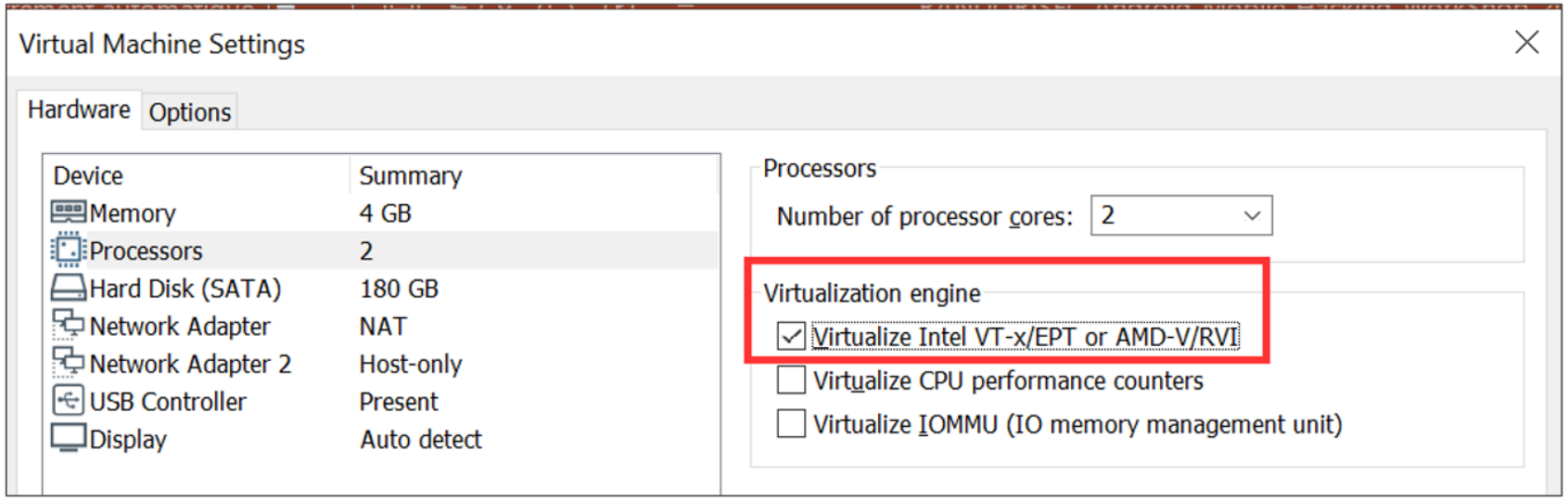
- **It is mandatory to install and use [VMWare Workstation Player]**
  - VMWare is needed in order to use the Android emulator

- First, 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 :)



Mobexler - VMWare Hardware Settings

- Then, edit the Virtual Machine hardware settings
  - Memory: At least 4 GB
  - Processors: Enable "Virtualize Intel VT-x/EPT or AMD-V/RVI"



Mobexler - Configuration

- Boot the Mobexler virtual machine
  - Install the VMWare tools (optional but recommended)

```
sudo apt install open-vm-tools-desktop
```

  - Enable KVM permissions for Android Studio

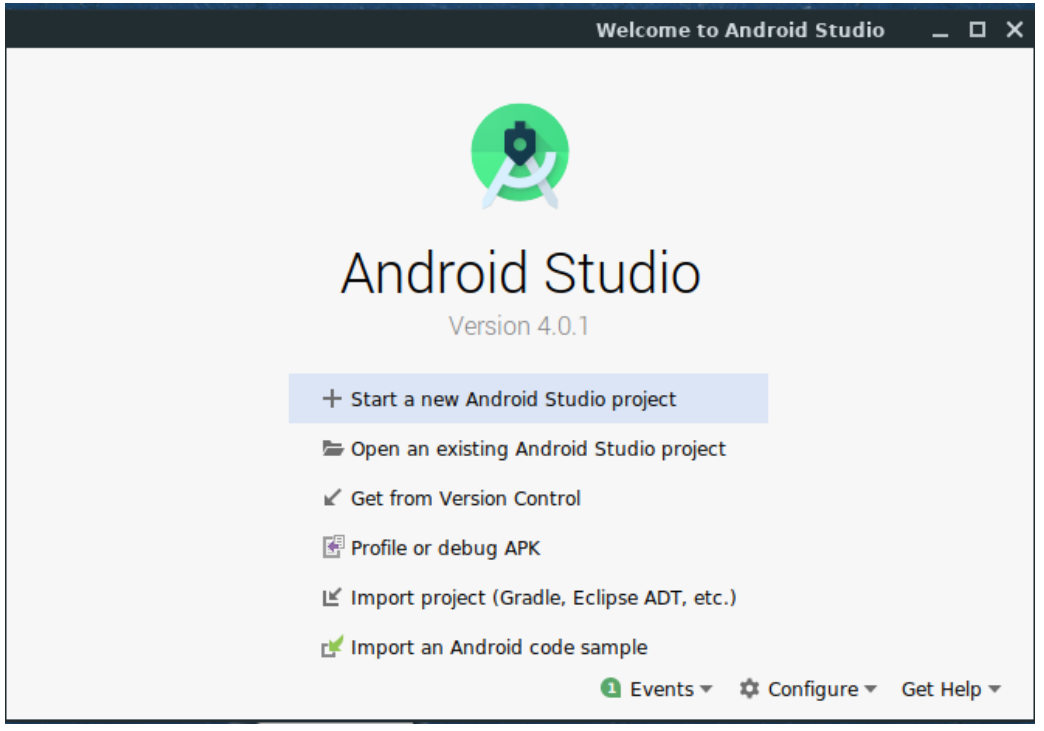
```
sudo apt update
sudo apt install qemu-kvm
sudo apt adduser Mobexler kvm
reboot
```

Emulator

- Free versions
  - Android Virtual Device (AVD): Official android emulator
  - Android x86: An x86 port of the Android code base
- Commercial versions
  - Genymotion: Mature emulator with many features, both as local and cloud-based solution (free version available for non-commercial use)
  - Corellium: Offers custom device virtualization through a cloud-based or on-prem solution

Android Studio

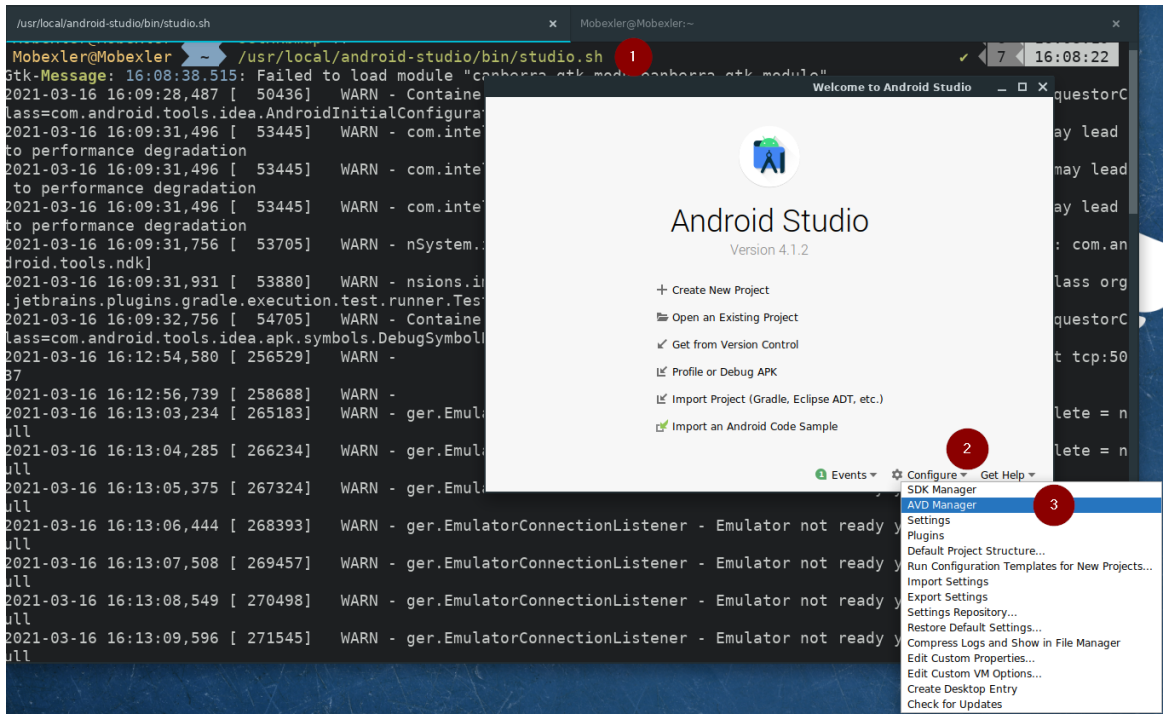
- Default IDE
  - Create Android apps
  - Debug apps
  - Logcat
  - Create/Manage emulators



Android Studio - AVD

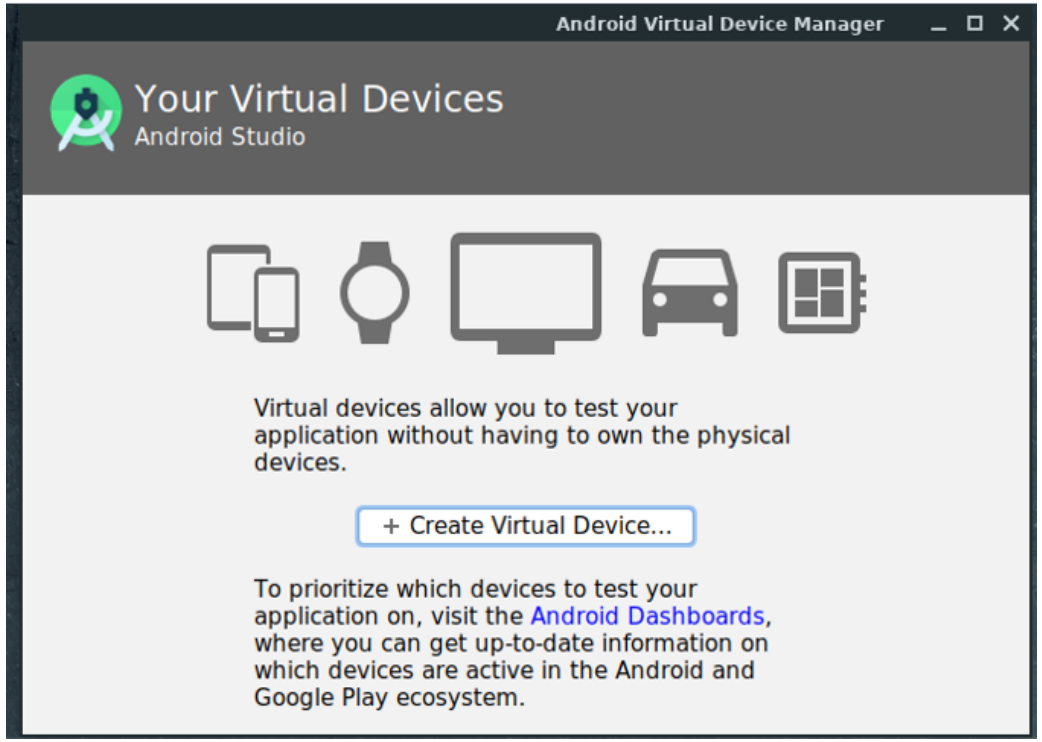
- Let's create an Android Virtual Device (AVD) by launching Android Studio

```
/usr/local/android-studio/bin/studio.sh
```

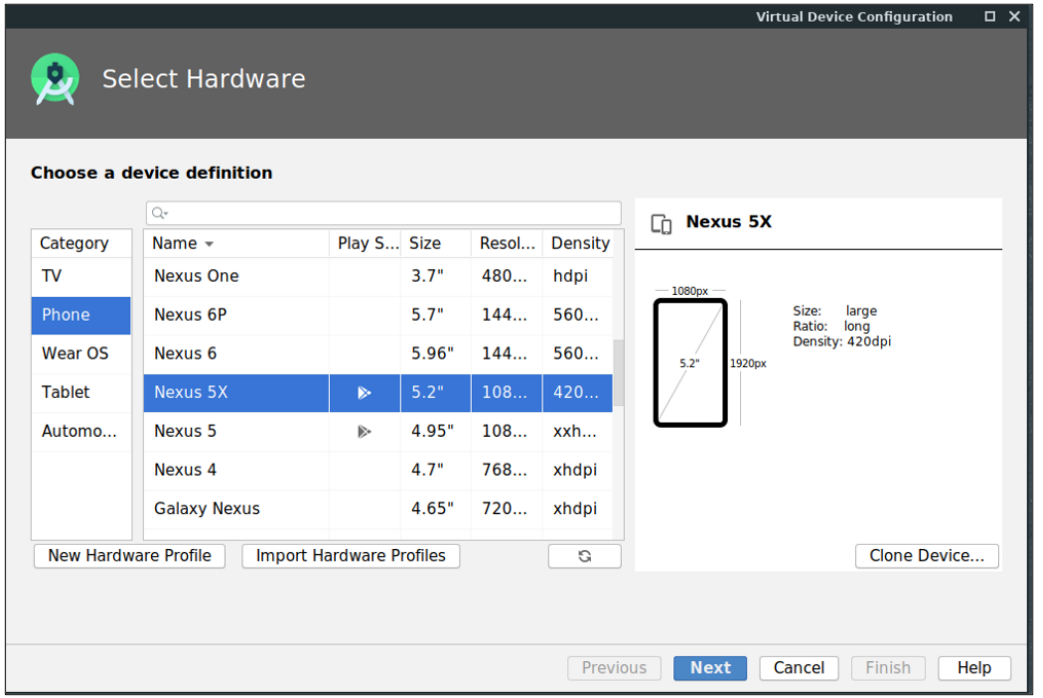


Android Studio - Virtul Device

- Create a Virtual Device

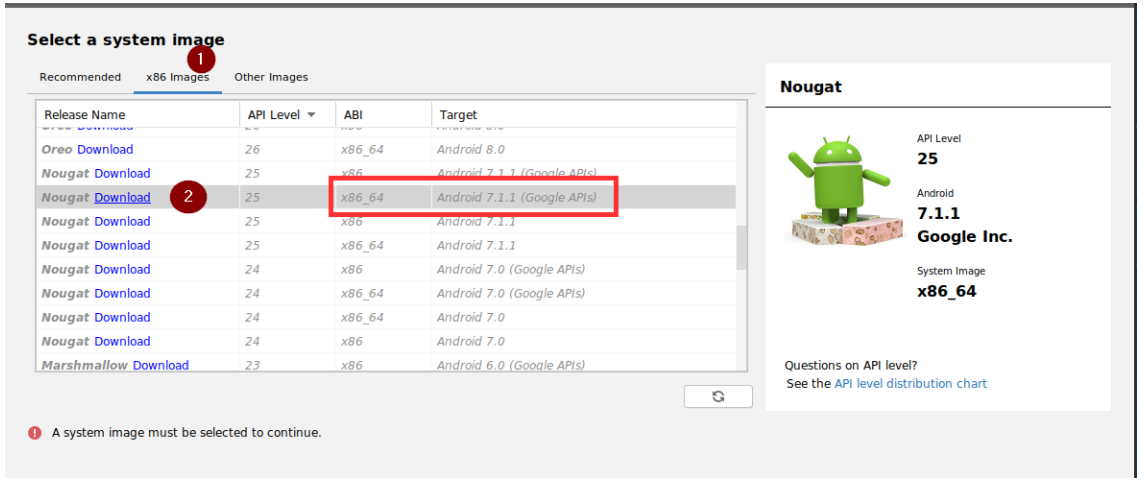


- Select Nexus 5X



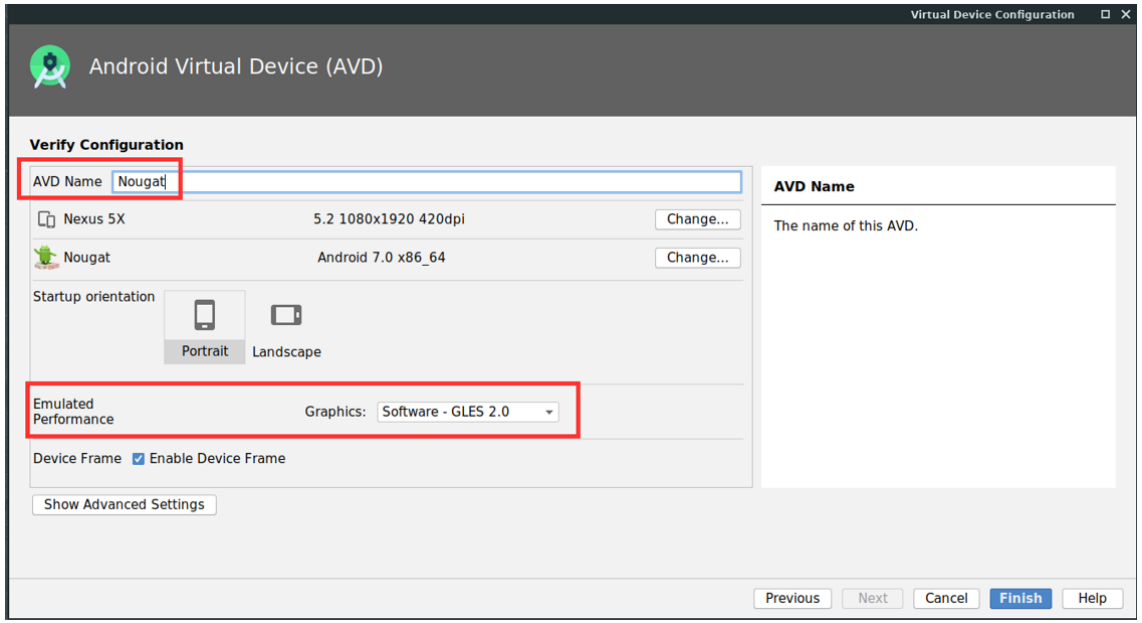
Android Studio - System Image

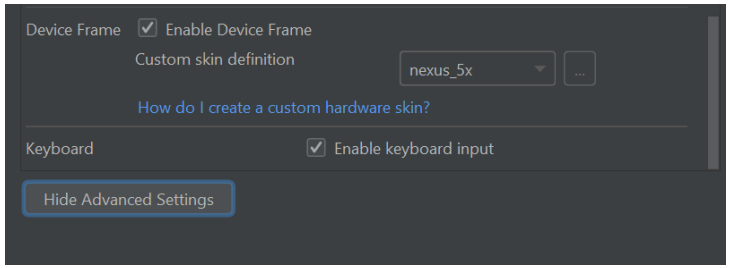
- Select an x86 system image
  - Release Name: **Nougat**
  - API Level: **25**
  - ABI: **x86_64**
  - Target: **Android 7.1.1 (Google APIs)**
- Download the system image and then click Next
- Select **Google APIs images** to have root privileges



Android Studio - Configuration

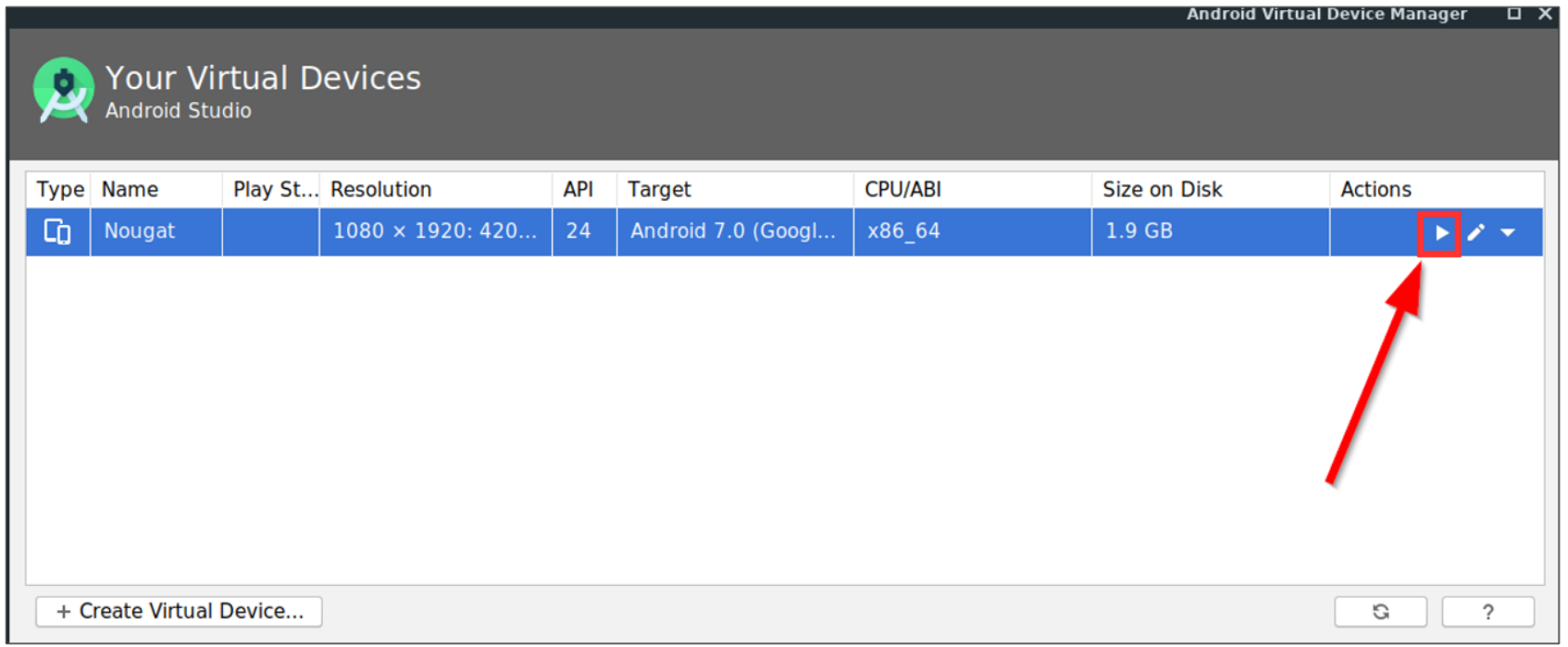
- Set-up the AVD Name
  - `Nougat`
- Modify the Graphics performance
  - `Software - GLES 2.0`
- Modify the `Advanced Settings`
  - `Enable keyboard input`





Android Studio

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



Github

- Please clone the following repository

```
git clone https://github.com/randorisec/BSidesDublin2021-Workshops
```

- Go to the Android subdirectory

```
cd BSidesDublin2021-Workshops/Android
```

- And you should have the following

```
# ls
README.md  ul1  ul2  ul3
```

OWASP Crackmes Installation

- If the emulator is not running, launch it!
- Install each OWASP Crackme using `adb`

```
# UnCrackable Level 1
adb install ul1/UnCrackable-Level1.apk

# UnCrackable Level 2
adb install ul2/UnCrackable-Level2.apk

# UnCrackable Level 2
adb install ul3/UnCrackable-Level3.apk
```

adb

- Command line tool to interact with an Android device
  - Get a shell

```
adb shell
```

  - Run a command

```
adb shell [cmd]
```

  - Restart adb with root privileges

```
adb root
```

  - List Android devices connected

```
adb devices
```

  - Copy local file to device

```
adb push [local] [device]
```

  - Copy file from remote device

```
adb pull [remote] [local]
```

adb - Package Manager

- List apps installed on your device (`-f` option displays the APK path)

```
adb shell pm list packages -f

adb shell cmd package list packages -f
```

- Search apps with a specific string

```
adb shell pm list packages owasp

adb shell cmd package list packages owasp
```

Questions

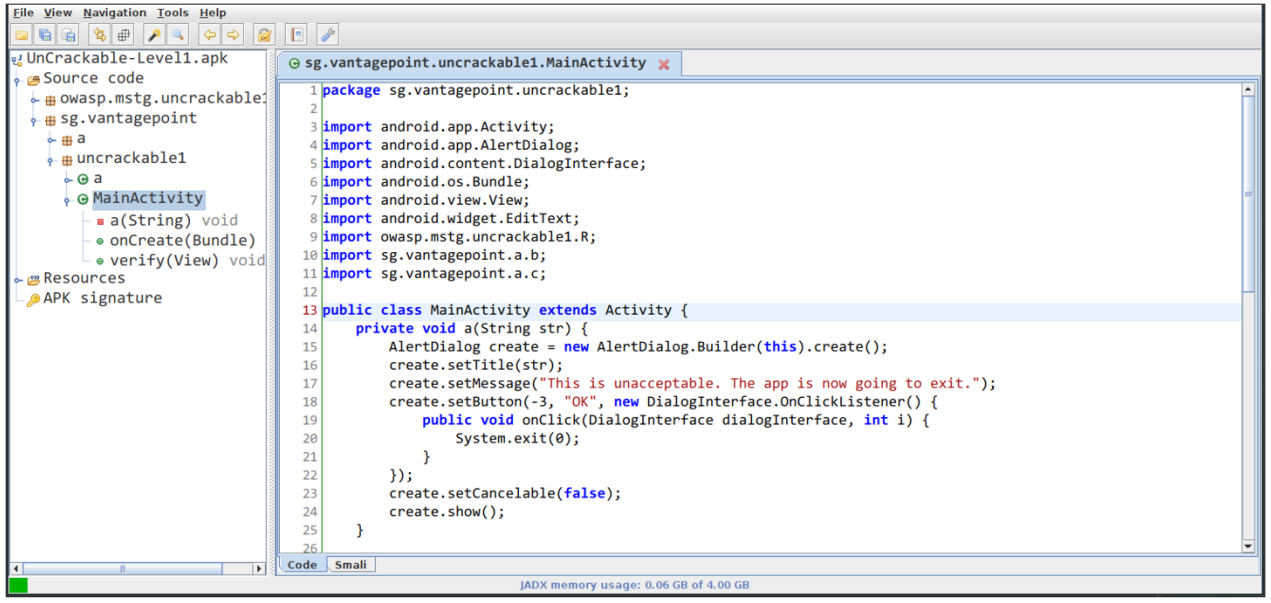
UnCrackable Level 1

UnCrackable 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 3 different ways to solve this crackme
  1. Code Tampering
  2. Frida with a rooted device
  3. Frida with a non-rooted device

UnCrackable Level 1 - Application Analysis

- First step is to analyze the source code of the Android app with JADX
- JADX is a tool allowing to decompile DEX and APK files
  - The tool outputs a Java representation of the DEX files



UnCrackable Level 1 - Executing JADX

- Let's launch JADX!

```
jadx-gui Uncrackable-level1.apk
```

- First, let's take a look at the AndroidManifest.xml file
  - The goal is to identify the different components of the app
  - And especially the entry point (*MainActivity*)

UnCrackable Level 1 - Android Manifest

- Here is the content of the `AndroidManifest.xml` file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1"
android:versionName="1.0" package="owasp.mstg.uncrackable1">
    <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="28"/>
    <application android:theme="@style/AppTheme" android:label="@string/app_name"
android:icon="@mipmap/ic_launcher" android:allowBackup="true">
        <activity android:label="@string/app_name"
android:name="sg.vantagepoint.uncrackable1.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</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

- Class: **`sg.vantagepoint.uncrackable1.MainActivity`**

```
public void onCreate(Bundle bundle) {
        if (c.a() || c.b() || c.c()) {
            a("Root detected!");
        }
        if (b.a(getApplicationContext())) {
            a("App is debuggable!");
        }
        super.onCreate(bundle);
        setContentView(R.layout.activity_main);
    }
```

UnCrackable Level 1 - Root Detection

- To identify if the device is rooted, the app performs 3 checks

  1. The `su` binary is present inside the `PATH`
  2. The build contains "test-keys"
  3. Common files used for rooting purposes exist on the device (i.e. Superuser.apk)

- Class: **`sg.vantagepoint.a.c`**

UnCrackable Level 1 - Root Detection

```
public class c {
    public static boolean a() {
        for (String file : System.getenv("PATH").split(":")) {
            if (new File(file, "su").exists()) {return true;}
        }
        return false;
    }
    public static boolean b() {
        String str = Build.TAGS;
        return str != null && str.contains("test-keys");
    }
    public static boolean c() {
        for (String file : new String[]{"/system/app/Superuser.apk", "/system/xbin/daemonsu",
"/system/etc/init.d/99SuperSUDaemon", "/system/bin/.ext/.su", "/system/etc/.has_su_daemon",
"/system/etc/.installed_su_daemon", "/dev/com.koushikdutta.superuser.daemon/"}) {
            if (new File(file).exists()) {return true;}
        }
        return false;
    }
}
```

UnCrackable Level 1 - Root/Debug Detection

- If the device is rooted/debugged, the app displays a dialog box and exits
  - Class: **`sg.vantagepoint.uncrackable1.MainActivity`**

```
private void a(String str) {
    AlertDialog create = new AlertDialog.Builder(this).create();
    create.setTitle(str);
    create.setMessage("This is unacceptable. The app is now going to exit.");
    create.setButton(-3, "OK", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialogInterface, int i) {
            System.exit(0);
        }
    });
    create.setCancelable(false);
    create.show();
}
```

UnCrackable Level 1 - Root/Debug Bypass

- Solutions to bypass the root/debug detection

  1. Easy way remove the `System.exit()` method
     - Goal: Avoid the app to exit
  2. Modify the `a()` function from `sg.vantagepoint.uncrackable1.MainActivity` class
     - Goal: Perform no action
  3. Modify the `a()`,`b()` and `c()` functions from `sg.vantagepoint.c` class
     - Goal: Return `false` in any case

UnCrackable Level 1 - Hidden Secret

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

```
public void verify(View view) {
        String str;
        String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
        AlertDialog create = new AlertDialog.Builder(this).create();
        if (a.a(obj)) {
            create.setTitle("Success!");
            str = "This is the correct secret.";
        } else {
            create.setTitle("Nope...");
            str = "That's not it. Try again.";
        }
...
```

UnCrackable Level 1 - Hidden Secret

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

```
 public static boolean a(String str) {
        byte[] bArr;
        byte[] bArr2 = new byte[0];
        try {
            bArr = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"),
Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=", 0));
        } catch (Exception e) {
            Log.d("CodeCheck", "AES error:" + e.getMessage());
            bArr = bArr2;
        }
        return str.equals(new String(bArr));
    }
```

UnCrackable Level 1 - Hidden Secret

- The secret passphrase is encrypted using AES
  - Class: **`sg.vantagepoint.a.a`**

```
    public static byte[] a(byte[] bArr, byte[] bArr2) {
        SecretKeySpec secretKeySpec = new SecretKeySpec(bArr, "AES/ECB/PKCS7Padding");
        Cipher instance = Cipher.getInstance("AES");
        instance.init(2, secretKeySpec);
        return instance.doFinal(bArr2);
    }
}
```

UnCrackable Level 1 - Hidden Secret Solution

- 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 `a()` from `sg.vantagepoint.a.a` to retrieve the return value

UnCrackable Level 1 - Code Tampering

- This process can be used to bypass security mechanisms or to modify the behavior of an app

- `apktool` is "a tool for reverse engineering Android APK files"
  - Decode resource files (XMLs, DEX files, etc.)
  - Build APK files

- When decompiling an Android app with `apktool`
  - You are going to obtain the **Dalvik bytecode** from the DEX files
  - The bytecode is stored inside the `smali` directory

UnCrackable Level 1 - Dalvik Bytecode

- Here is an example of [Dalvik bytecode]

```
.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
```

UnCrackable Level 1 - apktool decode

- How to tamper an Android app with `apktool`
- First, you need to decode the APK file with the **d** option

```
$ apktool d UnCrackable-Level1.apk
I: Using Apktool 2.4.1 on UnCrackable-Level1.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/mobexler/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

$ ls UnCrackable-Level1/
AndroidManifest.xml  apktool.yml  original  res  smali
```

UnCrackable Level 1 - apktool build

- After modifying the Dalvik bytecode or the resource files, you can rebuild the APK with the **b** option

```
apktool b UnCrackable-Level1 -o newUnCrackable1.apk
```

- Finally, you need to sign your APK using a self-signed certificate
  - This step is mandatory in order to install the app in your device
- In order to sign your APK file, you need to create your own keystore or you can use the debug keystore provided by the Android SDK

UnCrackable Level 1 - Create your keystore

- Create your own keystore using `keytool`

```
keytool -genkeypair -dname "cn=John Doe, ou=Security, o=Randorisec, c=FR" -alias bsides -
keystore keystore.jks -storepass dublin -validity 20000 -keyalg RSA -keysize 2048
```

  - `-alias`: Name of the key
  - `-keystore`: Name of the keystore
  - `-storepass`: Password used for the keystore and the key

- *Note:*
  - Using this command, the keystore password and the key password will be the same
  - It is also possible to generate a keystore using Android Studio

UnCrackable Level 1 - Sign your APK

For signing your APK file, you have 2 options

- jarsigner: Only supports v1 signature scheme (JAR signature)
  - Caution: From Android 11 (API level 30), v1 signature is not supported

```
jarsigner -verbose -keystore keystore.jks -storepass dublin newUnCrackable1.apk bsides
```

- apksigner: Official tool from Android SDK (since version 24.0.3)
  - Supports all the signature schemes (from v1 to v4)

```
~/Android/Sdk/build-tools/30.0.2/apksigner sign --ks keystore.jks --ks-pass pass:dublin
newUnCrackable1.apk
```

UnCrackable Level 1 - Code Tampering

- Decompile the app using apktool

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

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

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

- Remove the following lines

```
const/4 p1, 0x0
invoke-static {p1}, Ljava/lang/System;->exit(I)V
```

UnCrackable Level 1 - Code Tampering

- Edit the smali code in order to display the secret using logcat

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

- 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
```

- 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

```
 GNU nano 5.3              UnCrackable-Level1/smali/sg/vantagepoint/uncrackable1/a.smali
    invoke-virtual {v3, v0}, Ljava/lang/StringBuilder;→append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    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

- Build the new APK with apktool

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

- Sign the new APK with jarsigner

```
jarsigner -verbose -keystore keystore.jks -storepass dublin newUl1.apk bsides

~/Android/Sdk/build-tools/30.0.2/apksigner sign --ks keystore.jks --ks-pass pass:dublin
newUl1.apk
```

- Install the new APK

```
adb install newUl1.apk
```

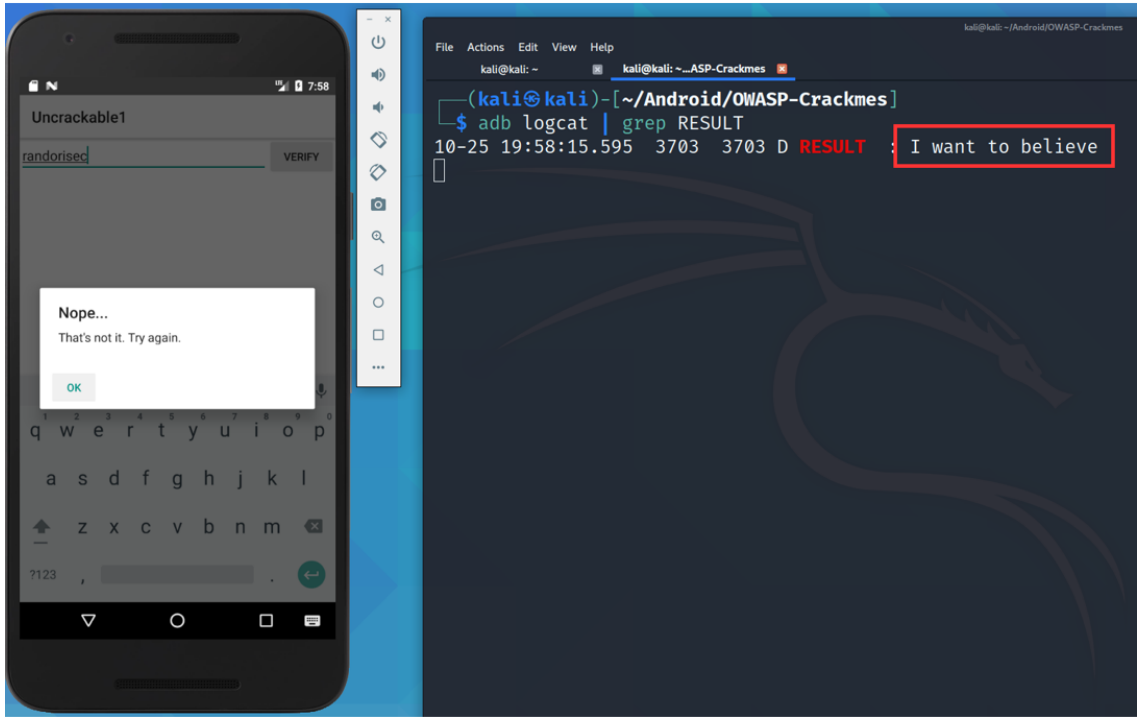
UnCrackable Level 1 - Solution 1 Demo

- Don't forget to remove the original app from the emulator

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

- Launch logcat
  - ... and grep the RESULT tag

```
adb logcat | grep RESULT
```



UnCrackable Level 1 - Frida

- 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

UnCrackable Level 1 - Frida

- Install Frida on your system (Easy way with `pip`)

```
# Install Frida and Python bindings
pip install frida frida-tools
# Upgrade to the last version available
pip install -upgradable frida
```

- Then, install Frida server on the Android device (or emulator)
  - Select the correct architecture on Github
  - If not sure which version, execute the following command

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

UnCrackable Level 1 - Frida

- Download the Frida server

```
VER=`frida --version`
ABI=`adb shell getprop ro.product.cpu.abi`
wget https://github.com/frida/frida/releases/download/$VER/frida-server-$VER-
android-$ABI.xz
xz -d frida-server-$VER-android-$ABI.xz
```

- Finally, upload the server on your emulator and execute it

```
adb push frida-server-$VER-android-$ABI /data/local/tmp/frida
adb shell "chmod 755 /data/local/tmp/frida"
adb shell "/data/local/tmp/frida"
```

- *Note:* `adb` needs to run with **root privileges**!

UnCrackable Level 1 - Frida Tools

- `frida-ps`
  - Get the list of running processes/applications

```
# -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
```

UnCrackable Level 1 - Frida Tools

- `frida-ps`
  - Get the list of installed applications with **`-i`** option

```
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
```

UnCrackable Level 1 - Frida CLI

- By default, Frida tries to attach to a running 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
```

UnCrackable Level 1 - Frida Scripts

- 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});`**
    - Perform code instrumentation inside the Java code
  - **`Java.use(class_name)`**
    - Use a specific Java class
  - **`overload`**
    - Overload a specific method (functions with different prototypes)
  - **`implementation`**
    - Tamper the implementation of the selected method

UnCrackable Level 1 - Frida Scripts

- Here is an example allowing to **hook** the `onCreate` function

```
Java.perform(function () {

  // Declare the Activity class as a variable
  var Activity = Java.use("android.app.Activity");

  // Hook the onCreate function
  Activity.onCreate.implementation = function () {
     console.log("[*] onCreate() got called!");
     this.onCreate();
  };
});
```

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 = String.fromCharCode.apply(null, retval);

  console.log("[*] Decrypted: " + password);
  return retval;
  };
});
```

UnCrackable Level 1 - Frida (Rooted device)

- Final solution: **solution-ul1.js**

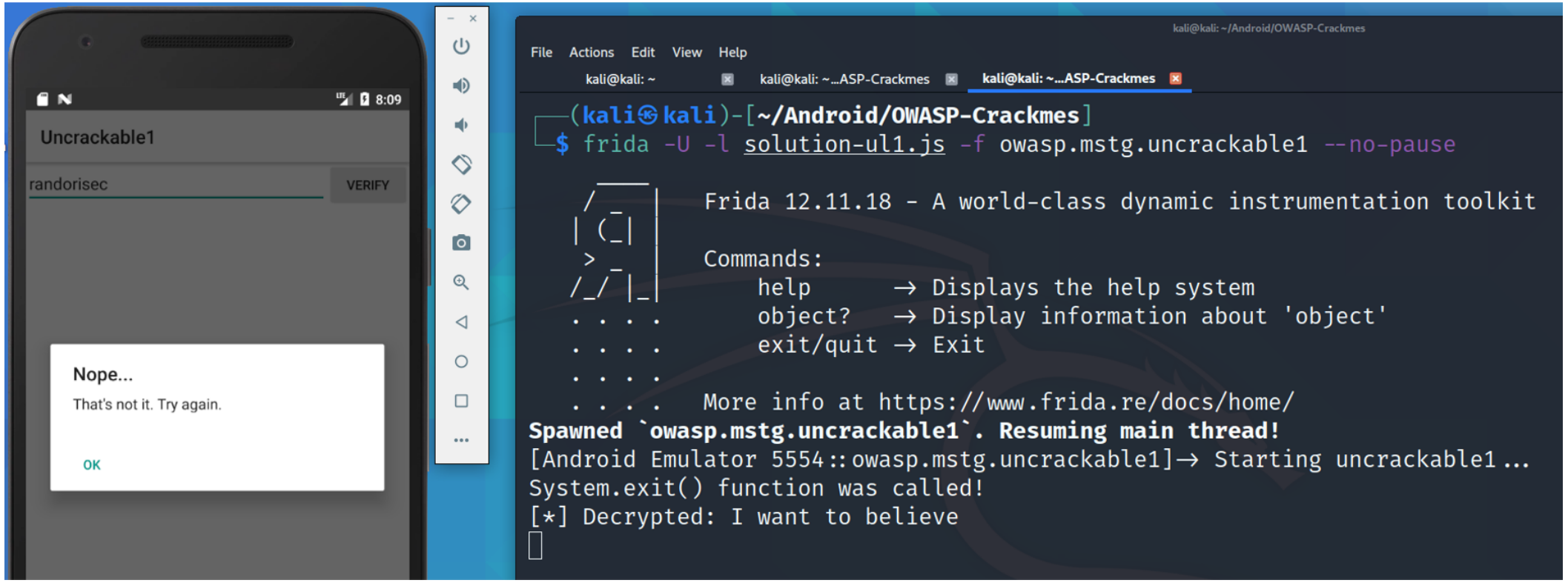
```
Java.perform(function () {
  console.log("Starting uncrackable1...");

  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 = String.fromCharCode.apply(null, retval);
    console.log("[*] Decrypted: " + password);
    return retval; };
});
```

UnCrackable Level 1 - Solution 2 Demo

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



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)

- Decompile the app using apktool

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

- Edit the AndroidManifest.xml file

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

- Add the INTERNET permission

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

UnCrackable Level 1 - Frida (lib-gadget)

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

```
VER=`frida --version`
ABI=`adb shell getprop ro.product.cpu.abi`
wget https://github.com/frida/frida/releases/download/$VER/frida-gadget-$VER-android-$ABI.so.xz

xz -d frida-gadget-$VER-android-$ABI.so.xz
```

- Add the library inside the app

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

cp frida-gadget-$VER-android-$ABI.so UnCrackable-Level1/lib/x86_64/libfrida-gadget.so
```

UnCrackable Level 1 - Frida (lib-gadget)

- 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)

- Now, build the new APK with `apktool`

```
apktool b UnCrackable-Level1 -o ul1-gadget.apk
```

- And sign it!

```
/home/mobexler/Android/Sdk/build-tools/30.0.2/apksigner sign --ks keystore.jks --ks-pass
pass:dublin ul1-gadget.apk
```

- Finally, uninstall the original app and install the new one

```
adb uninstall owasp.mstg.uncrackable1

adb install ul1-gadget.apk
```

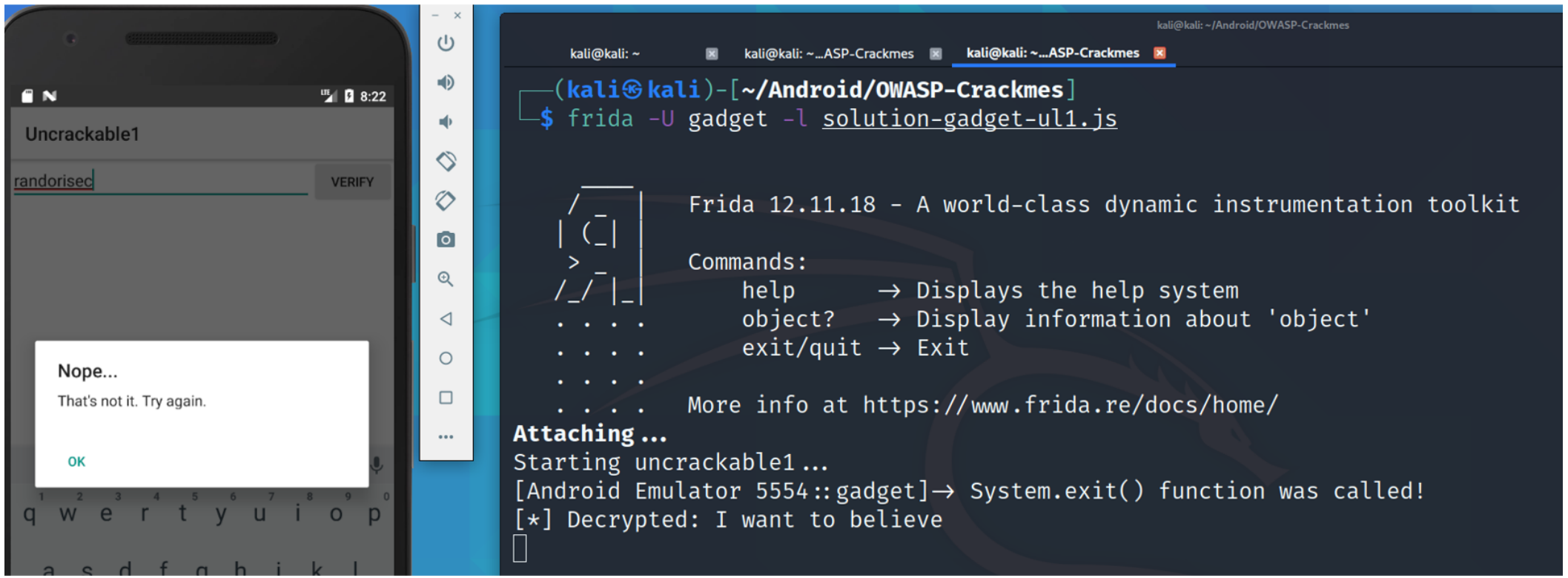
UnCrackable Level 1 - Frida (lib-gadget)

- Finally, just launch Frida using this 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 - Solution 3 Demo

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



UnCrackable Level 1 - Objection

- Tamper the app with `Objection`
- Execute `Objection` by specifying the Frida version

```
objection patchapk --source UnCrackable-Level1.apk -V $VER
```

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

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

UnCrackable Level 1 - Objection

- After this modification, you can just install the newly created APK and use the previous Frida script:

```
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 - Questions

UnCrackable Level 2

UnCrackable 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
  - So we need to analyze the **Native Code** embedded inside the app

UnCrackable Level 2 - Native Code

- 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

UnCrackable Level 2 - Native Functions

- Standard way to load Native Code (or static approach)
  - First, load the shared library (usually inside the Main Activity)
  - Then, define the prototype of your 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();
```

- *Note*: The `native` functions can be loaded dynamically inside the shared library using the `RegisterNatives` method

UnCrackable Level 2 - Shared Library

- 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());
}
```

UnCrackable Level 2 - Analysis

- Looking at the code using jadx

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

- A shared library named **foo** is loaded inside the `onCreate()` method
  - Class: **`sg.vantagepoint.uncrackable2.MainActivity`**

```
public class MainActivity extends c {
  private CodeCheck m;

  static {
      System.loadLibrary("foo");
  }
```

UnCrackable Level 2 - Analysis

- 2 native functions are defined inside the Android app
  - Class: **`sg.vantagepoint.uncrackable2.MainActivity`**
  - Function: **`init`**

```
private native void init();
```

  - Class: **`sg.vantagepoint.uncrackable2.CodeCheck`**
  - Function: **`bar`**

```
private native boolean bar(byte[] bArr);

public boolean a(String str) {
    return bar(str.getBytes());
}
```

UnCrackable Level 2 - Analysis

- The `a()` function calls the native function `bar()` to check the string provided by the user
  - Class: **`sg.vantagepoint.uncrackable2.MainActivity`**

```
public void onCreate(Bundle bundle) {
      init();
      [...]
      // the object CodeCheck is created on the onCreate method
      this.m = new CodeCheck();
}
public void verify(View view) {
    [...]
    // the a function from CodeCheck object is called to check the secret
    if (this.m.a(obj)) {
        create.setTitle("Success!");
        charSequence = "This is the correct secret.";
```

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

```
$ 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
```

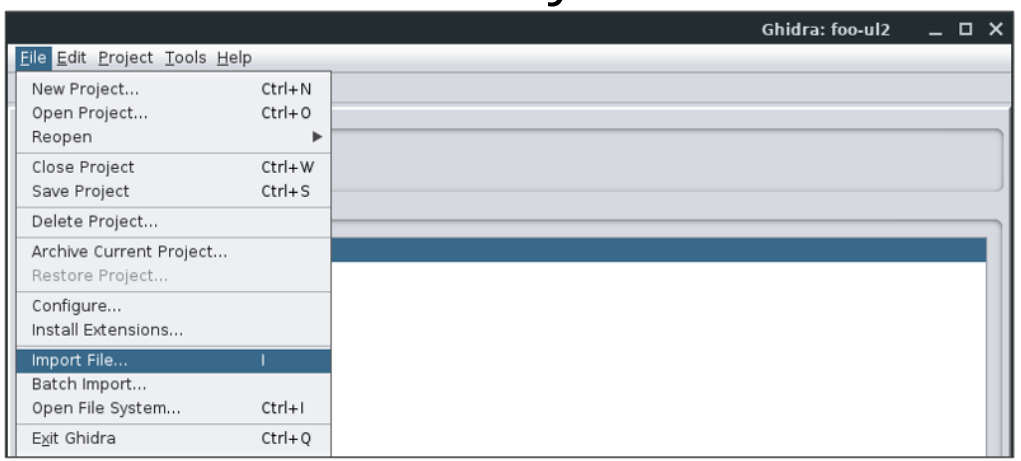
UnCrackable Level 2 - Ghidra

- We are going to use Ghidra to reverse the `libfoo` library
  - We are going to focus on x86_64 architecture
- Ghidra is *"a software reverse engineering (SRE) suite of tools developed by NSA"*
  - Useful to reverse engineer native libraries
- Launch Ghidra using the following command

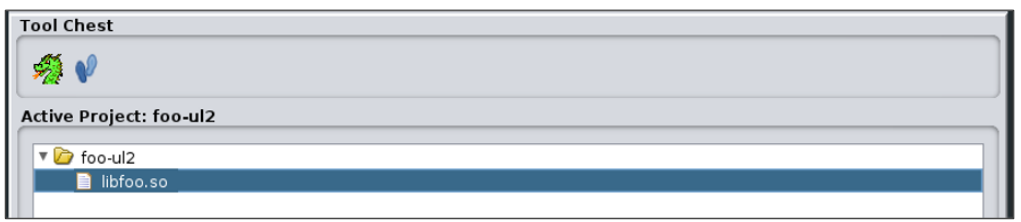
```
/home/mobexler/AndroidZone/ghidra_9.1.2_PUBLIC/ghidraRun
```

UnCrackable Level 2 - Launch Ghidra

- Start a **`File -> New Project…`** and then select **`Non-Shared Project`**
- Set the project name **`foo-ul2`** and click `Finish`
- Now, you can import the `libfoo` library

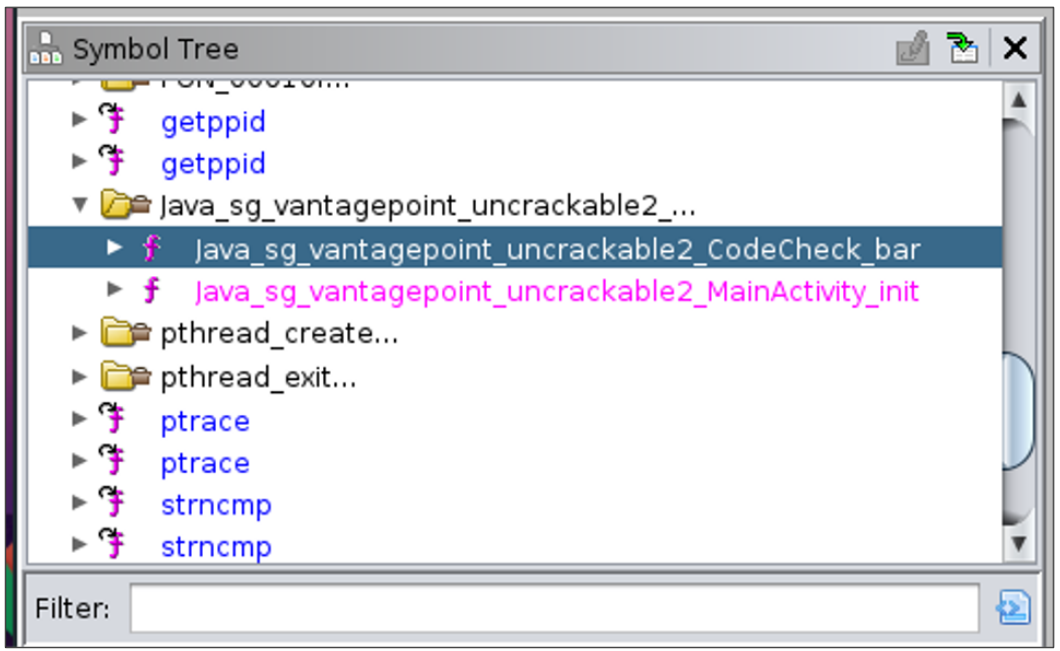


- Then, double click on the `libfoo.so` file



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

```
undefined8
Java_sg_vantagepoint_uncrackable2_CodeCheck_bar(long *param_1,undefined8 param_2,undefined8
param_3)
{
    // retrieve the user input param_3 (byte array)
    __s1 = (char *)(**(code **)(*param_1 + 0x5c0))(param_1,param_3,0);
    // retrieve the length of the user input
    iVar1 = (**(code **)(*param_1 + 0x558))(param_1,param_3);
    // check if the length of the user input is equlas to 23 (0x17)
    if (iVar1 == 0x17) {
      // compare the user input with the value of the hidden secret (local_38)
      iVar1 = strncmp(__s1,(char *)&local_38,0x17);
```

UnCrackable Level 2 - Frida (Native Code)

- Solution

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

- How to perform dynamic instrumentation on **Native Code**?
  - Frida provides an API to hook native code
  - The Interceptor API

UnCrackable Level 2 - Interceptor

- **`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 (Memory.readUtf8String(args[1]));
  },
  onLeave: function (retval) {
  }
});
```

UnCrackable Level 2 - Solution Overview

- In order to avoid displaying parameters for all `strncmp()` calls
  - Check if the 1st parameter starts with a specific string
  - 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")`**

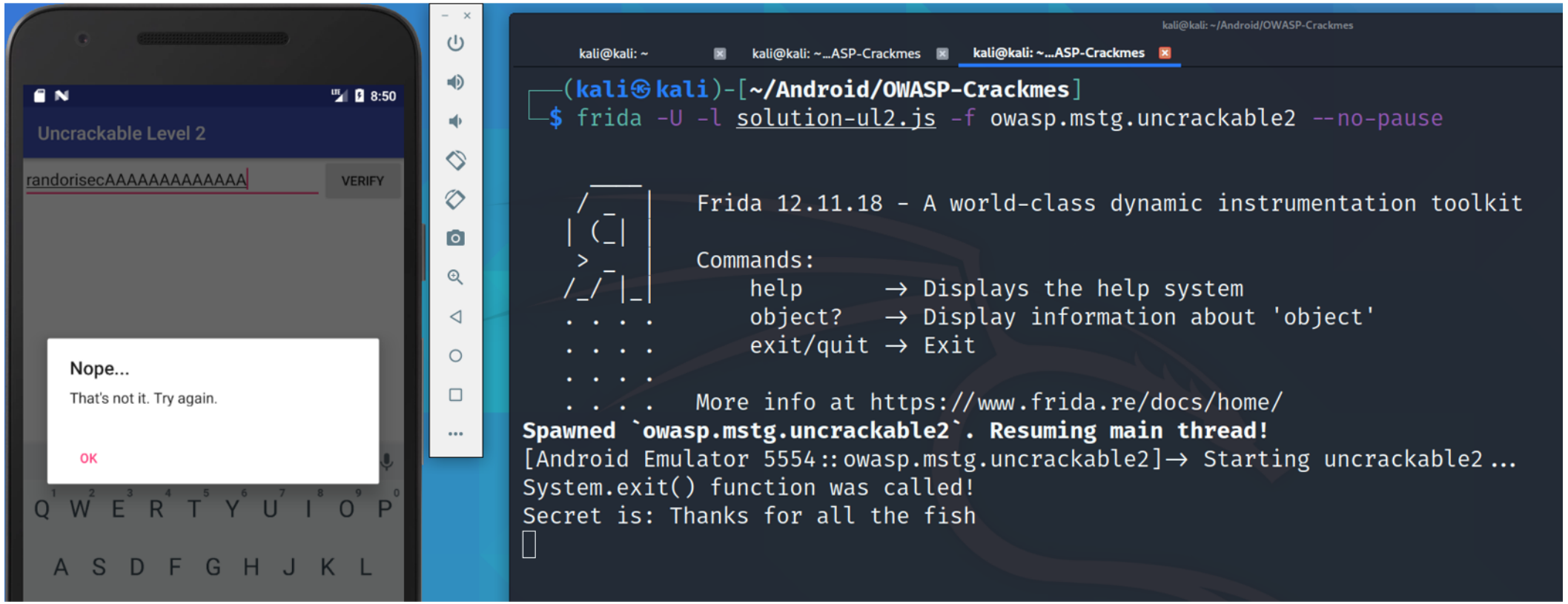
UnCrackable Level 2 - Frida Solution

- Here is the final Frida script: `solution-ul2.js`

```
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 - Solution Demo

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



UnCrackable Level 2 – Questions

UnCrackable Level 3

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

UnCrackable Level 3 – Anti-Tampering

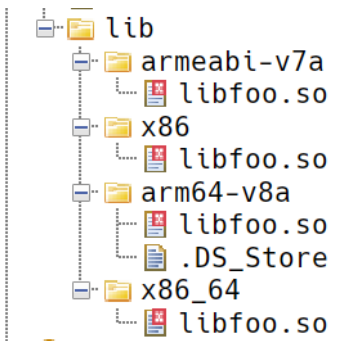
- Checks are performed inside the Java code
  - Verify the CRC of all the libraries (**lib** folder)
  - Verify the CRC of the `classes.dex` file
- If one of those files is modified
  - The variable `tampered` is set to 31337
  - The CRC values for the libraries are stored in the `strings.xml` file
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

UnCrackable Level 3 – Anti-Tampering

- Here is an extract of the `strings.xml` file

```
<string name="arm64_v8a">1608485481</string>
<string name="armeabi_v7a">881998371</string>
<string name="x86">1618896864</string>
<string name="x86_64">2856060114</string>
```

- For each file, a CRC code is hardcoded inside the app



UnCrackable Level 3 – Anti-Tampering

- Class: **`sg.vantagepoint.uncrackable3.MainActivity`**
- Function: **`verifyLibs`**

```
private void verifyLibs() {
                [...]
                Entry entry = (Entry) it.next();
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.append("lib/");
                stringBuilder.append((String) entry.getKey());
                stringBuilder.append("/libfoo.so");
                String stringBuilder2 = stringBuilder.toString();
                ZipEntry entry2 = zipFile.getEntry(stringBuilder2);
                [...]
                // Check if the CRC stored inside strings.xml is different
                if (entry2.getCrc() != ((Long) entry.getValue()).longValue()) {
                    tampered = 31337;
```

UnCrackable Level 3 – Anti-Hooking

- Anti-hooking checks are performed inside the shared library
  - If Frida or Xposed binaries are detected, the app crashes!

```
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

- Analyzing the shared library with Ghidra, we can identify which function is performing the check
  - Search for **"frida"** or **"xposed"** strings

```
else {
    while( true ) {
      pcVar1 = fgets(acStack568,0x200,__stream);
      if (pcVar1 != (char *)0x0) break;
     [...]
      __stream = fopen("/proc/self/maps","r");
      if (__stream == (FILE *)0x0) goto LAB_0010386c;
    }
    pcVar1 = strstr(acStack568,"frida");
    if (pcVar1 != (char *)0x0) break;
     pcVar1 = strstr(acStack568,"xposed");
    } while (pcVar1 == (char *)0x0);
    pcVar1 = "Tampering detected! Terminating...";
```

UnCrackable Level 3 – Anti-Hooking

- The `strstr()` function is used to search "frida" or "xposed" strings inside `/proc/self/maps`
  - `char *strstr(const char *haystack, const char *needle)`
  - *"The `strstr()` function finds the first occurrence of the substring needle in the string haystack."*
- If "frida" or "xposed" is found, the `goodbye()` function is called
  - As the names implies, the app stops

UnCrackable Level 3 – Anti-(Tamper|Hook)ing

- To sum-up

  1. We need to bypass the root detection (same as Level 1)
  2. For the tampering protection, we just avoid to patch the APK :)
  3. Finally, we hook the `strstr()` function in order to bypass the anti-hooking mechanism

- The `strstr()` function is provided by the libc library

  - Our hook is just going to check if one of the parameter is "frida"
  - If this is the case, the return value is modified to return `false`

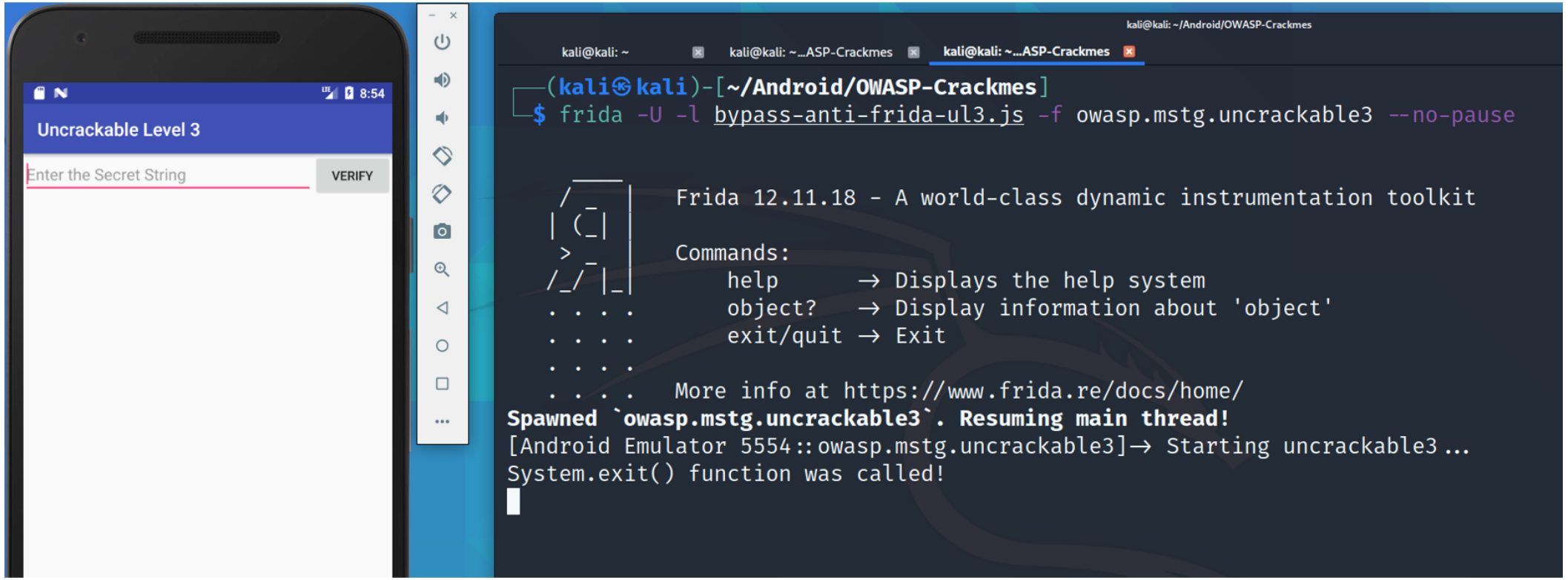
UnCrackable Level 3 – Bypass Root and Hooking Detection

bypass-anti-frida-ul3.js

```
Java.perform(function () {
  console.log("Starting uncrackable3...");
  var sys = Java.use("java.lang.System");
  sys.exit.implementation = function() { console.log("System.exit() function 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 – Frida bypass

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



UnCrackable Level 3 – Hidden Secret

- Same as Level 2, the hidden secret is inside the shared library: `libfoo.so`
  - On the `verify()` function, the `check_code` function is used to check the user input
  - Class: **`sg.vantagepoint.uncrackable3.MainActivity`**

```
    this.check = new CodeCheck();
    [...]

public void verify(View view) {
    String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
    AlertDialog create = new Builder(this).create();
    if (this.check.check_code(obj)) {
        create.setTitle("Success!");
        create.setMessage("This is the correct secret.");
    } else {
```

UnCrackable Level 3 – Hidden Secret

- Class: **`sg.vantagepoint.uncrackable3.CodeCheck`**
  - The `check_code` function calls the `bar` native function

```
public class CodeCheck {

private static final String TAG = "CodeCheck";

private native boolean bar(byte[] bArr);

public boolean check_code(String str) {
  return bar(str.getBytes());
}
}
```

UnCrackable Level 3 – Hidden Secret

- Let's have a look at the `bar` native function using Ghidra
  - Function: **`Java_sg_vantagepoint_uncrackable3_CodeCheck_bar`**

```
undefined8
Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(long *param_1,undefined8 param_2,undefined8
param_3)
{
FUN_001012c0(local_48);
lVar2 = (**(code **)(*param_1 + 0x5c0))(param_1,param_3,0);
iVar1 = (**(code **)(*param_1 + 0x558))(param_1,param_3);
if (iVar1 == 0x18) {
  uVar4 = 0;
  do {
    if (((*(byte *)(lVar2 + uVar4) != ((&DAT_00107040)[uVar4] ^ local_48[uVar4])) ||
        (*(byte *)(lVar2 + 1 + uVar4) != ((&DAT_00107041)[uVar4] ^ local_48[uVar4 + 1]))) ||
        (*(byte *)(lVar2 + 2 + uVar4) != ((&DAT_00107042)[uVar4] ^ abStack70[uVar4])))
```

UnCrackable Level 3 – Hidden Secret

- Analyzing the `bar()` native function, we can identify that

  1. The length of the secret is 24 (0x18)

  2. A XOR operation is performed against 2 variables

     - One variable is used like a key (**`xorkey`**)
     - One variable is used like an "encrypted" secret (**`secret`**)

  3. The `xorkey` is obtained from the `init()` function

  4. The `secret` is given by an internal function from the shared library

UnCrackable Level 3 – Hidden Secret

- To sum up, the following pseudo-code simplifies the workflow of the `bar()` function

```
function Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(JNIenv, jobject, user_input) {
    lVar2 = user_input;
    iVar1 = length(user_input);
    FUN_001012c0(&local_48);
    // now local_48 contains the "encrypted" secret
    // check if the user_input length is equals to 24
    if (iVar2 == 24){
        // the xorkey retrieved from the Java code
        xorkey = &DAT_00107040;
        // XOR operation performed and comparison to the user input
        for(i=0; i<24; i++) {
            if(iVar1[i] != (secret[i] ^ xorkey[i]) break;
        }
    }
}
```

UnCrackable Level 3 – Hidden Secret

- The `xorkey` is provided to the shared library using the `init()` function
  - The hardcoded `xorkey` can be found on the MainActivity class

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

    [...]

    public void onCreate(Bundle bundle) {
      verifyLibs();
      init(xorkey.getBytes());
```

UnCrackable Level 3 – Hidden Secret

- The xorkey is **pizzapizzapizzapizzapizz**
  - This value is copied by the shared library into an internal variable `DAT_00107040`

```
void Java_sg_vantagepoint_uncrackable3_MainActivity_init (long *param_1,undefined8
param_2,undefined8 param_3)
{
  char *__src;

  FUN_00103910();
  __src = (char *)(**(code **)(*param_1 + 0x5c0))(param_1,param_3,0);
  strncpy(&DAT_00107040,__src,0x18);
  (**(code **)(*param_1 + 0x600))(param_1,param_3,__src,2);
  _DAT_0010705c = _DAT_0010705c + 1;
  return;
}
```

UnCrackable Level 3 – Hidden Secret

- The encrypted `secret` is provided by another function `FUN_001012c0`
  - This function performs multiple operations which are difficult to analyze using static analysis
  - But, in fact, it is not a problem, we just want to retrieve the result at the end :)

- We are going to use Frida to retrieve the value of the `secret`
  - Because the function is not named, we need to use the address of the function
  - Our analysis was performed against `x86_64` so we just get the 4 bytes of the address: `0x000012c0`

UnCrackable Level 3 – Hidden Secret

- Here is the Frida script allowing to get the content of the `secret`

```
function hook_native_libs() {
  var offset_fun = 0x000012c0;
  /* retrieve the base address of the "foo" library */
  var p_foo = Module.findBaseAddress('libfoo.so');
  var p_fun_secret = p_foo.add(offset_fun);
  Interceptor.attach( p_fun_secret, {
    onEnter: function (args) {
    /* get a pointer on the secret variable */
      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 – Hidden Secret

- 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 () {
  // 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(); });
```

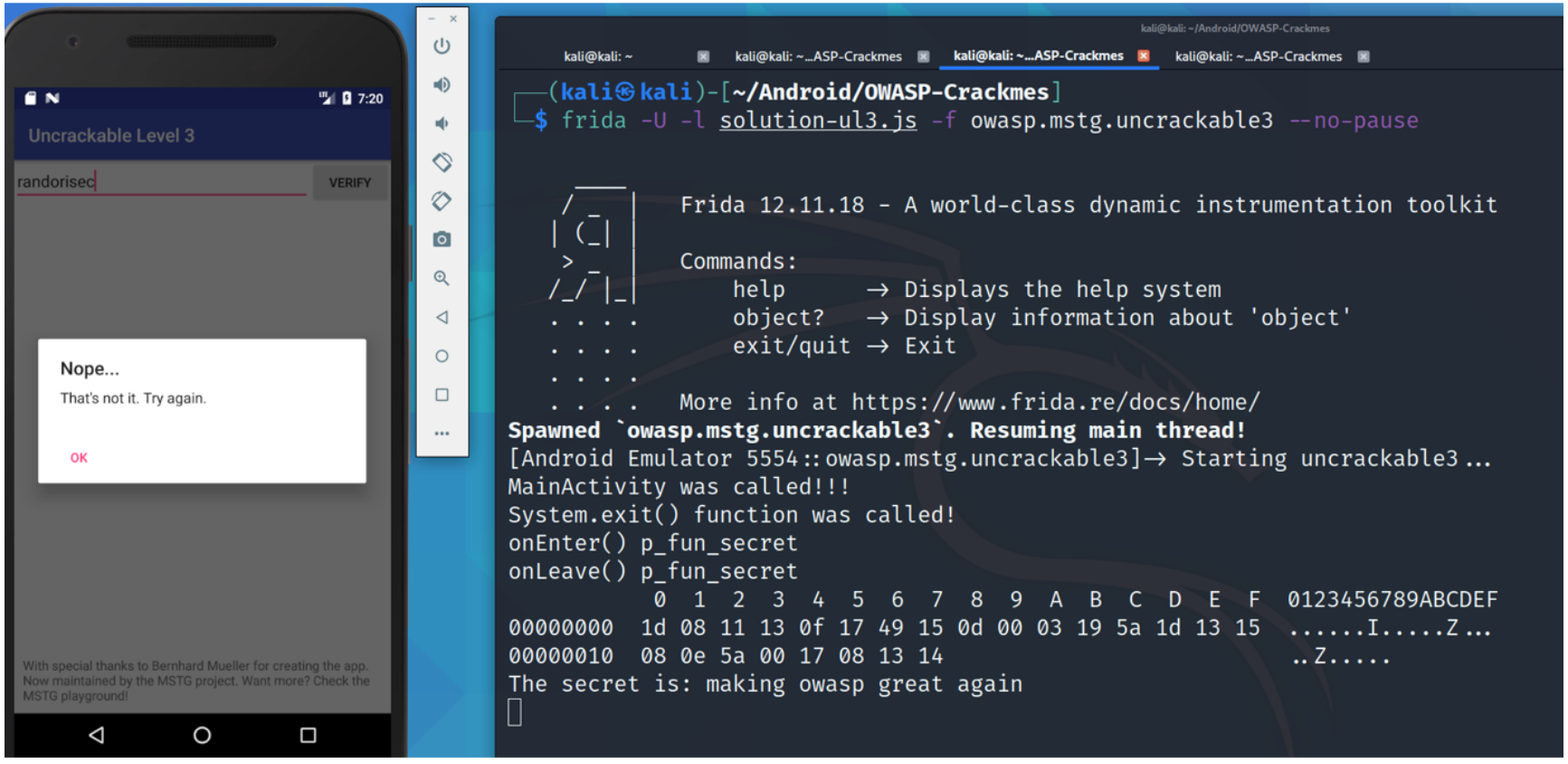
UnCrackable Level 3 – Solution

- To sum-up

  1. The "root" detection was bypassed like Level 1 and 2
  2. The anti-hooking protection against Frida was bypassed using Frida in order to patch the `strstr()` function
  3. The `xorkey` is hardcoded inside the Java code
  4. The "encrypted" `secret` was retrieved by hooking the unamed function returning the `secret`

UnCrackable Level 3 – Solution Demo

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



UnCrackable Level 3 – 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 Basic

- 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 Advanced

- 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