MOBILE PENTESTING WITH FRIDA

#who_are_us

LAURA GARCÍA

> 10 years of experience in Computer Security

Senior Security Consultant

MARTA BARRIO

> 7 years of experience in Computer Security

Security Researcher / Pentester at Deloitte Hacking Team

Execution of controlled real-world attacks again systems, products and facilities. Perform penetration tests on various technologies,such as web applications, mobile applications, AD attacks, Wireless media and infrastructures.

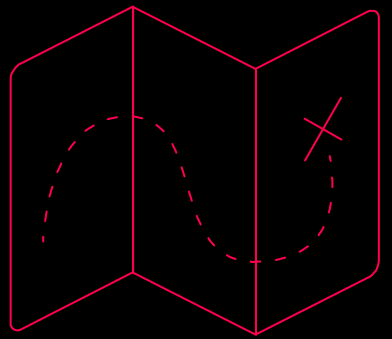
https://es.linkedin.com/in/laura-garcia-cybersec

https://es.linkedin.com/in/martabarriomarcos

@lain7z

@martrudix

#index



1.
History

Let’s start with the beginning

1.1 History

- Created by Ole Andre V. Ravås (@oleavr)
- Brainstorming with Håvard Sørbø (@hsorbo):
  - Turn tedious manual reverse-engineering into something much more fun, productive and interactive.
- Hackathons

- Result: **https://www.frida.re**

> *Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.*

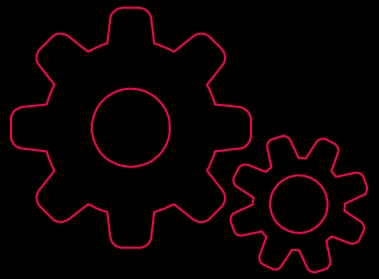
1.2 What is Frida?

- Free and open source dynamic code instrumentation toolkit written in C that works by injecting a JavaScript engine (Duktape and V8) into the target process.
- Lets you execute snippets of JavaScript into native apps on multiple platforms such as Android and iOS.
- Implements code injection by writing code directly into process memory.
- JS gets executed with full access to memory, hooking functions and even calling native functions inside the process.


https://t.me/fridadotre


@fridadotre

https://codeshare.frida.re/



2. Usage

First steps

2.1 Installation
Client

Requirements:

- Python – latest 3.x is highly recommended
- Multiplatform (GNU/Linux, MacOS or Windows)

- Terminal:

```
$ pip3 install frida-tools # CLI tools
$ pip3 install frida       # Python bindings
$ npm install frida        # Node.js bindings
```

2.1 Installation
Remote server

- Android:
  - Download frida-server from releases pages https://github.com/frida/frida/releases

```
$ adb root # might be required
$ adb push frida-server /data/local/tmp/
$ adb shell "chmod 755 /data/local/tmp/frida-server"
$ adb shell "/data/local/tmp/frida-server &"
```

- iOS:
  - Cydia: Add Frida's repository: https://build.frida.re
  - Install Frida package

2.1.1 Installation

- A quick test, to make the basics are working:

```
# Connect Frida to your device over USB and list all running
processes
$ frida-ps -U
 PID  Name
----  -----------------------------------
 124  adbd
1171  android.process.acore
1437  android.process.media
 267  audioserver
 257  batteryd
 268  cameraserver
3492  com.android.calendar
4252  com.android.chrome
```

2.2 Modes of Operation

- **Injected**: provides a two-way communication channel to spawn an existing program, attach to a running program, or hijack one as it's being spawned.
- **Embedded**: on jailed iOS and Android systems. *frida-gadget*, is a shared library to embed into the app, which will automatically run frida-server.
- **Preloaded**: *frida-gadget* when configured to run autonomously by loading a script from the filesystem. Similar to LD_PRELOAD or DYLD_INSERT_LIBRARIES.

2.3 Gadget

- Is a shared library meant to be loaded by programs to be instrumented when the Injected mode of operation isn't suitable. This may be done by:
  - Modifying the source code of the program.
  - Patching it or one of its libraries, e.g. by using a tool like insert_dylib.
  - Using a dynamic linker feature like *LD_PRELOAD* or *DYLD_INSERT_LIBRARIES.*

2.3.1 Gadget
iOS (without jailbreak)

- What we need:
  - IPA decrypted
  - Developer Apple account (free)
  - OSX with Xcode
  - brew, applesign, fastlane sign, ios-deploy, insert_dylib

2.3.2 Gadget
iOS (without jailbreak)

- How to get the IPA decrypted:
  - Dumped it from a Jailbreak device.
    - Clutch –d com.name.app
    - Frida-ios-dump
      https://github.com/AloneMonkey/frida-ios-dump
  - Ask the developer.

```
$ iproxy 2222 22
$ ssh root
// start the APP
$ /dump.py -l
$ /dump.py <APP>
```

2.3.3 Gadget
iOS (without jailbreak)

Inject **FridaGadget.dylib** into the iOS app:

- Generate embedded.mobileprovision with Xcode
- Sign dylib and IPA

```
$ applesign –m embedded.mobileprovision –w –l FridaGadget.dylib  APP.ipa
```

- Unzip IPA

```
$ unzip APP-resigned.ipa
```

- Deploy on device

```
$ ios-deploy --bundle Payload/*.app -m -L
```

- Run frida

```
$ frida -U Gadget --no-pause
```

2.3.4 Gadget
Android (without root)

Inject **FridaGadget.so** into the APK:

- Disassemble APK

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

- Know the arch

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

- Add the frida-gadget into the APK's /lib folder. Gadget libraries for each architecture on release pages.

```
$ wget
https://github.com/frida/frida/releases/download/x.x.x/frida-gadget-x.x.x
-android-arm.so.xz

$ unxz frida-gadget-x.x.x-android-arm.so.xz

$ cp frida_libs/armeabi/frida-gadget-x.x.x-android-arm.so
output/lib/armeabi/libfrida-gadget.so
```

2.3.5 Gadget
Android (without root)

- Find main activity

```
$ find . | grep -i main | grep smali$
```

- Inject a System.loadLibrary("frida-gadget") call into the bytecode of the app, ideally before any other bytecode executes or any native code is loaded. Add the following smali code to the constructor.

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

- Ensure network permissions in AndroidManifest.xml

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

- Repackage the application

```
$ apktool b -o repackaged.apk output/
```

2.3.6 Gadget
Android (without root)

- Sign the updated APK using your own keys and zipalign

```
$ keytool -genkey -v -keystore custom.keystore -alias mykeyaliasname -keyalg RSA -keysize 2048 -validity 10000
$ jarsigner -sigalg SHA1withRSA -digestalg SHA1 -keystore mycustom.keystore -storepass mystorepass repackaged.apk mykeyaliasname
$ jarsigner -verify repackaged.apk
$ zipalign 4 repackaged.apk repackaged-final.apk
```

- Install the updated APK to a device

```
$ adb install repackaged-final.apk
```

2.4 Frida Tools

- Frida CLI
- frida-trace
- frida-ps
- frida-ls-devices

2.4.1 Tools
Frida CLI

Interface that aims to emulate a lot of the nice features of IPython, which tries to get you closer to your code for rapid prototyping and easy debugging.

```
# Start debugging Chrome remotely
$ frida -U com.android.chrome

# Start debugging Safari locally
$ frida Safari
```

```
$ frida -U com.android.chrome
     ____
    / _  |   Frida 12.7.5 - 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/

[Genymotion Google Nexus 9::com.android.chrome]-> Java.available
true
[Genymotion Google Nexus 9::com.android.chrome]-> Java.androidVersion
"7.1.1"
```

2.4.2 Tools
Frida CLI + script

```
# Spawn class.app.com and load frida-bypass script over USB
$ frida -U -l frida-bypass.js -f class.app.com --no-pause

# Attach an application and load List classes script over USB
$ frida -U -l list-classes.js Gadget

# Spawn class.app.com and load find classes script over USB
$ frida -U -l find-classes.js -f class.app.com
```

2.4.3 Tools

Frida CLI + script

```
$ more prueba.js
Java.perform(function() {
   var Activity = Java.use("android.app.Activity");
   Activity.onResume.implementation = function () {
      console.log("[*] onResume() got called!");
      this.onResume();
      }; });
```

```
$ frida -U -l prueba.js -f com.android.chrome --no-pause
     ____
    / _  |   Frida 12.7.5 - 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 `com.android.chrome`. Resuming main thread!
[Genymotion Google Nexus 9::com.android.chrome]-> [*] onResume() got called!
```

2.4.4 Tools
frida-ps

Command-line tool for listing processes.

```
# Connect Frida to your device over USB and list running processes
$ frida-ps -U

# List running applications
$ frida-ps -Ua

# List installed applications
$ frida-ps -Uai
```

2.4.5 Tools

frida-trace + custom handlers

Tool for dynamically tracing function calls. It will automatically create handlers in the folder __handlers__.

```
# Trace specific (low-level) library calls
$ frida-trace -U com.android.chrome -i "open"

# Trace recv* and send* APIs in Safari
$ frida-trace -i "recv*" -i "read*" Safari

# Trace ObjC method calls from Class in Safari
$ frida-trace -U -m "-[NSURL *]" Safari
```

2.4.6 Tools

frida-trace + custom handlers

```
# This generates a little JavaScript in __handlers__/libsqlite.so/open.js,
which Frida injects into the process. The script traces all calls to the
open function in libsqlite.so that could be modify.
```

```
$ frida-trace -U com.android.chrome -i "open"
Instrumenting functions...
open: Auto-generated handler at "/Users/lain/__handlers__/libsqlite.so/open.js"
Started tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x109c */
  6624 ms  open(path="/dev/ashmem", oflag=0x2)
  6632 ms  open(path="/dev/ashmem", oflag=0x0)
  6633 ms  open(path="/dev/alarm", oflag=0x0)
  6651 ms  open(path="/dev/ashmem", oflag=0x2)
  6666 ms  open(path="/dev/alarm", oflag=0x0)
  6666 ms  open(path="/dev/alarm", oflag=0x0)
  6691 ms  open(path="/dev/alarm", oflag=0x0)
  6691 ms  open(path="/dev/ashmem", oflag=0x2)
  6734 ms  open(path="/dev/alarm", oflag=0x0)
  6736 ms  open(path="/dev/ashmem", oflag=0x2)
  6737 ms  open(path="/proc/meminfo", oflag=0x0)
  6737 ms  open(path="/dev/ashmem", oflag=0x2)
  6739 ms  open(path="/dev/ashmem", oflag=0x2)
           /* TID 0x10a1 */
  6766 ms  open(path="/dev/ashmem", oflag=0x2)
           /* TID 0x12b9 */
  6771 ms  open(path="/dev/ashmem", oflag=0x2)
```

2.4.7 Tools
frida-ls-devices

Command-line tool for listing attached devices.

```
# Lists all the attached devices
$ frida-ls-devices
```

```
$ frida-ls-devices
Id                  Type    Name
------------------  ------  ------------------------
local               local   Local System
192.168.16.5:5555   usb     Genymotion Google Nexus 9
tcp                 remote  Local TCP
```

2.5 JavaScript API

Main Contents

Python Bindings

2.5.1 JavaScript API
Main Contents

- console
  - console.log(line), console.warn(line), console.error(line)
- Module
  - Module.findExportByName(moduleName|null, exportName), Module.findBaseAddress(name)
- Memory
  - Memory.scan() – Scan process memory for occurrences of a string
- Process
  - Process.id, Process.arch, Process.getCurrentThreadId();

https://www.frida.re/docs/javascript-api/

2.5.2 JavaScript API
ObjC

- ObjC
  - ObjC.available – Is Objective-C runtime loaded ?
  - ObjC.classes – All loaded classes.
  - [NSString stringWithString:@"Hello World"] becomes var NSString = ObjC.classes.NSString; NSString.stringWithString_("Hello World");
  - Object.keys(ObjC.classes).forEach(function (className) { ... }); – Methods of a classname.

- ObjC.Object has some properties:
  - $kind, $super, $superClass,$class, $className,$methods,$ownMethods

2.5.3 JavaScript API
Interceptor

Intercept native function calls to run your own code at function entry and exit.

- Interceptor.attach(target, callbacks)
  - onEnter: function (args)
  - onLeave: function (retval)
    - retval.replace

```
var hook = ObjC.classes.<CLASS>["+ <METHOD>:"]
Interceptor.attach(hook.implementation, {
  onEnter: function (args) {
    console.log(ObjC.Object(args[2]));
  },
  onLeave: function (retval) {
    Memory.readUtf8String(retval);
    } };
```

2.5.4 JavaScript API
Interceptor

- Interceptor.replace(target, replacement)

Use NativeCallback to implement a replacement.

- NativeCallback(func, returnType, argTypes[, abi])

```
if (ObjC.available){
   var hook = ObjC.classes.<CLASS>["- <METHOD>:"]
   Interceptor.replace(hook.implementation, new
NativeCallback(function() { return; }, int, [])  );
```

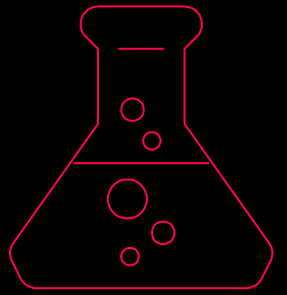
2.5.5 JavaScript API
Java

- Java
  - Java.available – Is the current process has the a Java VM loaded?
  - Java.use(className) – Instantiate Java objects and call static and non-static class methods. You can even replace a method implementation.
    - aClass.method.implementation = function(args) { .. }
  - Java.enumerateLoadedClasses(callbacks) – All loaded classes right now.
  - Java.choose(className, callbacks) – Enumerate live instances of specific classes.

2.5.6 Frida Bindings

Python binding to spawn an app

```
import frida, sys

ss = """
Java.perform(function () {
  // declare classes that are going to be used
  const System = Java.use('java.lang.System');
  const Log = Java.use("android.util.Log");
  const Exception = Java.use("java.lang.Exception");

  System.exit.implementation = function() {
      // console.log(Log.getStackTraceString(Exception.$new()));
  };
});
"""
device = frida.get_usb_device()
pid = device.spawn(["com.example.test"])
session = device.attach(pid)
script = session.create_script(ss)
script.load()
device.resume(pid)
sys.stdin.read()
```



3.
Hands on Labs

Let’s start instrumenting Android and iOS Apps using Frida

3. Hands on Labs

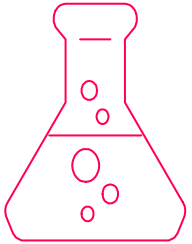
- Root Bypass
- Secret String
- FridaLab
- r2frida - In-Memory Search
- r2frida - Runtime RE

- Login Bypass
- Jailbreak Bypass

Instrumentation Android Applications

3. Hands on Labs

Lab 1: Uncrackable App Level 1

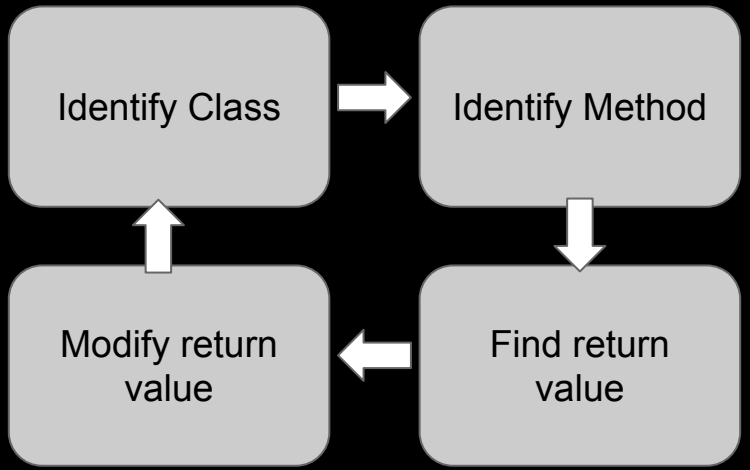


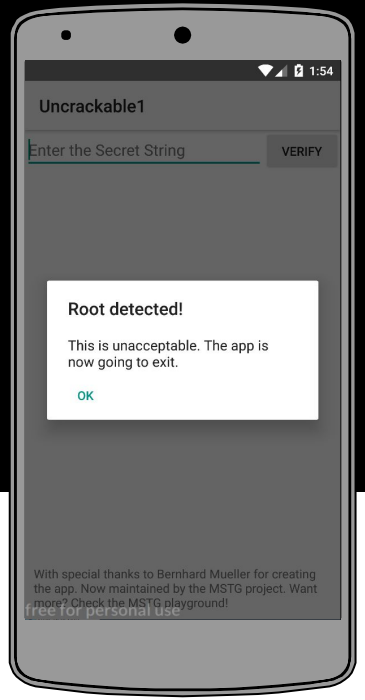
OWASP mobile application

https://github.com/OWASP/owasp-mstg/tree/master/Crackmes/Android/Level_01

3. Hands on Labs

Lab 1: Uncrackable Level 1 – Root Bypass



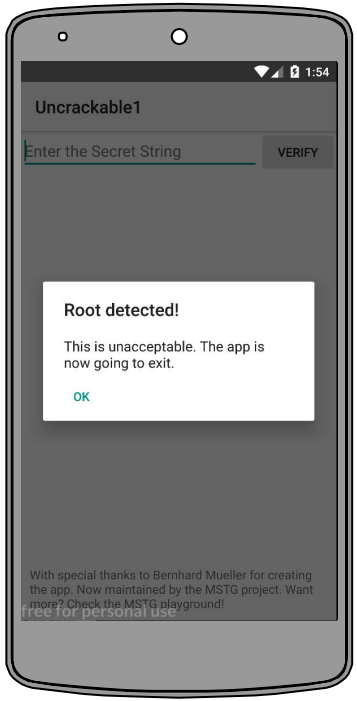


3. Hands on Labs

Lab 1: Uncrackable Level 1 – Root Bypass

- What does the application do?

1. Detect root on the device
2. Appears a pop-up with a message and an “OK” button
3. If you click on OK, the application closes.

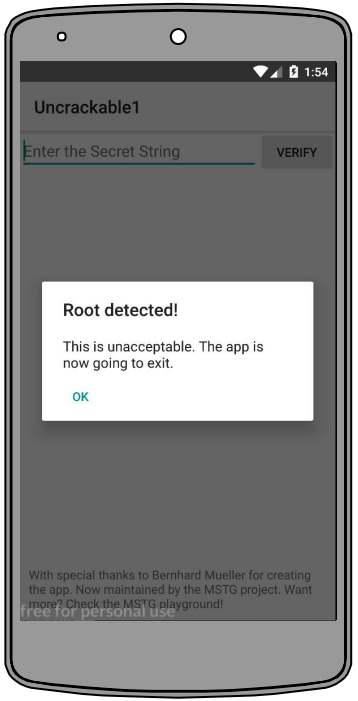


3. Hands on Labs

Lab 1: Uncrackable Level 1 – Root Bypass

- Find the function that checks if the device is rooted:

```
/* access modifiers changed from: protected */
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);
}
```



3. Hands on Labs

Lab 1: Uncrackable Level 1 – Root Bypass

- Any Ideas?

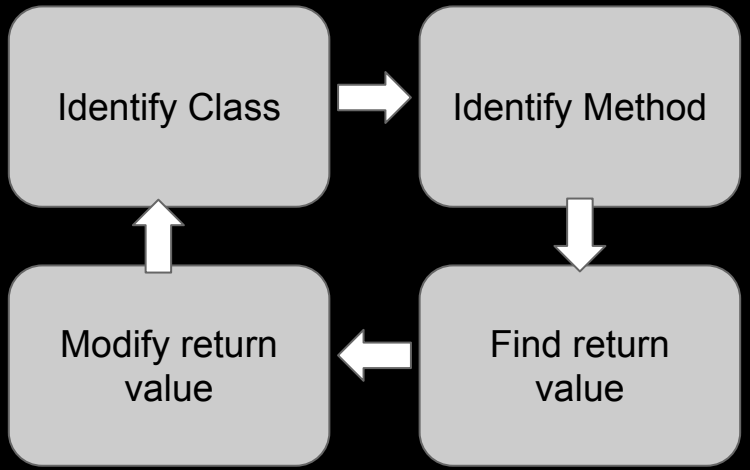
➔ Hook the functions that checks the root (c.a(), c.b(), c.c() and b.a()) and return false.

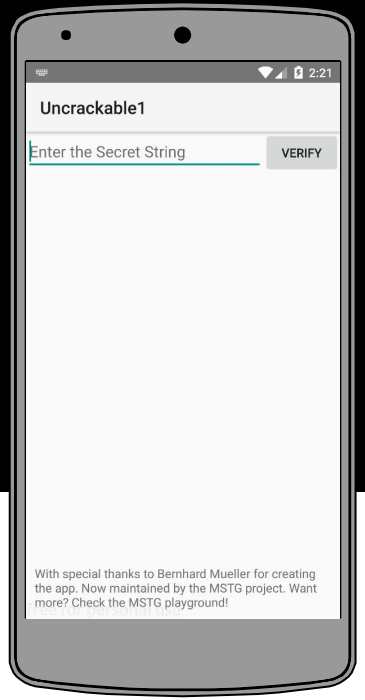
➔ Hook the function a() and change implementation

➔ **Change the exit() function implementation to change the operation when click on "OK".**
  - **Create a .js**
  - **$ frida -l uncrackable_root.js -U owasp.mstg.uncrackable1**

3. Hands on Labs

Lab 1: Uncrackable Level 1 – Secret String



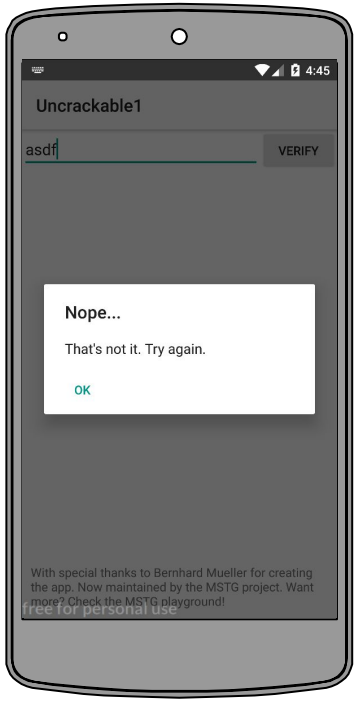


3. Hands on Labs

Lab 1: Uncrackable Level 1 – Secret String

- What does the application do?

1. We enter any string of characters
2. click on "Verify"
3. A message appears indicating that it is not the expected string **"Nope... That's not it. Try again."**



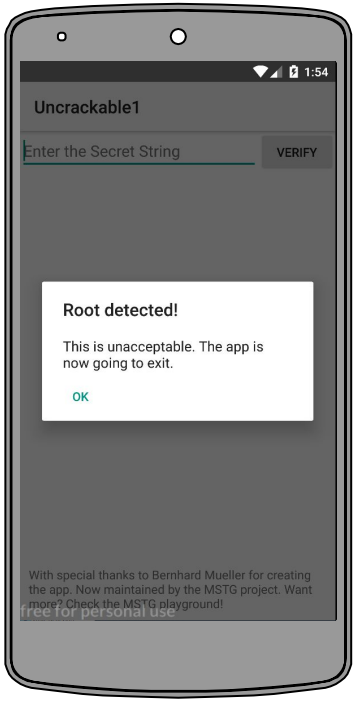
3. Hands on Labs

Lab 1: Uncrackable Level 1 – Secret String

- Find the function to discover the secret string

sg.vantagepoint.a.a

```
1  package sg.vantagepoint.a;
2
3  import javax.crypto.Cipher;
4  import javax.crypto.spec.SecretKeySpec;
5
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 }
```



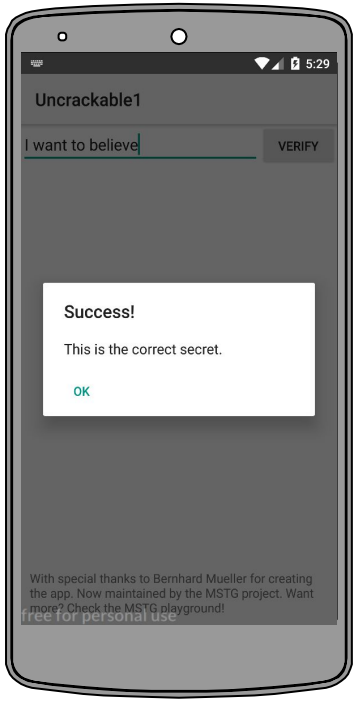
3. Hands on Labs

Lab 1: Uncrackable Level 1 – Secret String

- Find the function to discover the secret string

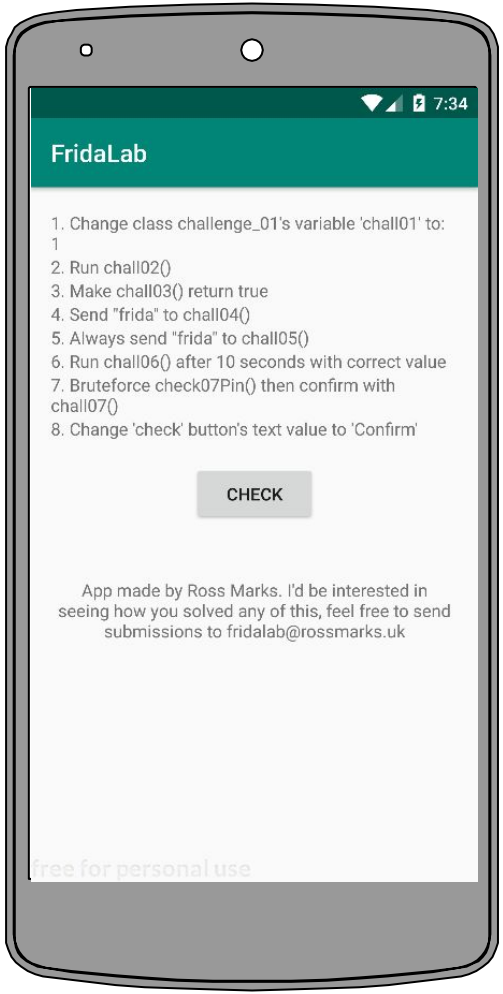
  ➔ **`$ frida -l uncrackable_secret.js -U owasp.mstg.uncrackable1`**

```
root@kali:~/Desktop/PentestingFrida/Apps/Uncrackable-Level1# frida -l uncrackable_secret.js -U owasp.mstg.uncrackable1
     ____
    / _  |   Frida 12.7.5 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/

[Genymotion Samsung Galaxy S6 2::owasp.mstg.uncrackable1]-> [+] Starting our hooker
[+] Decrypted string :I want to believe
[Genymotion Samsung Galaxy S6 2::owasp.mstg.uncrackable1]->
```



3. Hands on Labs
Lab 2: FridaLab

1. Change class challenge_01's variable to:1
2. Run chall02()
3. Make chall03() return "true"
4. Send "Frida" to chall04()
5. Always send "Frida" to chall05()
6. Run chall06() after 10 seconds with correct value
7. Bruteforce check07Pin() then confirm with chall07()
8. Change 'check' button's text value to 'Confirm'

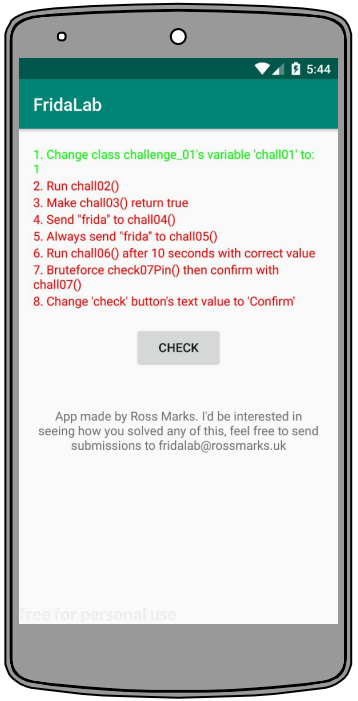


3. Hands on Labs

Lab 2: FridaLab - Challenge 1

- Change class challenge_01's variable to:1

```
package uk.rossmarks.fridalab;

public class challenge_01 {
    static int chall01;

    public static int getChall01Int() {
        return chall01;
    }
}
```

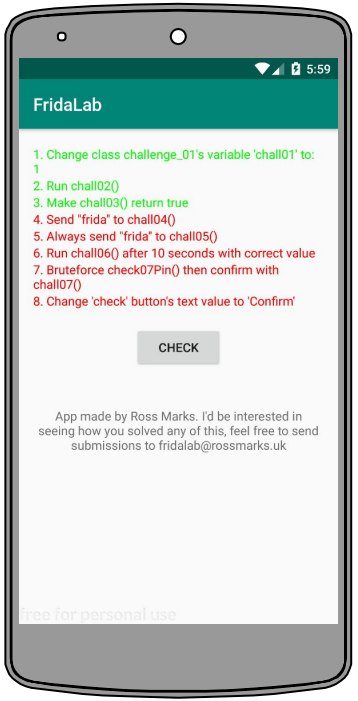


3. Hands on Labs

Lab 2: FridaLab - Challenge 2, 3

- Run chall02()
- Make chall03() return “true”

```
private void chall02() {
    this.completeArr[1] = 1;
}

public boolean chall03() {
    return false;
}
```

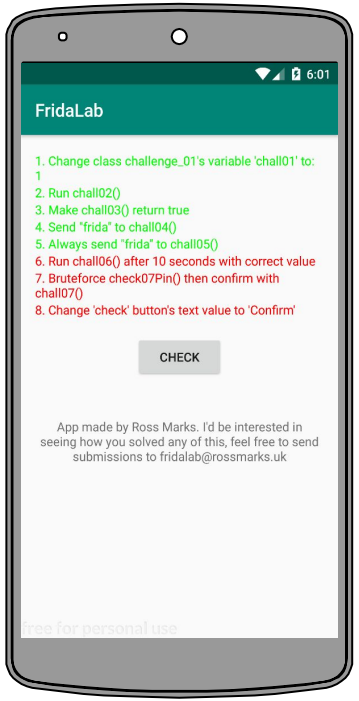


3. Hands on Labs

Lab 2: FridaLab - Challenge 4, 5

- Send “Frida” to chall04()
- Always send “Frida” to chall05()

```
public void chall04(String str) {
    if (str.equals("frida")) {
        this.completeArr[3] = 1;
    }
}

public void chall05(String str) {
    if (str.equals("frida")) {
        this.completeArr[4] = 1;
    } else {
        this.completeArr[4] = 0;
    }
}
```

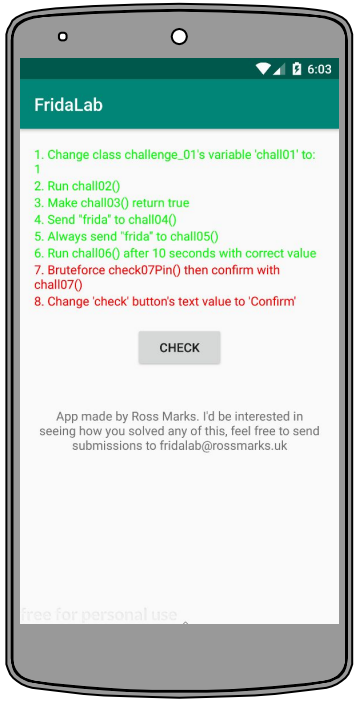


3. Hands on Labs

Lab 2: FridaLab - Challenge 6

- Run chall06() after 10 seconds with correct value

```
package uk.rossmarks.fridalab;

public class challenge_06 {
    static int chall06;
    static long timeStart;

    public static void startTime() {
        timeStart = System.currentTimeMillis();
    }

    public static boolean confirmChall06(int i) {
        return i == chall06 && System.currentTimeMillis() > timeStart + 10000;
    }

    public static void addChall06(int i) {
        chall06 += i;
        if (chall06 > 9000) {
            chall06 = i;
        }
    }
}
```

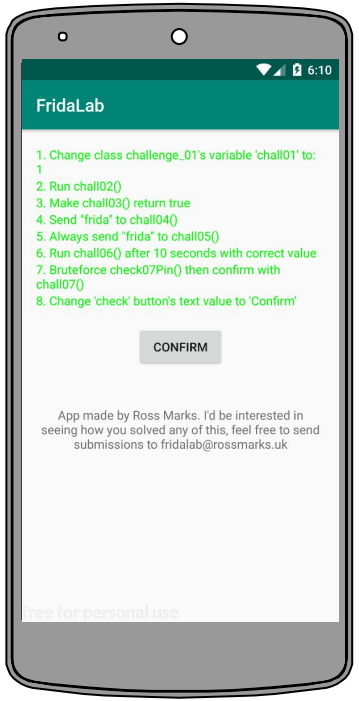


3. Hands on Labs

Lab 2: FridaLab - Challenge 7, 8

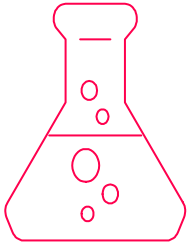
- Bruteforce check07Pin() then confirm with chall07()
- Change ‘check’ button’s text value to ‘Confirm’

```
package uk.rossmarks.fridalab;

public class challenge_07 {
    static String chall07;

    public static void setChall07() {
        chall07 = BuildConfig.FLAVOR + (((int) (Math.random() * 9000.0d)) + 1000);
    }

    public static boolean check07Pin(String str) {
        return str.equals(chall07);
    }
}


public boolean chall08() {
    return ((String) ((Button) findViewById(R.id.check)).getText()).equals("Confirm");
}
```



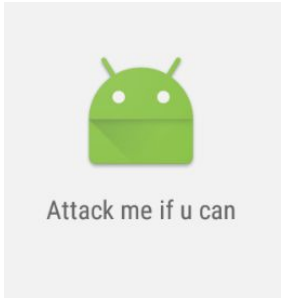
3. Hands on Labs

Lab 3: MSTG Hacking Playground



MSTG Playground for Android

https://github.com/OWASP/MSTG-Hacking-Playground/tree/master/Android

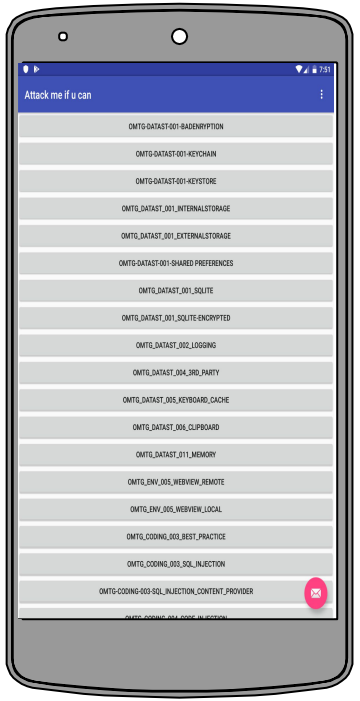


3. Hands on Labs

Lab 3: MSTG Hacking Playground

- Open the MSTG Hacking Playground app.
- List running applications.

```
$ frida-ps -Ua

  PID  Name                           Identifier

----  -----------------------------  -----------------

  650  Android Keyboard (AOSP)        com.android.inputmethod.latin

  952  Android Services Library       com.google.android.ext.services

  559  Android System                 android

 7732  Attack me if u can             sg.vp.owasp_mobile.omtg_android
```



3. Hands on Labs

Lab 3: r2frida - In-memory search

- Open a session with r2frida:

```
$ r2 frida://usb//sg.vp.owasp_mobile.omtg_android
 -- Hang in there, Baby!
[0x00000000]>
```

- See all options with **`r2 frida://?`**

```
$ r2 frida://?
r2 frida://[action]/[target]
* target = process-id | process-name | app-name
* program = find-in-path | absolute-path
* device = device-id | ''
* peer = ip-address:port            # no hostname resolution
Long URIs: (new)
* frida://spawn/$(program)          # start a new process
* frida://attach/(target)           # attach to current
* frida://usb/$(device)/$(target)   # connect to USB device
* frida://connect/$(peer)/$(target)  # connect to remote frida-server
* frida://spawn/usb/$(device)/$(program)
* frida://attach/usb/$(device)/$(target)
```

3. Hands on Labs

Lab 3: r2frida - In-memory search

- Display the app binary information by using **\i**

```
[0x00000000]> \i
arch                x86
bits                32
os                  linux
pid                 9239
uid                 10078
objc                false
runtime             V8
java                true
cylang              false
pageSize            4096
pointerSize         4
codeSigningPolicy   optional
isDebuggerAttached  false
cwd                 /
dataDir             /data/user/0/sg.vp.owasp_mobile.omtg_android
codeCacheDir        /data/user/0/sg.vp.owasp_mobile.omtg_android/code_cache
extCacheDir         /storage/emulated/0/Android/data/sg.vp.owasp_mobile.omtg_android/cache
obbDir              /storage/emulated/0/Android/obb/sg.vp.owasp_mobile.omtg_android
filesDir            /data/user/0/sg.vp.owasp_mobile.omtg_android/files
noBackupDir         /data/user/0/sg.vp.owasp_mobile.omtg_android/no_backup
codePath            /data/app/sg.vp.owasp_mobile.omtg_android-1/base.apk
packageName         sg.vp.owasp_mobile.omtg_android
androidId           30ea2a28fc31f3cd
cacheDir            /data/local/tmp
jniEnv              0xe8751510
```

\il : Modules (binaries and libraries) that the app has loaded

\is <lib> : Search all symbols of a certain module

\ii <lib> : List the imports

\iE <lib> : List the exports

\ic~com.android.class : Look at are the currently loaded Java classes

\ic com.android.class.MainActivity~com.android.class : List class fields

\icL : display information about the class loader

3. Hands on Labs

Lab 3: r2frida - In-memory search

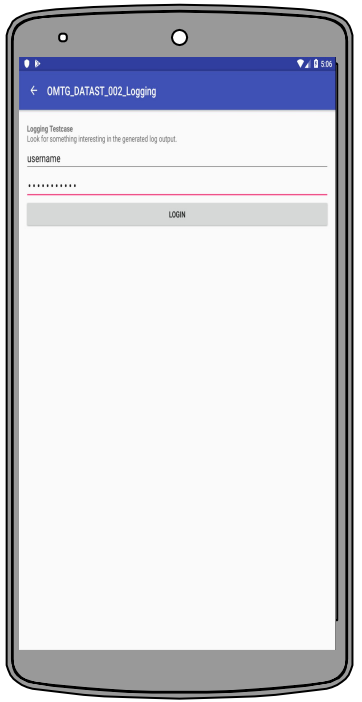
- Retrieve the app's memory maps directly belongs to the app: **\dm~<package_name>**

```
[0x00000000]> \dm~sg.vp
0xd46ab000 - 0xd4a74000 r-- /data/app/sg.vp.owasp_mobile.omtg_android-1/oat/x86/base.odex
0xd4a74000 - 0xd4aeb000 r-x /data/app/sg.vp.owasp_mobile.omtg_android-1/oat/x86/base.odex
0xd4b38000 - 0xd4b39000 r-- /data/app/sg.vp.owasp_mobile.omtg_android-1/oat/x86/base.odex
0xd4b39000 - 0xd4b3a000 rw- /data/app/sg.vp.owasp_mobile.omtg_android-1/oat/x86/base.odex
0xea3e5000 - 0xea424000 r-- /data/app/sg.vp.owasp_mobile.omtg_android-1/base.apk
0xea746000 - 0xea750000 r-- /data/app/sg.vp.owasp_mobile.omtg_android-1/base.apk
```

3. Hands on Labs

Lab 3: r2frida - In-memory search

- Navigate to "OMTG_DATAST_002_LOGGING" and enter a string to the password field, but do not click on Login just yet.



3. Hands on Labs

Lab 3: r2frida - In-memory search

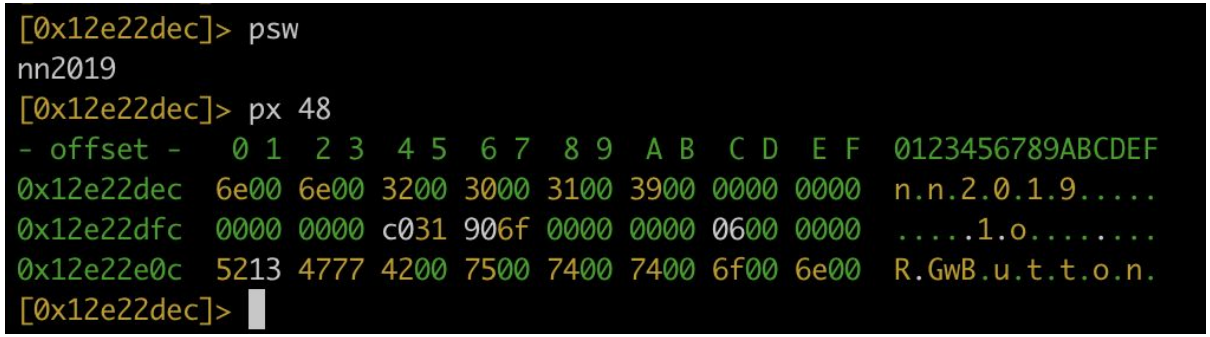
- Search for occurrences of the wide version of the string **`\/w <string>`**

```
[0x00000000]> \/w nn2019
Searching 12 bytes: 6e 00 6e 00 32 00 30 00 31 00 39 00
Searching 12 bytes in [0x12c00000-0x12e17000]
Searching 12 bytes in [0x12e17000-0x12e34000]
hits: 9
0x12c3f934 hit3_0 6e006e003200300031003900
0x12c40e50 hit3_1 6e006e003200300031003900
0x12c40ed0 hit3_2 6e006e003200300031003900
0x12c541b0 hit3_3 6e006e003200300031003900
0x12d0d73c hit3_4 6e006e003200300031003900
0x12d0d91c hit3_5 6e006e003200300031003900
0x12e22c0c hit3_6 6e006e003200300031003900
0x12e22dec hit3_7 6e006e003200300031003900
0xde522850 hit3_8 6e006e003200300031003900
```

3. Hands on Labs

Lab 3: r2frida - In-memory search

- Seek to its address using **`s <address>`** or **`s hitX_X`**, print it using **`psw`** (print string wide) or use **`px`** to print its raw hexadecimal values.



You may check if you still can find those strings in memory after the login is completed to verify if this sensitive data is wiped from memory after its use.

3. Hands on Labs

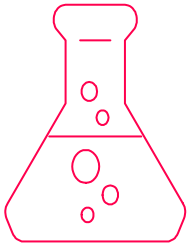
Lab 3: r2frida

- Retrieve the strings from a certain binary and filter them. Run the **`\iz`** command. Apply a filter with a keyword **`~<keyword>/~+<keyword>`** (to minimize the terminal output).

```
[0xd46ab000]> \il~base.odex
0xd46ab000 base.odex
[0xd46ab000]> s 0xd46ab000
[0xd46ab000]> \iz~+login
Reading 3.78515625MB ...
0xd46d14dc  "A401: Login validation error on server - request will be discarded"
0xd46dce94  "Creating DialogIntent for "
0xd46e6060  "2Landroid/content/DialogInterface$OnCancelListener;"
0xd46e6094  "1Landroid/content/DialogInterface$OnClickListener;"
0xd46e60c7  "3Landroid/content/DialogInterface$OnDismissListener;"
0xd46e60fc  "/Landroid/content/DialogInterface$OnKeyListener;"
0xd46e612d  "<Landroid/content/DialogInterface$OnMultiChoiceClickListener;"
0xd46e616b  "!Landroid/content/DialogInterface;"
0xd4743189  "*button_OMTG_CODING_003_SQL_Injection_Login"
0xd4743351  "button_OMTG_DATAST_002_Login"
0xd47483f3  "crashReportDialogIntent"
0xd474866b  "createCrashReportDialogIntent"
0xd474fd50  "formUriBasicAuthLogin"
0xd4778dc1  "setFormUriBasicAuthLogin"
```

3. Hands on Labs

Lab 4: UnCrackable App Level 2



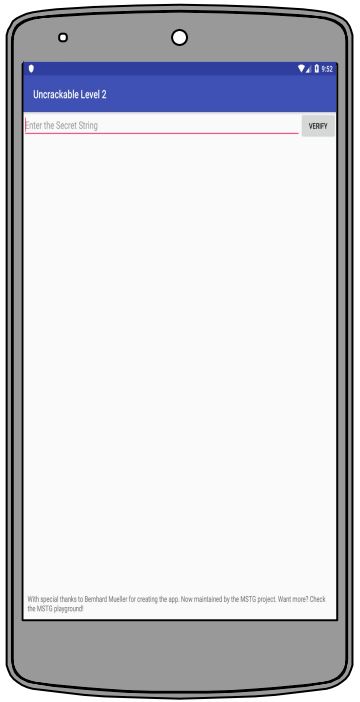
UnCrackable App for Android Level 2

https://github.com/OWASP/owasp-mstg/tree/master/Crackmes/Android/Level_02/

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- A secret string is hidden somewhere in this app. Find a way to extract it.
- Anti-rooting checks are in place at the Java level. We do not need to bypass it.



3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Write a script called bypassroot_uncrackable2.js containing following lines:

```
Java.perform(function () {

  var sysexit = Java.use("java.lang.System");

  sysexit.exit.overload("int").implementation = function(var_0) {

     console.log("java.lang.System.exit(I)V  // We avoid exiting the
application  :)");

  };

});
```

https://mega.nz/#!RhBnkKJQ!uASHVaZKyy4_fMRSceHspByMQSoKa_G9mZIXACnHOdc

3. Hands on Labs

Lab 4: Decompilation of the APK

- Unzip de APK and decompile the classes.

```
$ apkx UnCrackable-Level2.apk

Extracting UnCrackable-Level2.apk to UnCrackable-Level2

Converting: classes.dex -> classes.jar (dex2jar)

dex2jar UnCrackable-Level2/classes.dex ->
UnCrackable-Level2/classes.jar

Decompiling to UnCrackable-Level2/src (cfr)
```

3. Hands on Labs

Lab 4: Decompilation of the APK

- In MainActivity.java there is a static block with a call to System.load that loads library foo.

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

- Declare a native method part of foo and calls this.init() inside of onCreate() method.

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

3. Hands on Labs

Lab 4: Decompilation of the APK

- In MainActivity.java our text field (“secret string”) is checked in this.m.a(String(object)) method.

```
this.m = new CodeCheck();
super.onCreate(bundle);
this.setContentView(2131296283);
```

```
public void verify(View object) {
    object = ((EditText)this.findViewById(2131165237)).getText().toString();
    AlertDialog alertDialog = new AlertDialog.Builder((Context)this).create();
    if (this.m.a((String)object)) {
        alertDialog.setTitle((CharSequence)"Success!");
        object = "This is the correct secret.";
    } else {
        alertDialog.setTitle((CharSequence)"Nope...");
        object = "That's not it. Try again.";
    }
```

3. Hands on Labs

Lab 4: Decompilation of the APK

- In CodeCheck.java the input of our text field gets passed to a native function called bar. Bar function is in the libfoo.so library.

```
public class CodeCheck {
    private native boolean bar(byte[] var1);

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

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Open bar function in libfoo.so library with **`r2 UnCrackable-Level2/lib/x86/libfoo.so`**
- Analyze all **aaa** and find library exports **iE**.

```
$ r2 UnCrackable-Level2/lib/x86/libfoo.so
 -- radare2 is like windows 7 but even better.
[0x00000600]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x00000600]> iE
[Exports]
Num Paddr      Vaddr      Bind     Type Size Name
007 0x00000f60 0x00000f60 GLOBAL   FUNC  199 Java_sg_vantagepoint_uncrackable2_CodeCheck_bar
009 0x00000f30 0x00000f30 GLOBAL   FUNC   40 Java_sg_vantagepoint_uncrackable2_MainActivity_init
014 ---------- 0x00004004 GLOBAL NOTYPE    0 __bss_start
015 ---------- 0x00004009 GLOBAL NOTYPE    0 _end
016 ---------- 0x00004004 GLOBAL NOTYPE    0 _edata
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Seek to CodeCheck_bar address **`s <address>`** and decompile function with Ghidra plugin **pdg**.

```
[0x00000600]> s 0x00000f60
[0x00000f60]> pdg

// WARNING: Variable defined which should be unmapped: var_ch

undefined4
sym.Java_sg_vantagepoint_uncrackable2_CodeCheck_bar(int32_t arg_8h, undefined4 placeholder_1, int
32_t arg_10h)
{
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

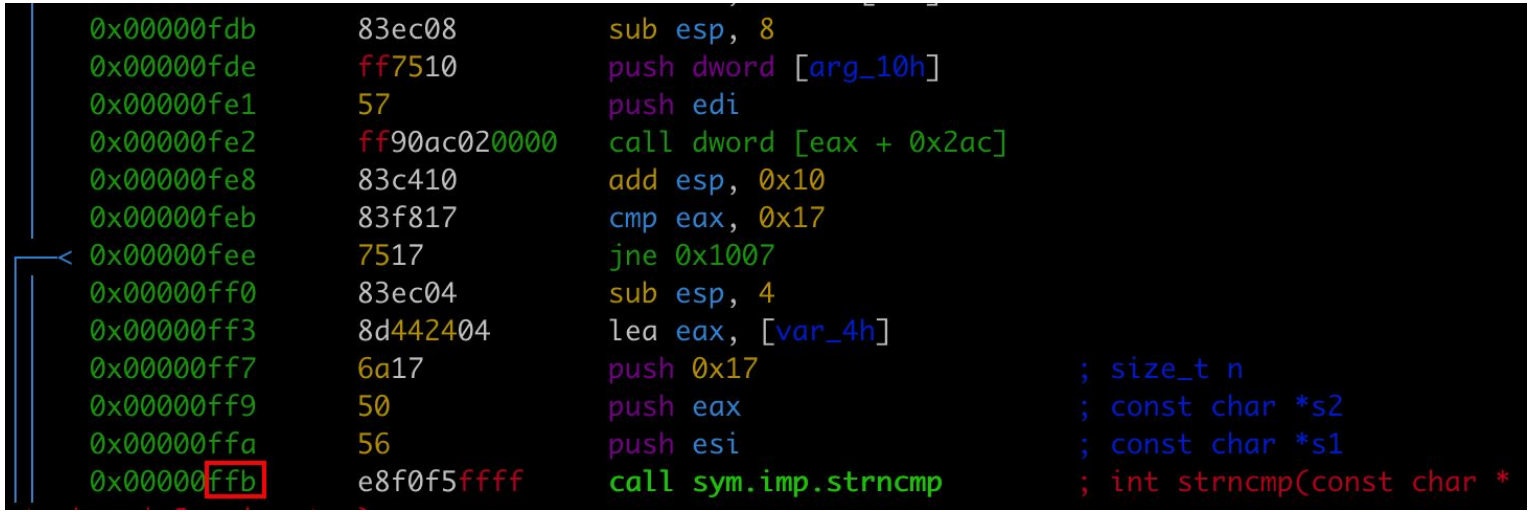
- The pseudocode shows that the native CodeCheck_bar function will return 1 if the input string length (iVar2) is 0x17, which is 23 in decimal notation, and the strncmp returns 0.

```
int32_t var_ch;

iStack24 = *(int32_t *)(in_GS_OFFSET + 0x14);
if (cRam00004008 == '\x01') {
    uStack48 = 0x6e616854;
    uStack44 = 0x6620736b;
    uStack40 = 0x6120726f;
    uStack36 = 0x74206c6c;
    uStack32 = 0x6568;
    uStack30 = 0x73696620;
    uStack26 = 0x68;
    uVar1 = (**(code **)(*(int32_t *)arg_8h + 0x2e0))(arg_8h, arg_10h, 0);
    iVar2 = (**(code **)(*(int32_t *)arg_8h + 0x2ac))(arg_8h, arg_10h);
    if (iVar2 == 0x17) {
        iVar2 = sym.imp.strncmp(uVar1, &uStack48, 0x17);
        if (iVar2 == 0) {
            uVar1 = 1;
            goto code_r0x00001009;
        }
    }
}
uVar1 = 0;
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Print disassemble function **pdf** to identify the offset of the instruction.



3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Spawn uncrackable2 application using r2frida **`$ r2 frida://spawn/usb//<app>`** , load the bypass_root script with **`\. script.js`** and resumed spawned process **`\dc`**

```
$ r2 frida://spawn/usb//owasp.mstg.uncrackable2
 -- This page intentionally left blank.
[0x00000000]> \. /Frida_Taller/bypassroot_uncrackable2_1.js
[0x00000000]> \dc
resumed spawned process.
[0x00000000]> java.lang.System.exit(I)V  // We avoid exiting the application  :)
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Notice that the library exports 2 interesting functions using **\iE <lib>** and imports all the dynamic export **.\iE* <lib>** data from Frida and all the dynamic import data **.\ii* <lib>**

```
[0x00000000]> \iE libfoo.so
0xe79c0f60 f Java_sg_vantagepoint_uncrackable2_CodeCheck_bar
0xe79c0f30 f Java_sg_vantagepoint_uncrackable2_MainActivity_init
[0x00000000]> .\iE* libfoo.so
[0x00000000]> .\ii* libfoo.so
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Seek to this function in the libfoo.so library **`s <address>`**
- Analize function bar **`af`** and decompile current function with the Ghidra plugin **`pdg`**

```
[0x00000000]> s 0xe79c0f60
[0xe79c0f60]> af
[0xe79c0f60]> pdg

// WARNING: Control flow encountered bad instruction data
// WARNING: Instruction at (ram,0xe79c1040) overlaps instruction at (ram,0xe79c103f)
//
// WARNING: Variable defined which should be unmapped: var_ch

undefined4 __cdecl
sym.fun.Java_sg_vantagepoint_uncrackable2_CodeCheck_bar(int32_t arg_8h, undefined4 pl
{
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Print disassemble function **pdf** to identify the address to trace.

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

The static way:

- We need **addr_calculated_at_runtime = base_addr_libfoo + offset**.
- Base address : Inspect memory maps of libfoo **\dm~libfoo**

```
[0xe79c0f60]> \dm~libfoo
0xe79c0000 - 0xe79c3000 r-x /data/app/owasp.mstg.uncrackable2-1/lib/x86/libfoo.so
0xe79c3000 - 0xe79c4000 r-- /data/app/owasp.mstg.uncrackable2-1/lib/x86/libfoo.so
0xe79c4000 - 0xe79c5000 rw- /data/app/owasp.mstg.uncrackable2-1/lib/x86/libfoo.so
```

- Offset of the function:

```
0x00000ff7      6a17            push 0x17                       ; size_t n
0x00000ff9      50              push eax                        ; const char *s2
0x00000ffa      56              push esi                        ; const char *s1
0x00000ffb      e8f0f5ffff      call sym.imp.strncmp            ; int strncmp(const
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

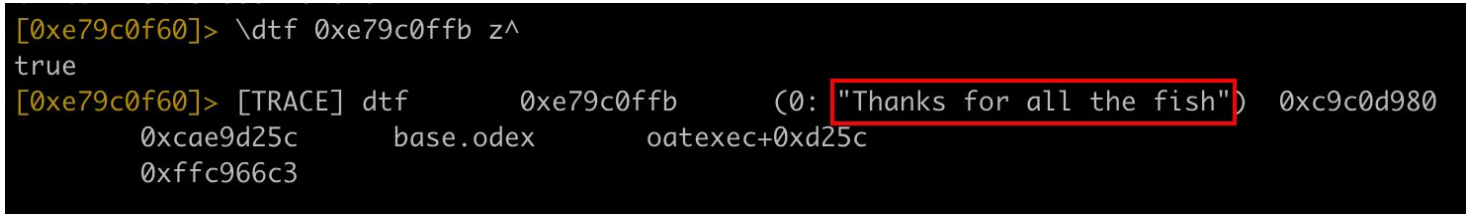
- The address of the function to be traced can be calculated:

```
[0xe79c0f60]> ? 0xe79c0000 + 0xffb
int32   -409202693
uint32  3885764603
hex     0xe79c0ffb
octal   034747007773
unit    3.6G
segment e79c000:0ffb
string  "\xfb\x0f\x9c\xe7"
fvalue: 3885764603.0
float:  -14739679178897476330455O4.000000f
double: 0.000000
binary  0b11100111100111000000111111111011
trits   0t101000210202021221112
```

3. Hands on Labs

Lab 4: r2frida - Runtime Reverse Engineering

- Let's start tracing using **`\dtf <address> z^`**
  - ^ = trace onEnter instead of onExit
  - z = show pointer to string
- This traces the function when input a 23 string long (secret string) and prints the flag.



Instrumentation iOS Applications

3. Hands on Labs
IPA Binary RE

IPA binary - Reverse Engineering phase:

- Classdump or Frida (only return Class names and methods)
- Hopper: trial version (semi-functional)
- IDA: more powerful

- Hopper and IDA generate pseudocode

3. Hands on Labs
Objective C

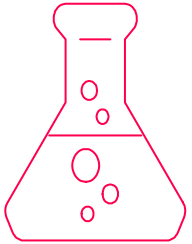
- Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language.
- The Objective-C model of object-oriented programming is based on message passing to object instances. In Objective-C one does not call a method; one sends a message.
    - C++:

      obj->method(parameter);
    - Objective C:

      [obj method:parameter];

3. Hands on Labs

Lab 5: Damn Vulnerable iOS Application

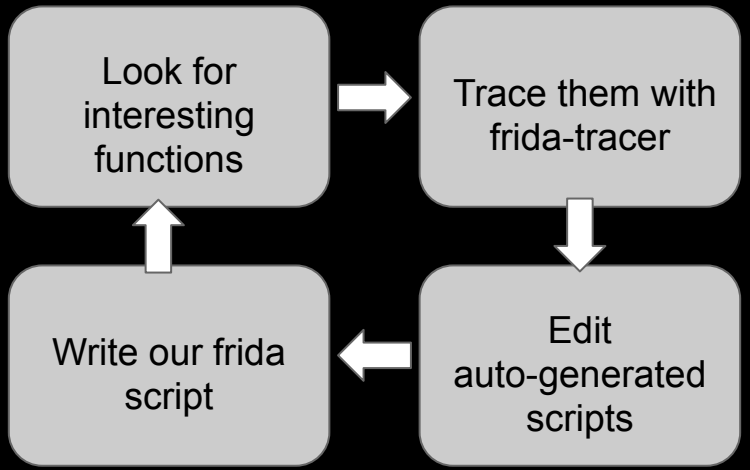


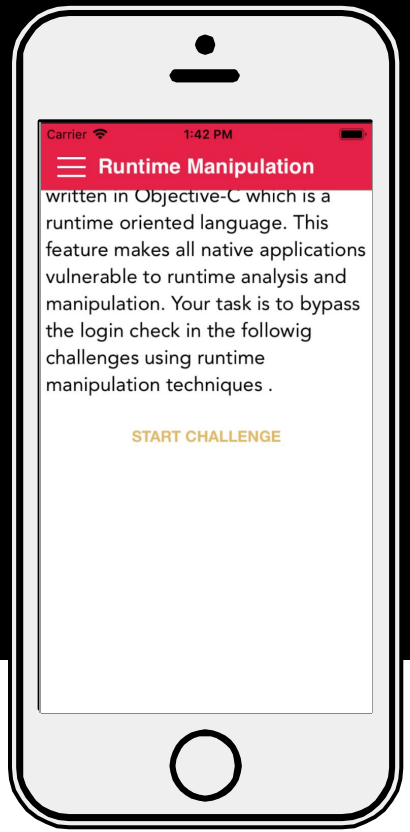
IPA/Project: http://damnvulnerableiosapp.com/

- Bypass Login Validate
- Bypass Jailbreak detection with Frida

3. Hands on Labs

Lab5: Bypass Login Validate

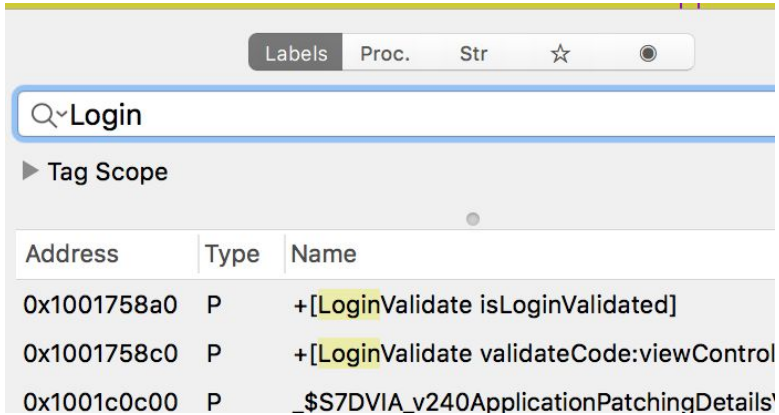




3. Hands on Labs

Lab5: Bypass Login Validate

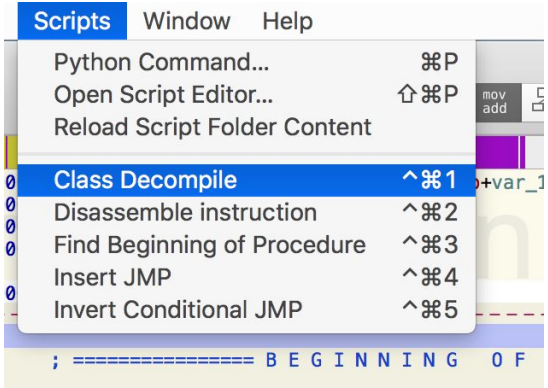
- Open /Users/tallerfrida/Master_iOS/DVIA-v2-master/DVIA-v2.xcworkspace with Xcode.
- Run DVIA-v2 on Simulator in Xcode.
- Load /Users/tallerfrida/Master_iOS/DVIA/DVIA-v2 binary (decrypted) into Hopper and wait for the analysis to be completed.
- Look for the "login" string in the search box.

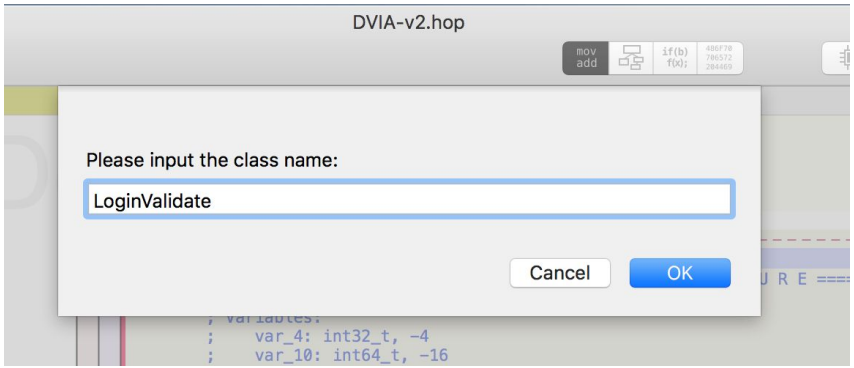


3. Hands on Labs

Lab5: Bypass Login Validate

- Hopper scripts placed in ~/Library/Application Support/Hopper/Scripts directory.
- Click the menu button Scripts -> Class Decompile. Decompile the functions to see what they are doing and what they return.





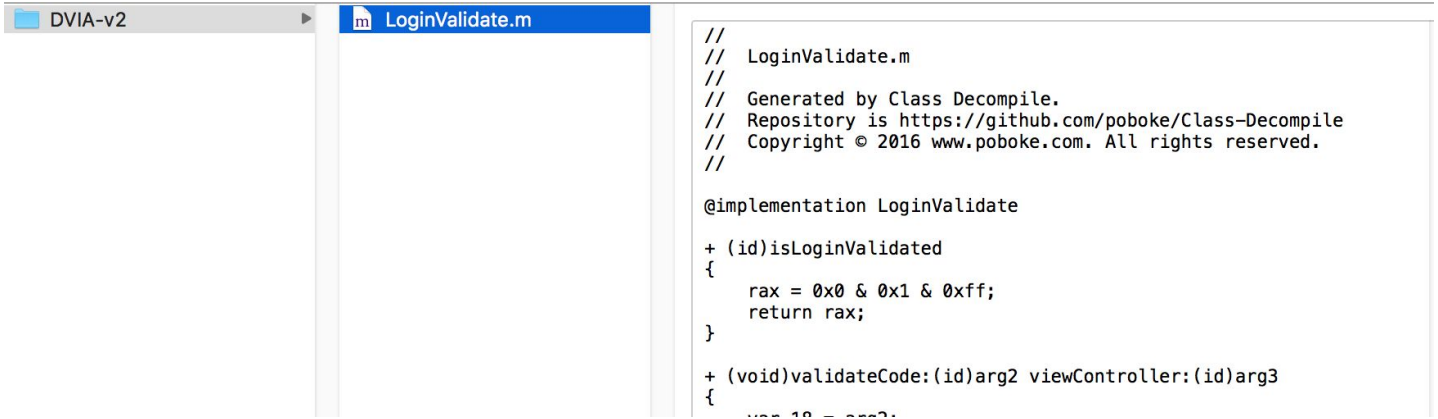
https://github.com/poboke/Class-Decompile

https://github.com/phracker/HopperScripts

3. Hands on Labs

Lab5: Bypass Login Validate

- The decompiled pseudo-code stored in the ~/ClassDecompiles directory.

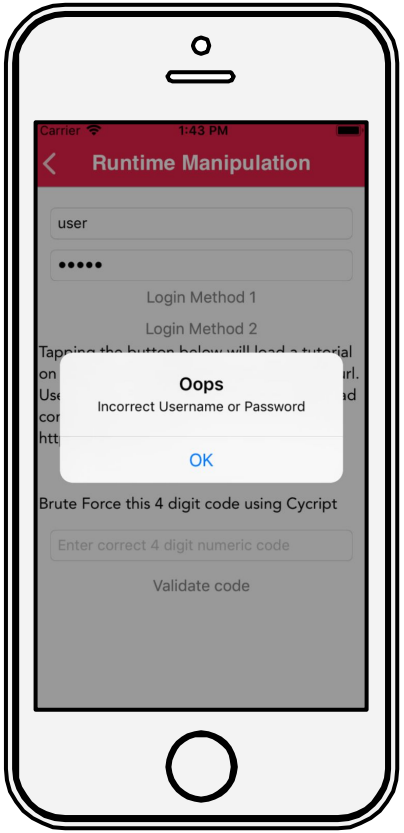


3. Hands on Labs

Lab5: Bypass Login Validate

- Trace calls to +[LoginValidate isLoginValidated], and create a JavaScript hook with the onEnter and onLeave callback functions.

```
$ frida-trace -R -f re.frida.Gadget -m "+[LoginValidate isLoginValidated]"
Instrumenting functions...
+[LoginValidate isLoginValidated]: Auto-generated handler at
"/Users/taller/Taller/__handlers__/__LoginValidate_isLoginValidated_.js"
Started tracing 1 function. Press Ctrl+C to stop.
            /* TID 0x4c03 */
 56167 ms  +[LoginValidate isLoginValidated]
```



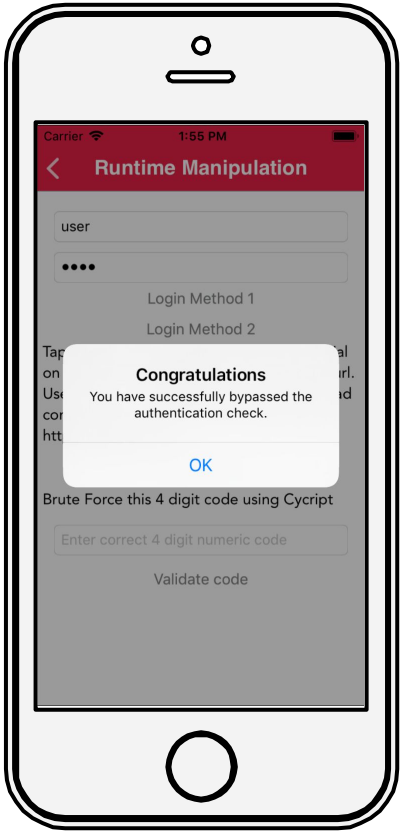
3. Hands on Labs

Lab5: Bypass Login Validate

- Edit JavaScript hook and replace the return value.

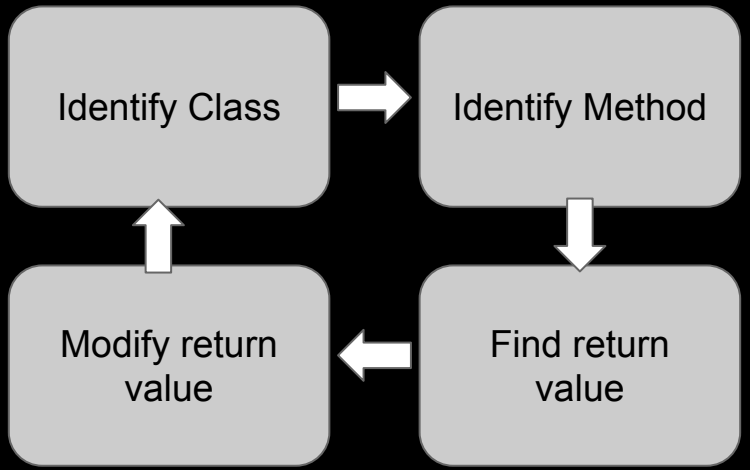
```
onLeave: function (log, retval, state) {
  console.log("Function isLogin Validated originally return: " + retval)
  retval.replace(1);
  console.log("Changing the return value to: " + retval)
} }
```

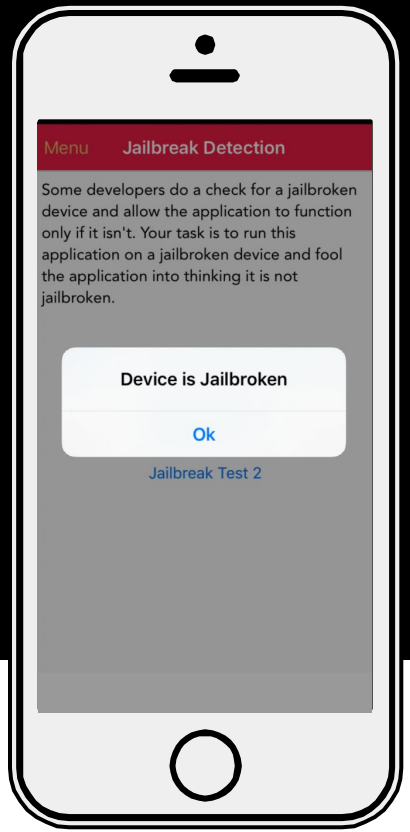
```
$ frida-trace -R -f re.frida.Gadget -m "+[LoginValidate isLoginValidated]"
Instrumenting functions...
+[LoginValidate isLoginValidated]: Loaded handler at
"/Users/taller/Taller/__handlers__/__LoginValidate_isLoginValidated_.js"
Started tracing 1 function. Press Ctrl+C to stop.
Function isLogin Validated originally return: 0x0
Changing the return value to: 0x1
            /* TID 0x2303 */
171096 ms  +[LoginValidate isLoginValidated]
```



3. Hands on Labs

Lab5: Bypass Jailbreak Detection





3. Hands on Labs

Lab5: Bypass Jailbreak Detection

- Find the class name which implements the Jailbreak Detection method.

```
$ more /Users/taller/Taller/frida-scripts/iOS/find-classes.js
console.log("[*] Started: Find Classes")
if (ObjC.available)
{
	for (var className in ObjC.classes)
	{
	if (ObjC.classes.hasOwnProperty(className))
	{
		console.log(className);
	}
```

```
$ frida -R -f re.frida.Gadget -l /Users/taller/Taller/frida-scripts/iOS/find-classes.js| grep -i jail

JailbreakDetection
DVIA_v2.JailbreakDetectionViewController
```

3. Hands on Labs

Lab5: Bypass Jailbreak Detection

- Modify the class name in "show-all-methods-of-specific-class.js" as shown below to find all the methods.

```
 console.log("[*] Started: Find All Methods of a Specific Class");
if (ObjC.available)
{
	try
	{
	//Your class name here
	var className = "JailbreakDetection";
	var methods = eval('ObjC.classes.' + className + '.$methods');
	for (var i = 0; i < methods.length; i++)
	{
		try
		{
		console.log("[-] "+methods[i]);
		}
```

3. Hands on Labs

Lab5: Bypass Jailbreak Detection

- Find the method name which detects the Jailbreak Detection

```
 $ frida -R -f re.frida.Gadget -l
/Users/taller/Taller/frida-scripts/iOS/show-all-methods-of-specific-class.js

      ____
     / _  |   Frida 12.2.24 - A world-class dynamic instrumentation toolkit
   | (_| |
     > _  |   Commands:
   /_/ |_|     help   -> Displays the help system
   . . . .     object?   -> Display information about 'object'
   . . . .     exit/quit -> Exit
   . . . .
   . . . .     More info at http://www.frida.re/docs/home/
Spawning `re.frida.Gadget`...
[*] Started: Find All Methods of a Specific Class
[-] + isJailbroken
```

3. Hands on Labs

Lab5: Bypass Jailbreak Detection

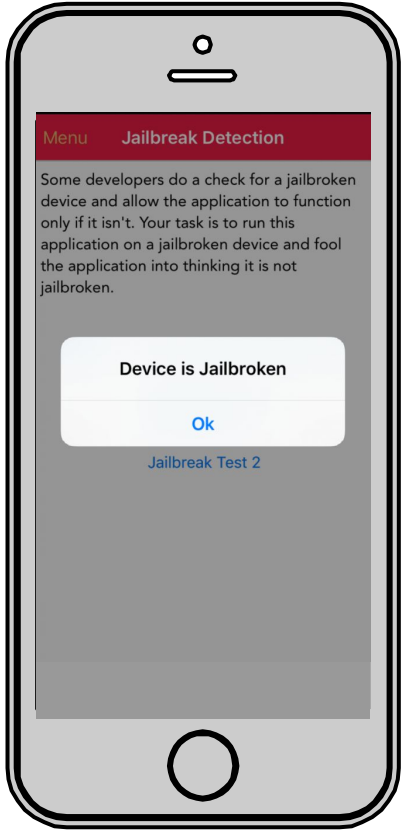
- Modify the class name and method name in 'hook-specific-method-of-class.js' file.

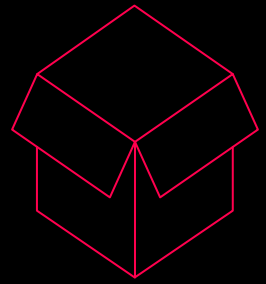
```
//Your class name here
var className = "JailbreakDetection";
//Your function name here
var funcName = "isJailbroken";
var hook = eval('ObjC.classes.' + className + '["' + funcName + '"]');
Interceptor.attach(hook.implementation, {
onLeave: function(retval) {
       console.log("[*] Class Name: " + className);
       console.log("[*] Method Name: " + funcName);
console.log("\t[-] Return Value: " + retval);

//For modifying the return value
       newretval = ptr("0x1") //your new return value here
       retval.replace(newretval)
       console.log("\t[-] New Return Value: " + newretval)
}
```

3. Hands on Labs

Lab5: Bypass Jailbreak Detection

```
$ frida -R -f re.frida.Gadget -l
/Users/taller/Taller/frida-scripts/iOS/bypass-jailbreak-detection.js

     ____
    / _  |   Frida 12.2.24 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|    help   -> Displays the help system
   . . . .    object?   -> Display information about 'object'
   . . . .    exit/quit -> Exit
   . . . .
   . . . .    More info at http://www.frida.re/docs/home/
 Spawned `re.frida.Gadget`. Use %resume to let the main thread start executing!
 [Remote::re.frida.Gadget]-> %resume

 [Remote::re.frida.Gadget]-> [*] Class Name: JailbreakDetection
 [*] Method Name: isJailbroken
    [-] Return Value: 0x0
    [-] New Return Value: 0x1
 Server terminated
```





4.
Tools based on Frida

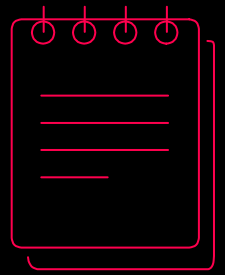
Let’s start with the first set of slides

4. Tools based on Frida

- **brida** - Bridge between Burp Suite and Frida
- **objection** - Runtime Mobile Exploration for iOS and Android
- **Dexcalibur** - A dynamic binary instrumentation tool designed for Android apps and powered by Frida
- **passionfruit** - iOS App Analyzer with Web UI
- **Frida-dump** - a memory dumping tool for both Android and iOS.
- **r2frida-wiki** - Unofficial wiki that provides practical examples on how to use r2frida
- **Frida CodeShare project** - collection of ready-to-run Frida scripts

4. Scripts based on Frida

- **https://github.com/as0ler/frida-scripts** - Repository including some useful frida script for iOS Reversing
- **https://github.com/0xdea/frida-scripts** - Instrumentation scripts to facilitate reverse engineering of android and iOS Apps.
- **https://gitlab.com/roxanagogonea/frida-scripts** - Repository including some useful frida scripts for Android
- **https://github.com/iddoeldor/frida-snippets** - Another useful frida snippets repository
- **https://github.com/oleavr/ios-inject-custom** - Use Frida for standalone injection of a custom payload for iOS.



5.
References

Documentation and resources

5.1 References

- https://www.frida.re/docs
- https://codeshare.frida.re
- https://t.me/fridadotre
- https://github.com/dweinstein/awesome-frida
- https://github.com/enovella/r2frida-wiki
- https://github.com/OWASP/owasp-mstg/

5.2 References

- https://github.com/FrenchYeti/dexcalibur
- https://github.com/Hamz-a/frida-android-libbinder
- https://github.com/Areizen/JNI-Frida-Hook
- https://github.com/xiaokanghub/Frida-Android-unpack
- https://github.com/rootbsd/fridump3
- https://github.com/pspace/bsidesmuc
- https://github.com/dineshshetty/FridaLoader
- https://github.com/iddoeldor/frida-snippets
- https://github.com/rubaljain/frida-jb-bypass
- https://github.com/sensepost/objection/
- https://github.com/interference-security/frida-scripts/
- https://www.amanhardikar.com/mindmaps/Practice.html

Credits

Special thanks to @h4ng3r, @4r4nL, @enovella_, @jaimesalasr, @neofito, pwn&swag and supporters.

Thanks!!

Any questions?

You can find me at @lain7z & @martrudix