# FRIDA 101: A Hands-On Look Inside Mobile App Reverse Engineering

NowSecure

# Agenda

- Intros
- Setting up
- Basic Commands
- Hands-on


Brian Lawrence
Director Solutions Engineering
NowSecure
blawrence@nowsecure.com



Tony Ramirez
Senior App Security Analyst
NowSecure
aramirez@nowsecure.com

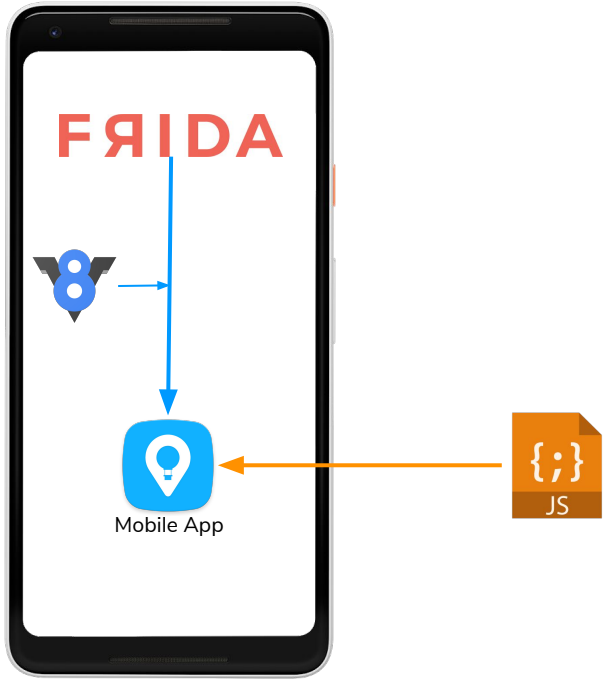
# What's Frida?


- Frida is a dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.

- It allows us to modify the behavior of a target application. It allows us to peek into the inner-workings of the app to further the understanding of how the app works and what it's doing. It's especially powerful for leveraging to understand an app that may have obfuscated source, anti-tamper/anti-debugging protections, properly implemented network comms, etc.

# How does it work?

1. Frida injects Google's V8* engine into a targeted process.
2. JavaScript is then able to run inside the targeted process.
3. Analyst writes JavaScript which is then injected into the targeted process.




**V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and in Node.js, among others.*

NowSecure

# Modes of Operation

- Injected
  - Most common case: spawn an existing program, attach to a running program, or hijack one as it's being spawned, and then run your instrumentation logic inside of it.
- Embedded
  - Think jailed iOS and Android systems. Uses a shared library that you're supposed to embed inside the program that you want to instrument.
- Preloaded

# Frida Tools

**frida-ps** - This is a command-line tool for listing processes, which is very useful when interacting with a remote system.

**frida-trace** is a tool for dynamically tracing function calls.

**frida-discover** is a tool for discovering internal functions in a program, which can then be traced by using frida-trace.

# Using Frida

- Understand what the app is doing without tampering with the code

- We sometimes need to understand these things because common tools we use aren't effective in certain scenarios.

- We need to use a little development know-how to debug


NowSecure

# Getting Set Up!

NowSecure

# Getting Started!

You will need:

- A computer you're able to install development utilities on.
  - Python installed – latest 3.x is highly recommended
- A jailbroken iOS device OR rooted Android device
  - You CAN use Frida without a jailbroken/rooted device using the ***embedded*** mode in Frida, but that is out of scope for this 101 course.

NowSecure

# Installing Frida on your Laptop

```
pip install frida-tools
pip install frida
```

```
pip install frida-tools
pip install frida
```

```
pip install frida-tools
pip install frida
```

NowSecure

# Checking the Version

Make sure you know what version you're running!


NowSecure

# Installing frida-server (Android)

https://github.com/frida/frida/releases

| | |
|---|---|
| frida-server-12.8.20-android-arm.xz | 5.36 MB |
| frida-server-12.8.20-android-arm64.xz | 10.7 MB |
| frida-server-12.8.20-android-x86.xz | 6.39 MB |
| frida-server-12.8.20-android-x86_64.xz | 12.8 MB |
| frida-server-12.8.20-ios-arm.xz | 5.27 MB |
| frida-server-12.8.20-ios-arm64.xz | 10.4 MB |
| frida-server-12.8.20-ios-arm64e.xz | 9.86 MB |
| frida-server-12.8.20-linux-x86.xz | |
| frida-server-12.8.20-linux-x86_64.x | |
| frida-server-12.8.20-macos-x86_6 | |
| frida-server-12.8.20-windows-x86.exe.xz | 10.4 MB |

```
~ adb push frida-server /data/local/tmp
~ adb shell "chmod 755 /data/local/tmp/frida-server"
~ adb shell "/data/local/tmp/frida-server &"
```

NowSecure

# Installing frida-server (iOS Jailbroken)



- Start Cydia
- Add Frida's repository by going to Manage -> Sources -> Edit -> Add
- Enter https://build.frida.re

NowSecure

# Smoke Test

```
$ frida-ls-devices
Id          Type    Name
----------  ------  ------------
local       local   Local System
9AXAY1GB10  usb     Pixel 3a
socket      remote  Local Socket
```

Same commands on iOS/iPadOS as on Android

```
$ frida-ps -Uia
  PID  Name                        Identifier
-----  --------------------------  ----------------------------------
 2014  Android Services Library     com.google.android.ext.services
 1166  Android System               android
21365  Calendar Storage            com.android.providers.calendar
 1166  Call Management             com.android.server.telecom
31717  Carrier Services            com.google.android.ims
21412  Contacts                    com.google.android.contacts
...
```

I could have also used:
`frida-ps -D 9AXAY1GB10 -ia`

NowSecure

# Common ‘duh’ moments

```
$ frida-ps -U
Waiting for USB device to appear...
```

Make sure your device is plugged in

This only applies to Android, on iOS this likely means frida-server is not installed.

```
$ frida-ps -Uia
Failed to enumerate applications: unable
to connect to remote frida-server: closed
```

```
$ frida-ps -U | grep frida-server
  992  frida-server
$ adb shell kill 992
```

```
$ adb shell "/data/local/tmp/frida-server &"
```

NowSecure

# Common ‘duh’ moments

```
frida-discover -U -f OWASP.iGoat-Swift
Failed to spawn: this feature requires an iOS Developer
Disk Image to be mounted; run Xcode briefly or use
ideviceimagemounter to mount one manually

% frida-discover -U -f OWASP.iGoat-Swift
Failed to attach: need gadget to attach; its default
location is:
```

Run XCode, and make sure your device is jailbroken

NowSecure

# Frida Basic Commands

```
$> frida --version
```

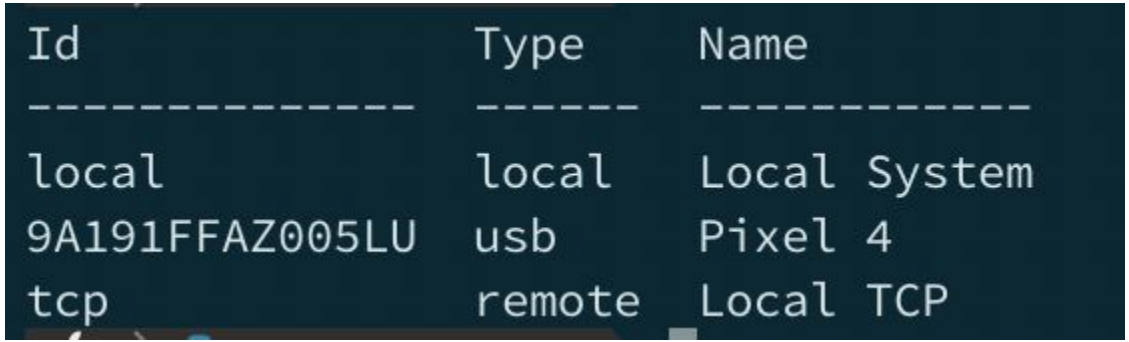
Find version information. Make sure that this matches the version of frida-server you put on your device!

NowSecure

# Frida Basic Commands

```
$> frida-ls-devices
```

Useful if you have multiple devices connected to your computer. List all devices.




NowSecure

# Frida Basic Commands

```
$> frida-ps -U
```

Lists processes on the device connected via USB (-U).

NowSecure

# Frida Basic Commands

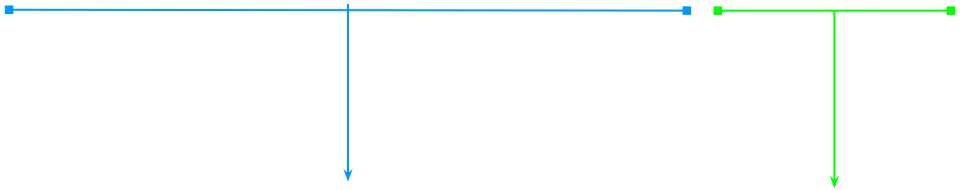
```
$> frida-ps -Ua
```

- Will list the running applications on the device connected via USB.

NowSecure

# frida-ps

```
frida-ps -U -ia
```



We're connecting to our USB connected device's (`-U`)...

We're connecting to our USB connected device's (`-ia`)...

*This command is fundamentally useful so it's important to remember it for later*

NowSecure

# Frida Basic Commands

```
$> frida -U -f owasp.mstg.uncrackable1 -l myscript.js
```

- Spawns a specific application by package name and loads a local script named myscript.js.
- NOTE: the process will be paused until you type `%resume`
- Add `--no-pause` if you want the script to execute immediately.

NowSecure

# Basic Analysis Workflow

1. Static analysis / unpack / decompile
2. Search for values of interest
3. Write or generate your script(s)
4. Execute the script

NowSecure

# Using frida-trace to Generate Starter Scripts

- frida-trace is a tool for dynamically tracing function calls.
- Especially powerful is the ability to trace any function for a given library, specified by the "`-I`".
- As soon as frida-trace is started, it will automatically enumerate the functions for that given module and also inside the current working directory, creates a "handler". This is a JavaScript file associated with a specific native function that we can edit and amend with our own code; for example to print out more information such as the arguments.

# What to Trace

## Network

```
frida-trace -U AppName -m "-[NSURL* *HTTP*]"
```

## Local Filesystem

```
frida-trace -U AppName -m "-[NSFileManager *System*]"
```

## Cryptographic Functions

```
frida-trace -U AppName -I "libcommonCrypto*"
```

NowSecure

# Frida Cheat Sheet

```
frida -U -f com.App.Identifier -l someScript.js
```

NowSecure

# Where to start...?

```
$ frida-discover
```

- App internal function discovery
- Statistics
- Crashed are common
- Results may vary

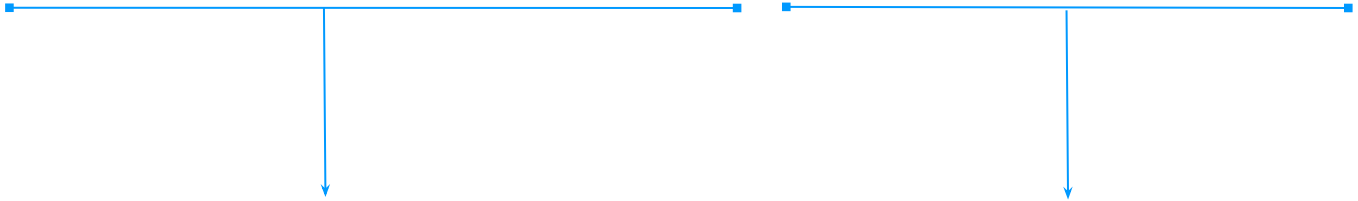
```
$ frida-trace
```

- Trace individual functions
- Capable of being a laser or a flood light
- You need to know what you're looking for

NowSecure

# Generic Discover

```
frida-discover -U -f com.app.ID
```



We're connecting to our USB connected device's (`-U`)...

AND we're selecting our App based on its identifier (`-f com.app.ID`)

NowSecure

# Discover

```
$ frida-discover -U -f OWASP.iGoat-Swift
Tracing 1 threads. Press ENTER to stop.
libobjc.A.dylib
       Calls        Function
       5253         objc_msgSend
       1877         sub_1f60
...
NSFileManager
       Calls        Function
       3            sub_1d8980
       2            sub_1d8374
...
libcommonCrypto.dylib
       Calls        Function
       1            CNEncoderUpdate
       1            CNEncoderCreate
       1            CNEncoderRelease
       1            CNEncoderFinal
       1            sub_b170
       1            CCDigest
       1            sub_4b54
```

Modules and Frameworks

Function Calls

*Results may vary...*

NowSecure

# Trace

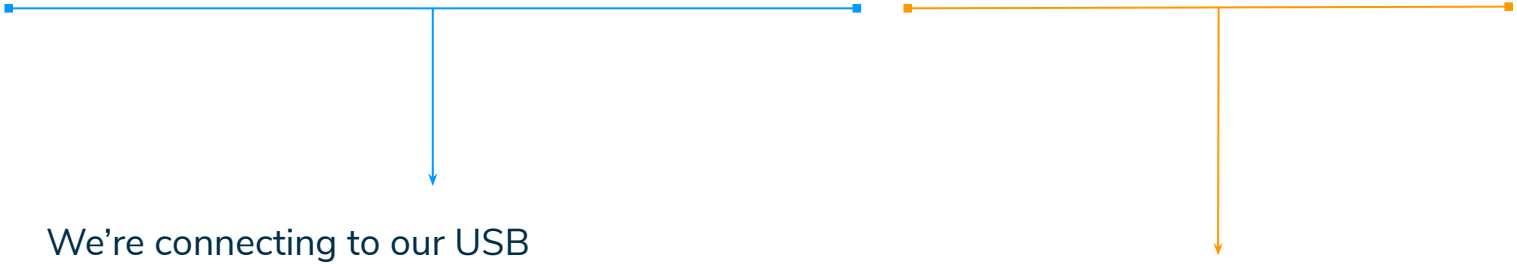
- We know the app uses `NSFileManager` class
- What should we be searching for?
- What's the right recipe for starting things off?

```
$ frida-trace ????
```

NowSecure

# Generic Trace

```
frida-trace -U -f com.app.ID -m “-[Class method:]”
```



We’re connecting to our USB connected device’s (`-U`)...

AND we’re selecting our App based on its identifier (`-f com.app.ID`)

We’re tracing class methods with the matching name specified

*There are other switches we can use to trace other types of functions*

NowSecure

# NSFileManager Method Enumeration

- We can use `-m "-[NSFileManager *"` to evaluate all methods in the NSFileManager, but it can be a bit much
  - Go ahead and try it.

- We can be more specific than that too. A better option might be to just look for methods that start with `file`

- `-m "-[NSFileManager file*]"`

```
$ frida-trace -U -f OWASP.iGoat-Swift -m "-[NSFileManager file*]"
Instrumenting...
...
Started tracing 13 functions. Press Ctrl+C to stop.
           /* TID 0x507 */
    82 ms  -[NSFileManager fileSystemRepresentationWithPath:0x1e8aecd60]
    84 ms  -[NSFileManager fileSystemRepresentationWithPath:0x2816d3980]
           /* TID 0x1c03 */
    94 ms  -[NSFileManager fileSystemRepresentationWithPath:0x1e9a35228]
           /* TID 0x507 */
   129 ms  -[NSFileManager fileSystemRepresentationWithPath:0x281bd6670]
           /* TID 0x2b07 */
   175 ms  -[NSFileManager fileExistsAtPath:0x281cde280]
   176 ms  -[NSFileManager fileExistsAtPath:0x281cd7660]
   178 ms  -[NSFileManager fileExistsAtPath:0x281cd7660]
           /* TID 0x5a03 */
   227 ms  -[NSFileManager fileExistsAtPath:0x280adbc30
isDirectory:0x16e116107]
   230 ms  -[NSFileManager fileSystemRepresentationWithPath:0x1e8a13080]
   230 ms  -[NSFileManager fileSystemRepresentationWithPath:0x283695740]
   233 ms  -[NSFileManager fileSystemRepresentationWithPath:0x1f0cfa3f0]
```

NowSecure

# File System Interactions

`NSFileManager`

- Class for app to interacting with file system

`fileExistsAtPath`

- Method for determine if a file exists at a supplied path

```
NSFileManager *fileManager =
    [[NSFileManager alloc] init];
NSString *path =
    [[self documentsDirectory]
stringByAppendingPathComponent:name];
BOOL isDir;
BOOL fileExists =
     [fileManager fileExistsAtPath:path
         isDirectory:&isDir];
```

NowSecure

# Tracing File System Use

```
$ frida-trace -U -f OWASP.iGoat-Swift -m "-[NSFileManager fileExistsAtPath:]"
Instrumenting...
-[NSFileManager fileExistsAtPath:]: Auto-generated handler at
"/__handlers__/NSFileManager/fileExistsAtPath_.js"
Started tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x1a07 */
   157 ms  -[NSFileManager fileExistsAtPath:0x282c4a9a0]
   157 ms  -[NSFileManager fileExistsAtPath:0x282c4ad00]
   160 ms  -[NSFileManager fileExistsAtPath:0x282c4a9a0]
           /* TID 0x507 */
 25489 ms  -[NSFileManager fileExistsAtPath:0x2839426c0]
 25501 ms  -[NSFileManager fileExistsAtPath:0x282b65e50]
 25501 ms  -[NSFileManager fileExistsAtPath:0x282b65ef0]
 25520 ms  -[NSFileManager fileExistsAtPath:0x283f4a5d0]
 25522 ms  -[NSFileManager fileExistsAtPath:0x281dcedc0]
 25522 ms  -[NSFileManager fileExistsAtPath:0x283a542d0]
 25523 ms  -[NSFileManager fileExistsAtPath:0x281dceec0]
```

NowSecure

# What about the _handler_?

- They’re hard to ignore
- Frida-trace generate scripts for each method it enumerates during a trace.
- Don’t be afraid to delete them, as they can be re-generated
- We can change it...

```
/*
 * Auto-generated by Frida. Please modify to match
 * the signature of -[NSFileManager
 * fileExistsAtPath:].
 */
{
  onEnter: function (log, args, state) {
    log('-[NSFileManager fileExistsAtPath:' +
       args[2] + ']');
  },
  onLeave: function (log, retval, state) {
  }
}
```

NowSecure

# frida-trace _handler_ Improvement

- We need to figure out what `args[2]` is.
- We can convert it to a readable format
- `ObjC.Object(args[2]).toString()`

```
{
  onEnter: function (log, args, state) {
    log('-[NSFileManager fileExistsAtPath:' +
        ObjC.Object(args[2]).toString() + ']');
  },
  onLeave: function (log, retval, state) {
  }
}
```

NowSecure

# Tracing File System Use++

```
$ frida-trace -U -f OWASP.iGoat-Swift -m "-[NSFileManager fileExistsAtPath:]"
Instrumenting...
-[NSFileManager fileExistsAtPath:]: Loaded handler at "/__handlers__/NSFileManager/fileExistsAtPath_.js"
Started tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x5407 */
   155 ms  -[NSFileManager
fileExistsAtPath:/System/Library/Frameworks/QuartzCore.framework/default.metallib]
           /* TID 0x507 */
 20151 ms  -[NSFileManager
fileExistsAtPath:/var/mobile/Library/ConfigurationProfiles/PublicInfo/MCMeta.plist]
 20162 ms  -[NSFileManager fileExistsAtPath:/System/Library/TextInput/TextInput_dictation.bundle]
 20163 ms  -[NSFileManager fileExistsAtPath:/System/Library/TextInput/kbd]
 20187 ms  -[NSFileManager
fileExistsAtPath:/System/Library/PrivateFrameworks/TextInput.framework/Keyboard-default.plist]
 47603 ms  -[NSFileManager
fileExistsAtPath:/var/mobile/Containers/Data/Application/9DB2574A-58F2-460E-987B-41DECBC1CCD8/Library/Appli
cation Support/CoreData.sqlite]
```

NowSecure

# Tracing File System Use++

```
$ frida-trace -U -f sg.vp.MSTG.JWT -m "-[NSURLSession dataTaskWithRequest*]"
Instrumenting...
-[NSURLSession dataTaskWithRequest:]: Loaded handler at
"/Users/aramirez/Desktop/__handlers__/NSURLSession/dataTaskWithRequest_.js"
-[NSURLSession dataTaskWithRequest:completionHandler:]: Loaded handler at
"/Users/aramirez/Desktop/__handlers__/NSURLSession/dataTaskWithRequest_completionHandler_.js"
-[NSURLSession dataTaskWithRequest:uniqueIdentifier:]: Loaded handler at
"/Users/aramirez/Desktop/__handlers__/NSURLSession/dataTaskWithRequest_uniqueIdentifier_.js"
Started tracing 3 functions. Press Ctrl+C to stop.
           /* TID 0x507 */
  5865 ms  -[NSURLSession dataTaskWithRequest:<NSMutableURLRequest: 0x2802e8410> { URL:
http://54.255.184.119:8181/auth/login }]
 25688 ms  -[NSURLSession dataTaskWithRequest:<NSMutableURLRequest: 0x2802f5b30> { URL:
http://54.255.184.119:8181/auth/login }]
 28307 ms  -[NSURLSession dataTaskWithRequest:<NSMutableURLRequest: 0x2802e9ff0> { URL:
http://54.255.184.119:8181/auth/login }]
```

NowSecure

# Hands-on Exercise: Tracing NSURLSession

- We can also trace network requests with frida-trace, for requests sent using additional security controls.

- Very popular network framework **`NSURLSession`**

- How would you trace **`dataTaskWithRequest:`**

- Can you also convert the arguments to a readable format?

NowSecure

# Walkthrough of Network Trace

- We can use `-m "-[NSURLSession dataTaskWithRequest:*]"`
- We can also modify handlers using `ObjC.Object(args[2]).toString()` to print URLs
- We can actually go a bit further if we look at the `NSURLRequest` documentation

NowSecure

# Tracing Network Calls

```
$ frida-trace -U -f sg.vp.MSTG.JWT -m "-[NSURLSession dataTaskWithRequest*]"
Instrumenting...
-[NSURLSession dataTaskWithRequest:]: Loaded handler at
"/__handlers__/NSURLSession/dataTaskWithRequest_.js"
-[NSURLSession dataTaskWithRequest:completionHandler:]: Loaded handler at
"/__handlers__/NSURLSession/dataTaskWithRequest_completionHandler_.js"
-[NSURLSession dataTaskWithRequest:uniqueIdentifier:]: Loaded handler at
"/__handlers__/NSURLSession/dataTaskWithRequest_uniqueIdentifier_.js"
Started tracing 3 functions. Press Ctrl+C to stop.
           /* TID 0x507 */
  5865 ms  -[NSURLSession dataTaskWithRequest:<NSMutableURLRequest: 0x2802e8410> { URL:
http://54.255.184.119:8181/auth/login }]
 25688 ms  -[NSURLSession dataTaskWithRequest:<NSMutableURLRequest: 0x2802f5b30> { URL:
http://54.255.184.119:8181/auth/login }]
 28307 ms  -[NSURLSession dataTaskWithRequest:<NSMutableURLRequest: 0x2802e9ff0> { URL:
http://54.255.184.119:8181/auth/login }]
```

NowSecure

# Building a better _handler_

```
{
  onEnter: function (log, args, state) {
    log('-[NSURLSession dataTaskWithRequest:' + args[2] + ']');
    var Request = new ObjC.Object(args[2]);
    log('method: ' + Request.HTTPMethod().toString());
    log('url: ' + Request.URL().toString());
    log('headers: ' +
       Request.allHTTPHeaderFields().toString());
    //We have to Data to String covert for Body
    var Body = ObjC.Object(args[2]).HTTPBody();
    var Body_String =
       ObjC.classes.NSString.alloc().initWithData_encoding_(Body, 4);
    log('body: ' + Body_String);
  },

  onLeave: function (log, retval, state) {
  }
}
```

**Accessing Request Components**

`var httpMethod: String?`
The HTTP request method.

`var url: URL?`
The URL being requested.

`var httpBody: Data?`
The request body.

`var httpBodyStream: InputStream?`
The request body as an input stream.

`var mainDocumentURL: URL?`
The main document URL associated with the request.

**Getting Header Fields**

`var allHTTPHeaderFields: [String : String]?`
A dictionary containing all of the HTTP header fields for a request.

From: https://developer.apple.com/documentation/foundation/nsurlrequest

NowSecure

# Tracing Network Calls++

```
$ frida-trace -U -f sg.vp.MSTG.JWT -m "-[NSURLSession dataTaskWithRequest*]"
Instrumenting...
-[NSURLSession dataTaskWithRequest:]: Loaded handler at
"/__handlers__/NSURLSession/dataTaskWithRequest_.js"
-[NSURLSession dataTaskWithRequest:completionHandler:]: Loaded handler at
"/__handlers__/NSURLSession/dataTaskWithRequest_completionHandler_.js"
-[NSURLSession dataTaskWithRequest:uniqueIdentifier:]: Loaded handler at
"/__handlers__/NSURLSession/dataTaskWithRequest_uniqueIdentifier_.js"
Started tracing 3 functions. Press Ctrl+C to stop.
           /* TID 0x507 */
  3960 ms  -[NSURLSession dataTaskWithRequest:0x283e1c7f0]
  3960 ms  method: POST
  3960 ms  url: http://54.255.184.119:8181/auth/login
  3960 ms  headers: {
    "Content-Type" = "application/x-www-form-urlencoded; charset=utf-8";
}
  3960 ms  body: email=&password=
```

NowSecure

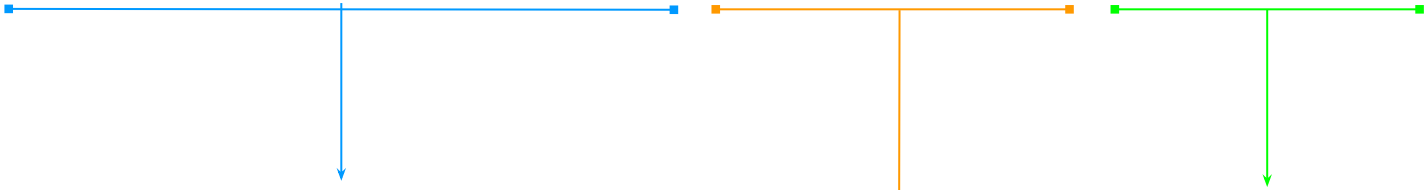
# Can we script this?

- We can use the Frida CLI to run it
- We can use our existing handler
- This will later-on provide some extra fun and short-cuts

NowSecure

# Generic frida script

```
frida -U -f com.app.ID -l Script.js --no-pause
```



We're connecting to our USB connected device's (`-U`)...

AND we're selecting our App based on its identifier (`-f com.app.ID`)

We're selecting our Script.js script (`-l Script.js`)

`--no-pause` is optional. It tells frida to launch the app right as you run the script. Just can always run `%resume`

NowSecure

# NetworkSniffer.js

```
// -l NetworkSniffer.js --no-pause
try{
    var dataTaskWithRequestSniff = ObjC.classes.NSURLSession["- dataTaskWithRequest:"];
    Interceptor.attach(dataTaskWithRequestSniff.implementation, {
        onEnter: function (args) {
            var Request = new ObjC.Object(args[2]);
            console.log('-[NSURLSession dataTaskWithRequest:' + args[2] + ']');
            var Request = new ObjC.Object(args[2]);
            console.log('method: ' + Request.HTTPMethod().toString());
            console.log('url: ' + Request.URL().toString());
            console.log('headers: ' + Request.allHTTPHeaderFields().toString());
            //We have to Data to String covert for Body
            var Body = ObjC.Object(args[2]).HTTPBody();
            var Body_String = ObjC.classes.NSString.alloc().initWithData_encoding_(Body, 4);
            console.log('body: ' + Body_String);
        }
        /*
         onLeave: function (log, retval, state) {
         }
         */
    });
} catch(err){
    console.log(err)}
```

Running command `frida -U -f com.app.ID -l NetworkSniffer.js --no-pause`

We still need to tell frida what we want to trace to, and we still need to attach to

Swap `log` out for `console.log`

Remove or comment out `onLeave:`, this is not used in this script

`try/catch` will help us find our mistakes

NowSecure

# SWITCH TO ANDROID

NowSecure

# Smoke Test

```
$ frida-ls-devices
Id          Type    Name
----------  ------  ------------
local       local   Local System
9AXAY1GB10  usb     Pixel 3a
socket      remote  Local Socket
```

Same commands on iOS/iPadOS as on Android

```
$ frida-ps -Uia
  PID  Name                        Identifier
-----  --------------------------  ----------------------------------
 2014  Android Services Library     com.google.android.ext.services
 1166  Android System               android
21365  Calendar Storage            com.android.providers.calendar
 1166  Call Management             com.android.server.telecom
31717  Carrier Services            com.google.android.ims
21412  Contacts                    com.google.android.contacts
...
```

I could have also used:
`frida-ps -D 9AXAY1GB10 -ia`

NowSecure

# Common ‘duh’ moments

```
$ frida-ps -U
Waiting for USB device to appear...
```

Make sure your device is plugged in

This only applies to Android, on iOS this likely means frida-server is not installed.

```
$ frida-ps -Uia
Failed to enumerate applications: unable
to connect to remote frida-server: closed
```

```
$ frida-ps -U | grep frida-server
  992  frida-server
$ adb shell kill 992
```

```
$ adb shell "/data/local/tmp/frida-server &"
```

NowSecure

# Rinse and Repeat?

*Is everything we've learned so far transferable?*

## YES

- `frida-trace`, `discover`, and the `CLI` are all still going to be useful
- Android apps can leverage the NDK to use native code

## NO

- Android apps rely heavily on Java/Kotlin
- We will rely more heavily on the CLI to run our analysis
- Android is a little bit easier

NowSecure

# Discover

- We can still use `frida-discover`
- What is different?
- What output do you see?

```
$ frida-discover -U -f com.app.ID
```

NowSecure

# Discuss Discover

- Way more function data
- We’re seeing ART compiler, along with other system utilities
- You might also notice that if an app does have some `lib*.so`, then that will show up too

```
$ frida-discover -U -f com.example.app
Tracing 7 threads. Press ENTER to stop.

libart.so
          Calls      Function
          4703       _ZN3art6mirror6String6EqualsENS_6ObjPtrIS1_EE
          2078       sub_2b65d4
          1988       _ZNK3art7Runtime19IsActiveTransactionEv
          1096       sub_a1af0
          1087       sub_a1200
          971        _ZNK3art6Thread13DecodeJObjectEP8_jobject
          943        sub_a16a0
          909        sub_a7ba4
          834        sub_3c7a8c
          830        sub_3c7600
          705        sub_aaec8
          705        _ZN3art3jit3Jit10AddSamplesEPNS_6ThreadEPNS_9ArtMethodEtb
          695        sub_36fa48
          633        _ZN3art5Mutex13ExclusiveLockEPNS_6ThreadE
          624        sub_a15c0
          611        _ZN3art5Mutex15ExclusiveUnlockEPNS_6ThreadE
          604        sub_a1520
          590        MterpShouldSwitchInterpreters
          558        _ZN3art9ArtMethod28IsOverridableByDefaultMethodEv
          554        sub_2bf41c
          526        sub_a11f0
          517        sub_1dd4cc
          496        sub_1069e0
          485        sub_5267f0
          478        sub_a11a0
          453        _ZN3art6mirror5Class9SetStatusENS_6HandleIS1_EENS_11ClassStatusEPNS_6ThreadE
          ...
```

NowSecure

# Trace

- We can still use frida-trace
  - We can use `-I “libfile.so”` for native

- Is that the best we can do?

```
frida-trace -U -f com.app.ID -I “libExample.so”
```

- What are we missing?

NowSecure

# Discuss Trace

- Java/Kotlin?

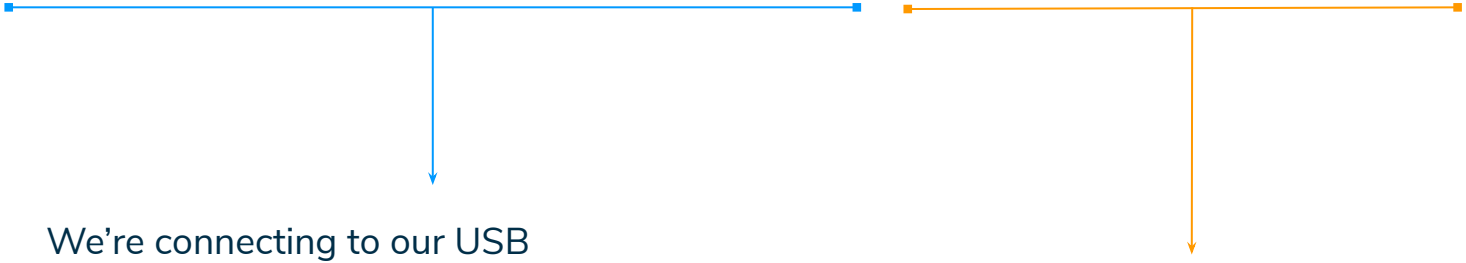
- We've limited ourselves to native
  - FYI, we should also be seeing similar _handler_ files we saw before in native

- Not all apps have `*.so` files, some of the stuff in discover and `-I` traces will be out of scope.


NowSecure

# Generic Java Method Trace

```
frida-trace -U -f com.app.ID -j '*!*Method*/isu'
```



We're connecting to our USB connected device's (`-U`)...

AND we're selecting our App based on its identifier (`-f com.app.ID`)

We're tracing methods, but this time, we're fudging it a bit using some extra wildcards (`-j '*!*Method*/isu'`)

NowSecure

# Hands-on Java Trace Exercise

- Pick any common method
- Pick any app
- See what you get back?  Do you notice a major difference from what you saw in iOS?

**Good Examples:**

```
frida-trace -U -f com.app.ID -j '*!*getExt*/isu'

frida-trace -U -f com.app.ID -j '*!*cert*/isu'

frida-trace -U -f com.app.ID -j '*!*loadUrl*/isu'
```

NowSecure

NowSecure

# Enumerating App Classes in Android

- Enumerating app classes can be useful to determine the contents of the application

- This script can be executed by running the following command:

```
frida -U --no-pause -l
androidListLoadedClasses.js -f
<binary_name>
```

Note: the output will be large

```
Java.perform(function() {
    Java.enumerateLoadedClasses({
        onMatch: function(className) {
            console.log(className);
        },
        onComplete: function() {}
    });
});
```



**androidListLoadedClasses.js**

NowSecure

# Enumerating App Classes in Android

- The output produced by the script can be filtered by utilizing the `grep` command.

- Use the following command to filter classes containing a specific <term>:

```
frida -U --no-pause -l
androidListLoadedClasses.js -f
<binary_name> | grep -i <term>
```

- Alternatively, use the following command to filter classes that DO NOT contain the specific <term> (e.g. negative search):

```
frida -U --no-pause -l
androidListLoadedClasses.js -f
<binary_name> | grep -v <term>
```

```
Java.perform(function() {
    Java.enumerateLoadedClasses({
        onMatch: function(className) {
            console.log(className);
        },
        onComplete: function() {}
    });
});
```



**androidListLoadedClasses.js**

NowSecure

# Enumerating App Classes in Android

- Let's say we want to study how WebViews are utilized in a certain app and we use the command in the previous slide to filter all classes that contain 'WebView' in their name.

- We can trace class calls and utilization for any WebView class by using frida-trace:

```
frida-trace -U -f <binary_name> -j
'*!*WebView*'
```

```
Started tracing 43 functions. Press Ctrl+C to stop.
           /* TID 0xe93 */
 10009 ms  WebViewFactory.isWebViewSupported()
 10011 ms  <= true
 10011 ms
WebViewFactory.getWebViewContextAndSetProvider()
 10011 ms     | WebViewFactory.isWebViewSupported()
 10011 ms     | <= true
 10014 ms     |
WebViewFactory.getWebViewLibrary("<instance:
android.content.pm.ApplicationInfo>")
 10018 ms     | <= "libmonochrome.so"
 10034 ms  <= "<instance: android.content.Context,
$className: android.app.ContextImpl>"
 10050 ms
WebViewFactory.getWebViewLibrary("<instance:
android.content.pm.ApplicationInfo>")
 10051 ms  <= "libmonochrome.so"
```

NowSecure

# Enumerating Methods in a Java Class

- It is also possible to list all methods contained in a specific class along with the parameters they take. Use the code snippet in this slide (remember to modify the targetClass from "`android.webkit.WebView`" to your desired class) and execute the following command:

```
frida -U --no-pause -l
androidEnumMethods.js -f
<binary_name>
```

**androidEnumMethods.js** →

```
function enumMethods(targetClass)
{
     var hook = Java.use(targetClass);
     var ownMethods =
hook.class.getDeclaredMethods();
     hook.$dispose;
     return ownMethods;
}
setTimeout(function() {
     Java.perform(function() {
          var a =
enumMethods("android.webkit.WebView")
          a.forEach(function(s) {
               console.log(s);
          });
     });
}, 0);
```

NowSecure

# Enumerating Methods in a Java Class

- The expected output from listing all methods in the specific WebView class `android.webkit.WebView` should look as follows:

```
Spawned `<binary_name>`. Resuming main thread!
[Pixel 3a::<binary_name>]-> static boolean
android.webkit.WebView.access$1001(android.webkit.WebView,int,android.graphics.Rect)
static int android.webkit.WebView.access$101(android.webkit.WebView)
static void android.webkit.WebView.access$1101(android.webkit.WebView,android.view.ViewGroup$LayoutParams)
static void android.webkit.WebView.access$1201(android.webkit.WebView,android.content.Intent,int)
static boolean
android.webkit.WebView.access$1300(android.webkit.WebView,int,int,int,int,int,int,int,int,boolean)
static boolean android.webkit.WebView.access$1400(android.webkit.WebView,int)
static boolean android.webkit.WebView.access$1500(android.webkit.WebView,int,boolean)
static float android.webkit.WebView.access$1600(android.webkit.WebView)
static float android.webkit.WebView.access$1700(android.webkit.WebView)
static void android.webkit.WebView.access$1800(android.webkit.WebView,int,int)
static int android.webkit.WebView.access$1900(android.webkit.WebView)
static void
android.webkit.WebView.access$2001(android.webkit.WebView,android.graphics.Canvas,android.graphics.drawable.Dr
```

# Unsophsicated java trace

frida-trace -U -f <com.app.Identifier> -j '*!*certificate*/isu'

NowSecure

# Workflow on iOS Without a Jailbreak

Without a jailbreak, you'll need to embed Frida's Gadget into your application.

**Case 1. Your own application**

Add a frida-gadget shared library into your project. Then load the library. Frida server is bringed up in the address space of the process, and a jailbreak is not needed in such case.

**Case 2. A third party application**

Patch the application binary or one of its libraries, e.g. by using a tool like insert_dylib.

Step 1. Use insert_dylib.

Step 2. Re-sign the application.

Step 3. Install the application.

NowSecure

# Introduction to Reverse Engineering Tools

NowSecure

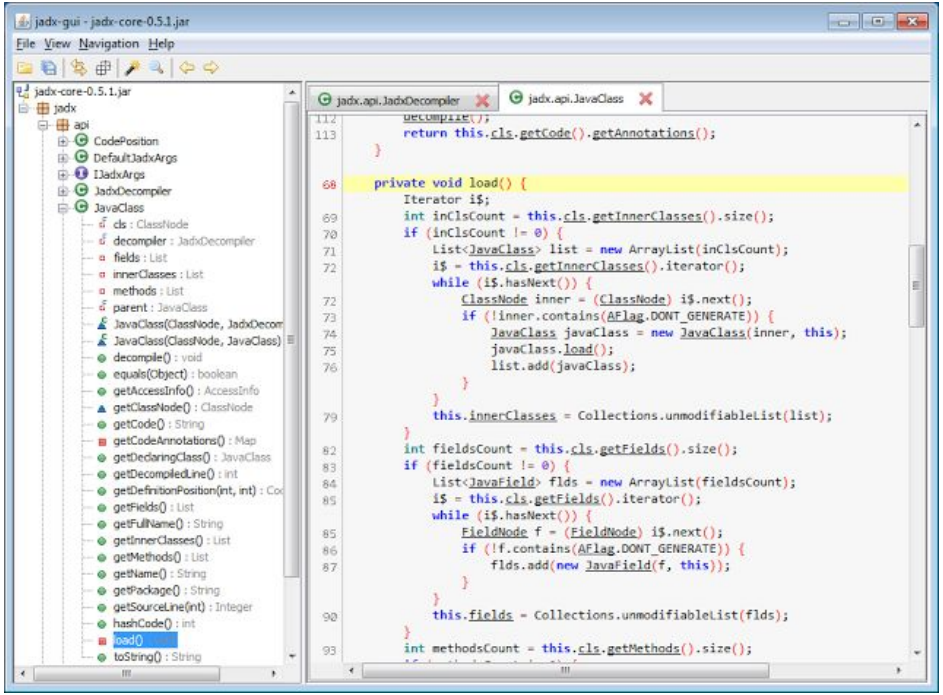
# Why Learn About RE Tools?

- While you *can* use Frida without any other tools, it helps to have assistance in locating the methods you want to hook.
- Speeds up your discovery process.

NowSecure

# jadx

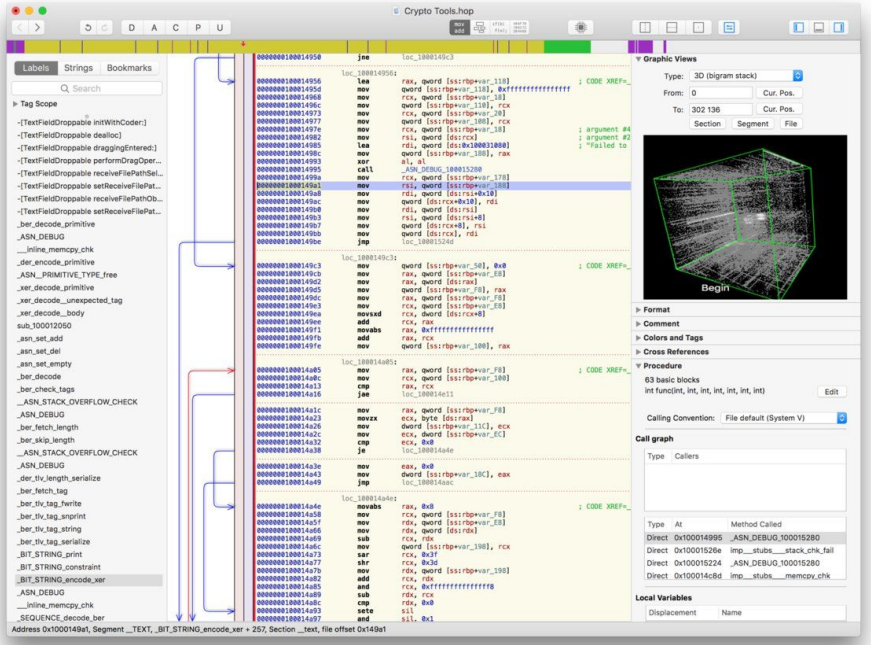
- jadx is a CLI Dex to Java decompiler that produces Java source code from Android Dex and APK files.
- Similarly, jadx-gui is a GUI version of the application.
- All Operating Systems:
  - https://github.com/skylot/jadx




NowSecure

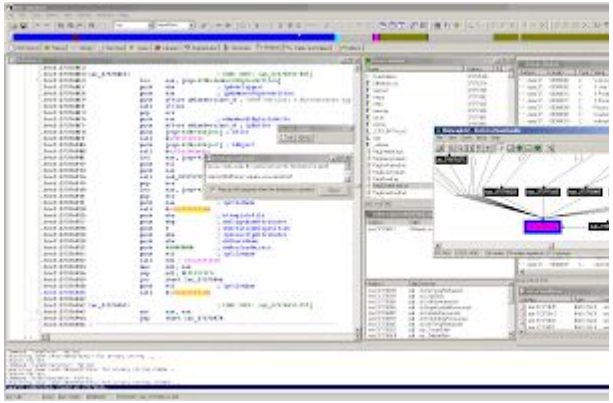
# Hopper

- GUI-driven decompiler for Mac OSX.
- Specialized in retrieving Objective-C information like selectors, strings and messages sent.
  - Able to decode the mangled Swift names.
- Hopper can disassemble binaries targeting Intel (32 and 64bits), ARM (ARMv6, ARMv7 and ARM64), and PowerPC processors.
  - It is also possible to extend to other CPU thanks to the SDK.
- https://www.hopperapp.com/



# IDA Pro

- IDA Pro is a disassembler able to reverse many types of binary code back into assembly or even pseudo-code.
- IDA Home is available in 5 versions, **each** supporting one of the common processor families:
  - x86/x64
  - ARM/ARM64
  - MIPS/MIPS64
  - PowerPC/PPC64
  - Motorola 68K/Coldfire
- IDA Professional Edition supports countless more.
- Cross-platform.
- https://www.hex-rays.com/products/ida/



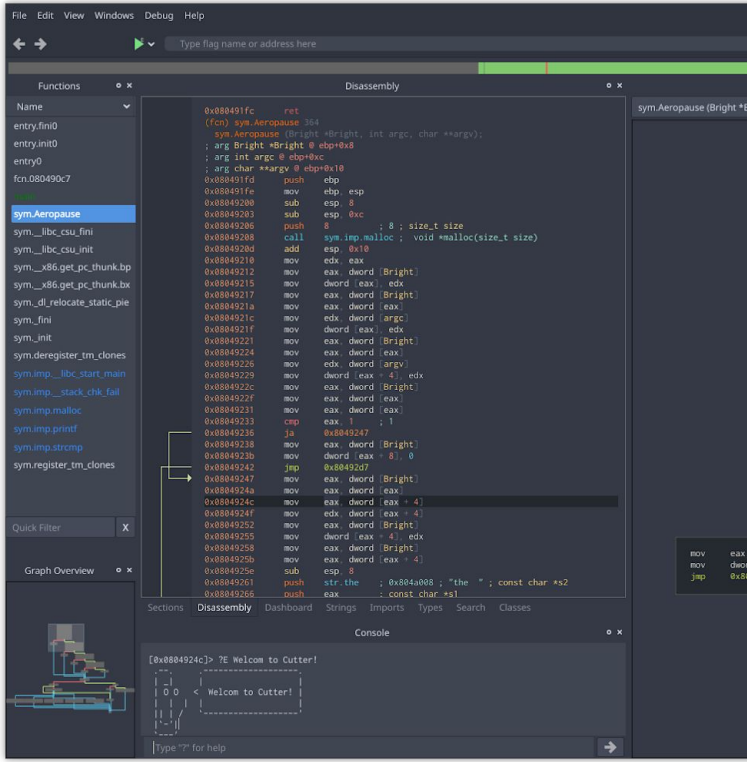
NowSecure

# radare2

- The main tool of the whole framework. It uses the core of the original open source radare hexadecimal editor and debugger.
- radare2 allows you to open a number of input/output sources as if they were simple, plain files, including disks, network connections, kernel drivers, processes under debugging, and so on.
- It implements an advanced command line interface for moving around a file, analyzing data, disassembling, binary patching, data comparison, searching, replacing, and visualizing. It can be scripted with a variety of languages, including Python, Ruby, JavaScript, Lua, and Perl.
- Started and maintained by Sergi ÀIvarez (AKA pancake).

# Cutter

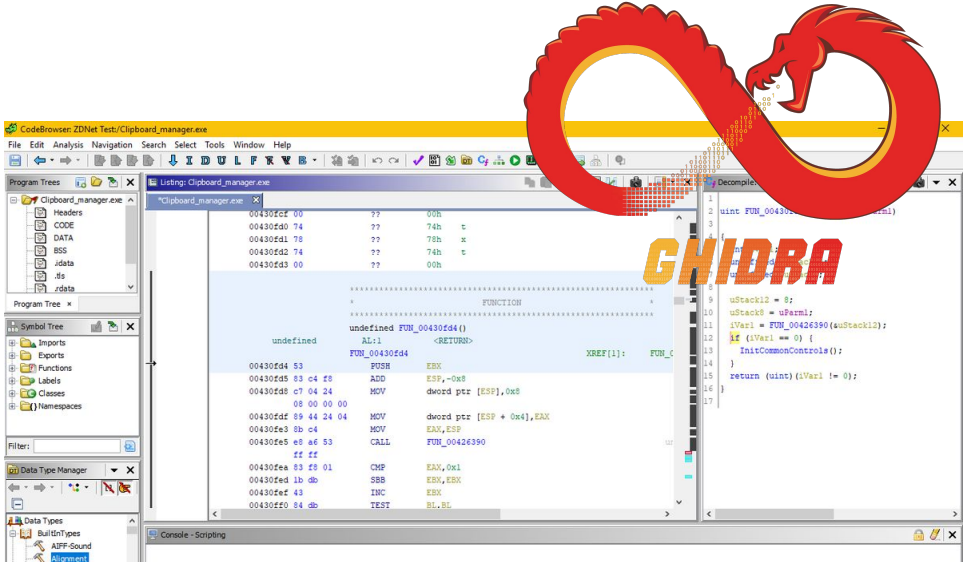
- Cutter is a free and open-source reverse engineering framework powered by radare2.
- Cutter releases are fully integrated with native Ghidra decompiler. No Java involved.




NowSecure

# Ghidra

- Ghidra is a free and open source reverse engineering tool developed by the National Security Agency.




## Key features of Ghidra:

includes a suite of software analysis tools for analyzing compiled code on a variety of platforms including Windows, Mac OS, and Linux

capabilities include disassembly, assembly, decompilation, graphing and scripting, and hundreds of other features

supports a wide variety of processor instruction sets and executable formats and can be run in both user-interactive and automated modes.

users may develop their own Ghidra plug-in components and/or scripts using the exposed API

NowSecure

# Getting Started With Frida

NowSecure

# Hands-On

NowSecure

# Example: WebView Analysis

NowSecure


NowSecure

# Three types of app security testing

Analysis of what was written

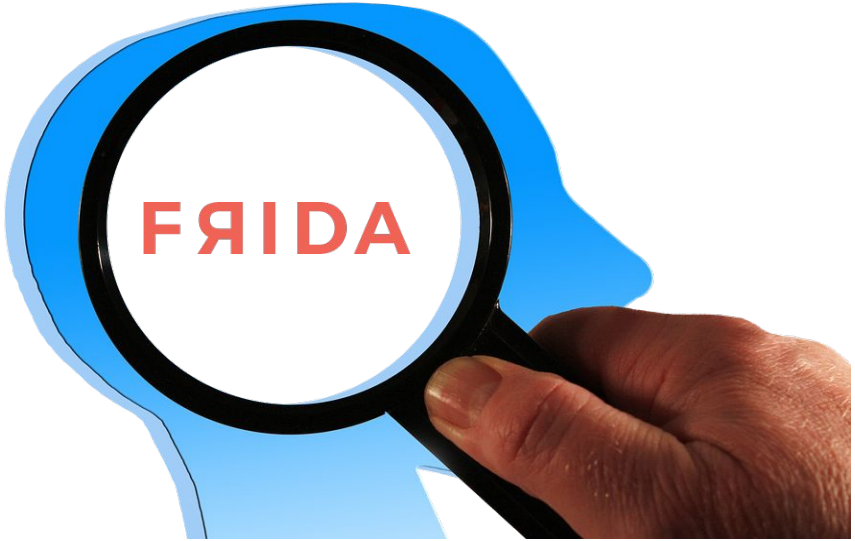
SAST - Code Decompiling/Disassembling

Analysis from outside

DAST - Network Analysis, Forensic tools

Analysis from inside

IAST - Runtime analysis, debuggers




NowSecure

# Road Blocks

Code obfuscation makes analysis difficult

App controls make tools like network proxy fail to intercept

frida will let us observer the apps runtime, without changing the app


NowSecure

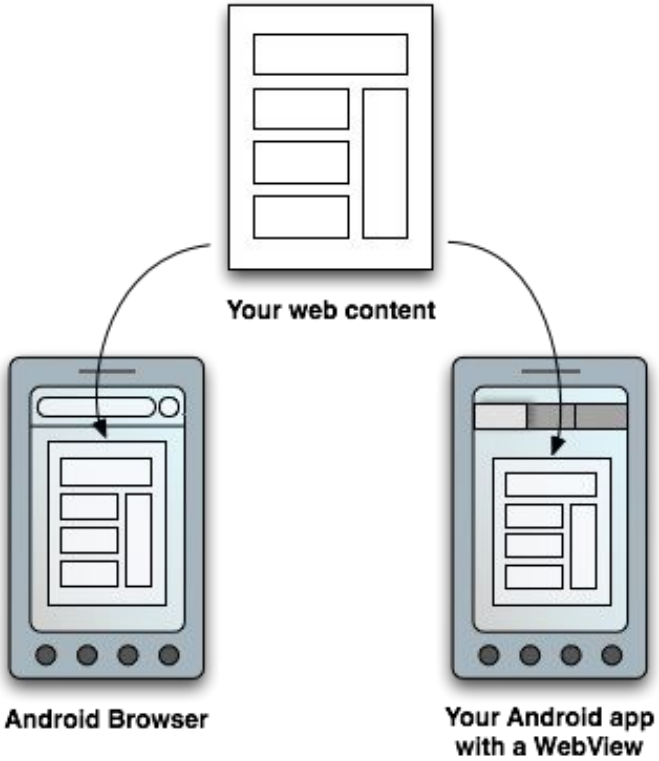
# Whats a WebView?

In app browser

Several ways to serve content to the app UI

Can be made to interact with app code

Source can be from external sources




NowSecure

# WebViews

loadData

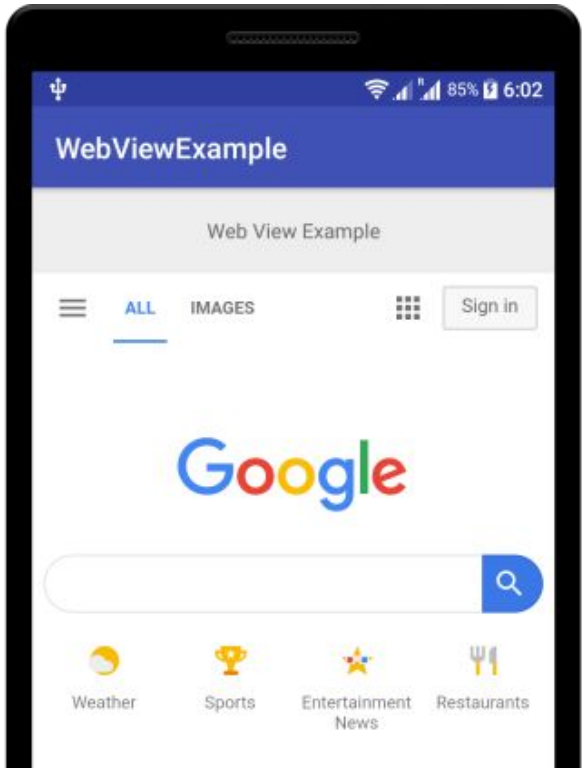
For rendering HTML content from within the app

loadDataWithBaseURL

For rendering HTML content from within the app, using content from other sources

loadURL

For loading external content, like a website




NowSecure

# Analysis Options

Static Reverse Engineering

- Use JADX or other decompiler
- Code Obfuscation can make that challenging

Instrumentation

- Use frida
- What gets loaded, and where from

NowSecure

# Frida WebView Hooks

https://pastebin.com/fuEsYswq

```
setImmediate(function(){
    Java.perform(function () {
        var WebView = Java.use("android.webkit.WebView");
        WebView.loadUrl.overload("java.lang.String").implementation = function (URL) {
            console.log("loadURL:  " + URL);
            this.loadUrl.overload("java.lang.String").call(this, URL);
        };
        WebView.loadData.overload("java.lang.String", "java.lang.String",
        "java.lang.String").implementation = function (dataLoad, mimeType, encoding) {
            console.log("loadData Data:  " + dataLoad);
            console.log("loadData Mime:  " + mimeType);
            console.log("loadData encoding:  " +  encoding);
            this.loadData.overload("java.lang.String", "java.lang.String",
            "java.lang.String").call(this, dataLoad, mimeType, encoding);
        };

        WebView.loadDataWithBaseURL.overload("java.lang.String", "java.lang.String",
        "java.lang.String", "java.lang.String", "java.lang.String").implementation = function
        (baseURL ,dataLoad, mimeType, encoding, histURL) {
            console.log("loadDataWBase baseURL:  " + baseURL);
            console.log("loadDataWBase Data:  " + dataLoad);
            console.log("loadDataWBase Mime:  " + mimeType);
            console.log("loadDataWBase encoding:  " +  encoding);
            console.log("loadDataWBase Hist:  " + histURL);
            this.loadDataWithBaseURL.overload("java.lang.String", "java.lang.String",
            "java.lang.String", "java.lang.String", "java.lang.String").call(this, baseURL
            ,dataLoad, mimeType, encoding, histURL);
        };

        WebView.clearCache.implement = function () {
            console.log("clearCache")
        };

    });
},0);
```

NowSecure


NowSecure

# Root Detection Lab

NowSecure

# Root Detection

- Root Detection is a common tamper prevention/detection technique. It alerts the user (or prevents them altogether) that they are running the app on a “compromised device”.
- How do we commonly see root detection being performed?
  - Searching for known binaries such as cydia, su, supersu, etc.
  - Attempting to execute the “su” command.
  - Searching for firmware that has not been signed as expected.

# Demo of Target

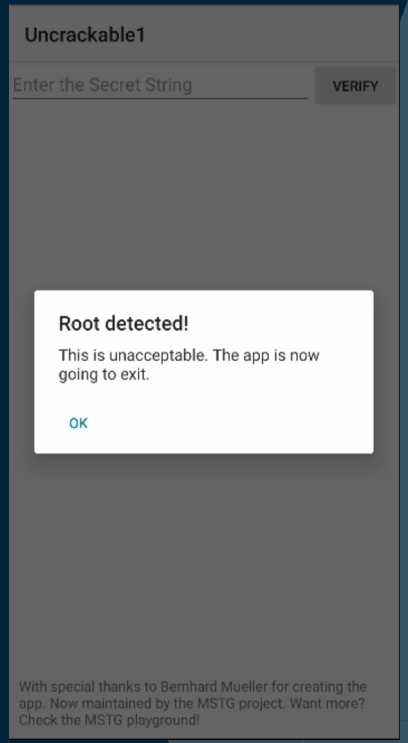
NowSecure

# Lab 1

**Goal: Bypass Root Detection in the App**

**Next Steps:**

- Download the Level 1 apk: https://bit.ly/2ZhQ9LG
- Download/open your decompiler of choice (I like jadx-gui)
- Look for the error message.
- See if you can find the function(s) that are triggering root detection.
  - What do they return?
  - Can we bypass?
- See if you can write a bypass.
  - We're available for hints/questions




NowSecure

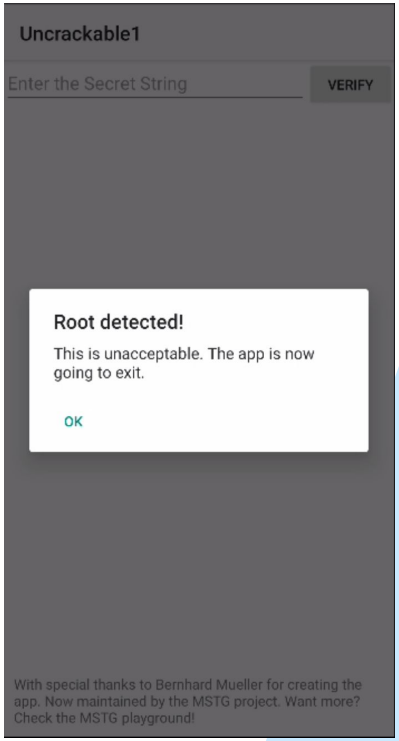
# Example 1 - Simple Root Detection

Tools to be used:


jadx-gui

Frida

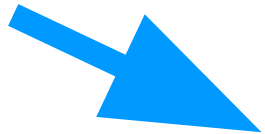
OWASP Crackme Level 1




NowSecure

# Example 1

```
public void onCreate(Bundle bundle) {
    if (c.a() || c.b() || c.c()) {
        a("Root detected!");
```



```
public static boolean a() {
    for (String file : System.getenv("
        if (new File(file, "su").exists
            return true;
        }
    }
    return false;
}

public static boolean b() {
    String str = Build.TAGS;
    return str != null && str.contains
}

public static boolean c() {
    for (String file : new String[]{"/s
        if (new File(file).exists()) {
            return true;
        }
    }
    return false;
```

NowSecure

# Example 1

https://pastebin.com/uTjwtbmV

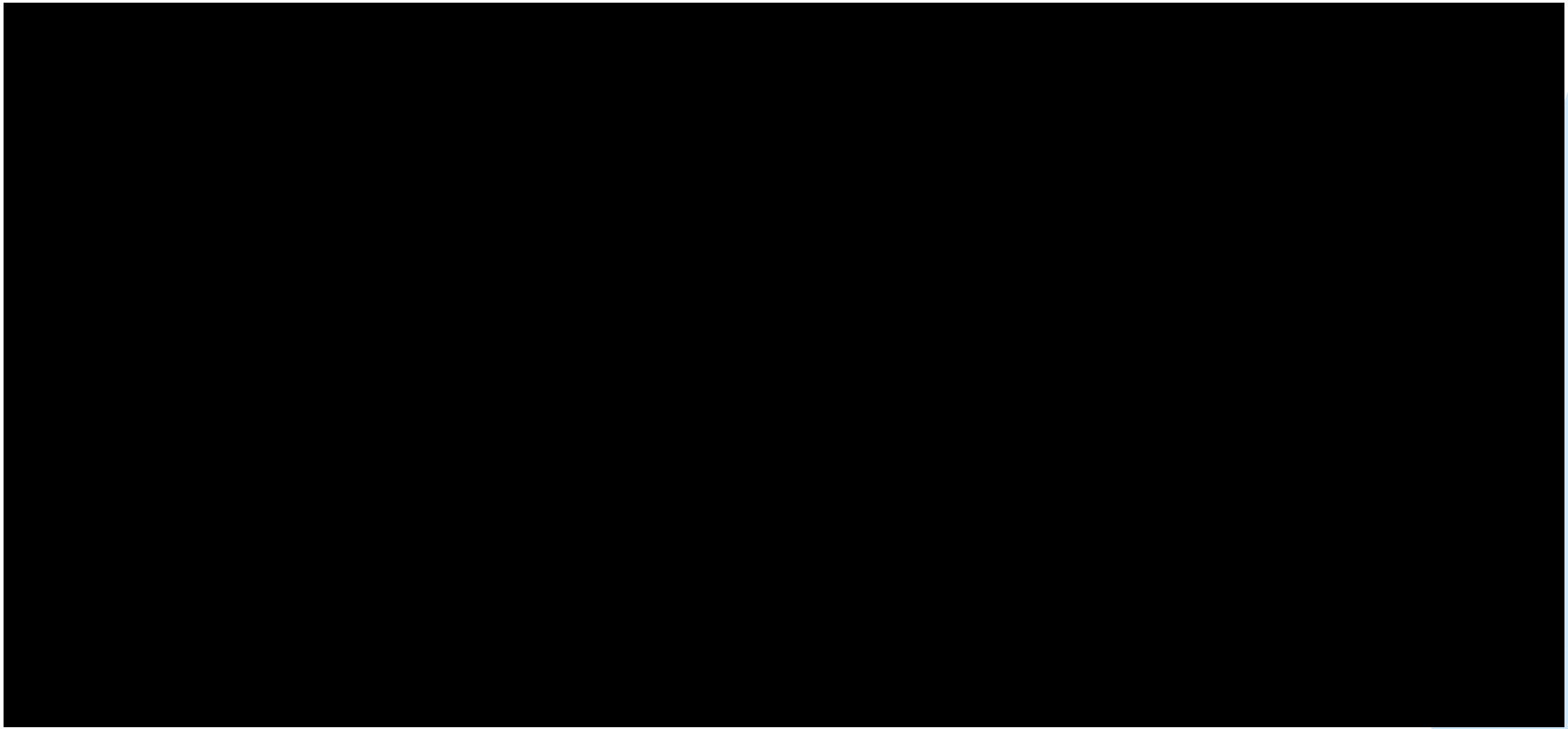
```
Java.perform(function () {
  var noroot = Java.use("sg.vantagepoint.a.c");
  noroot.a.overload().implementation = function() {
    console.log("[!] Hiding su");
    return false;
  };

  noroot.b.overload().implementation = function() {
    console.log("[!] Hiding build test-keys");
    return false;
  };

  noroot.c.overload().implementation = function() {
    console.log("[!] Hiding all of our root packages!");
    return false;
  };
});
```

NowSecure

# Completed Script in Action



NowSecure

# Crypto Analysis Lab

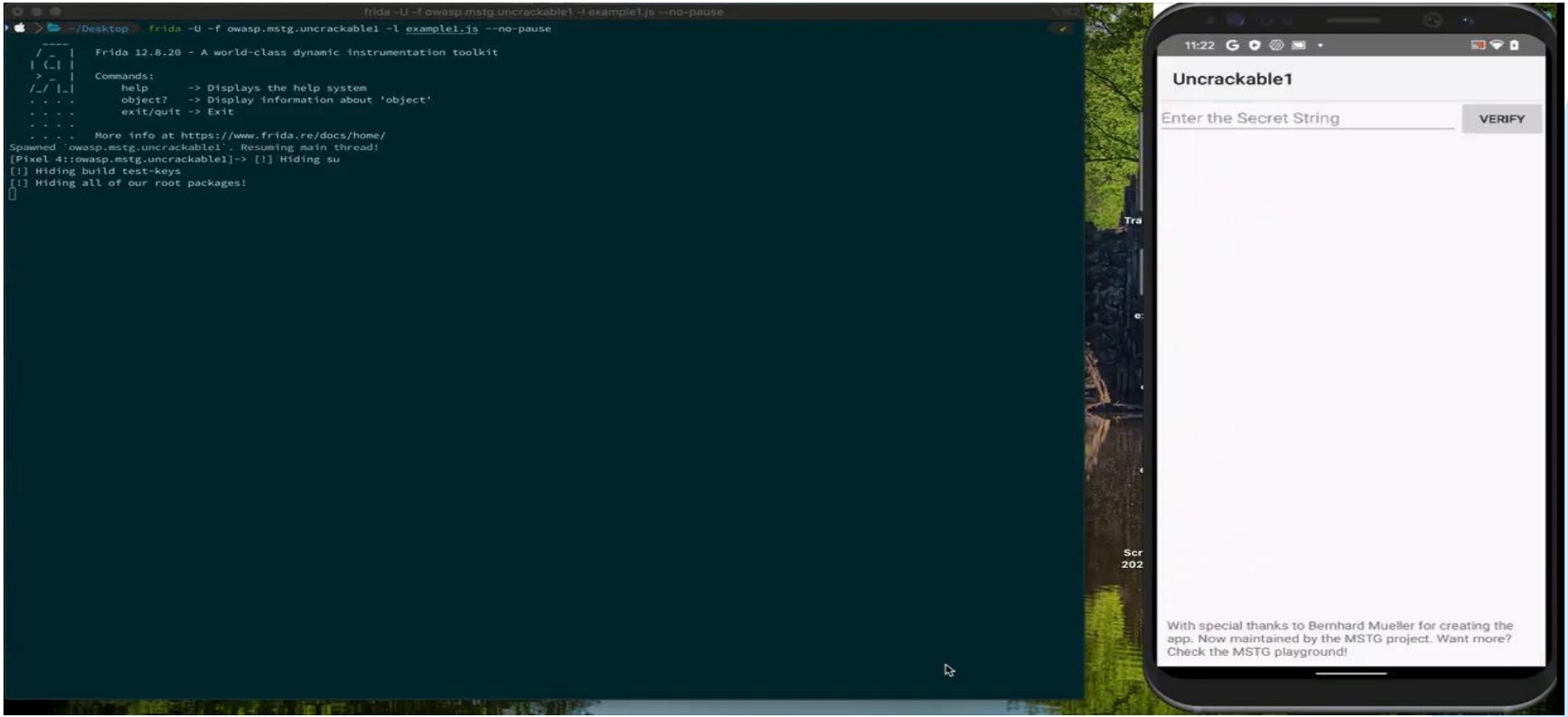
NowSecure

# Cryptography Lab

- A common issue analysts face is identifying and reversing hardcoded values in an application.
- By using Frida, we can hook values before and after cryptographic operations and display plaintext values.

NowSecure

# Demo of Target



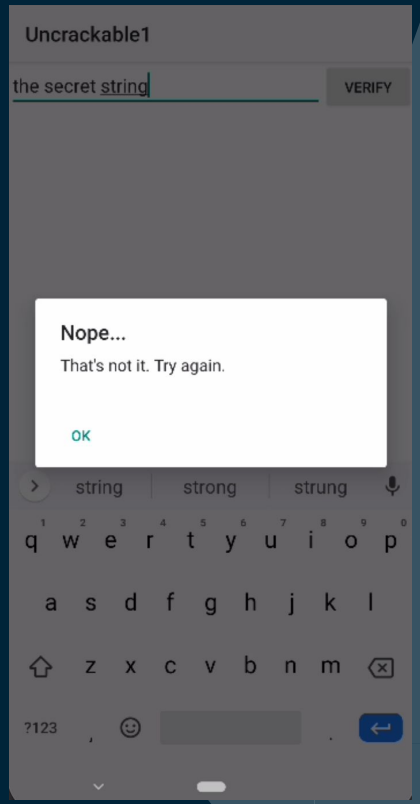

NowSecure

# Lab 2

**Goal: Identify the Secret String**

**Next Steps:**

- Look for the error message.
- See if you can find the function(s) that are validating your input.
  - Clearly the value it compares is hardcoded.
  - Can you find it?
- See if you can get the plaintext value.
  - Hint: May require some research into the Java crypto implementation.
  - We're available for hints/questions




NowSecure

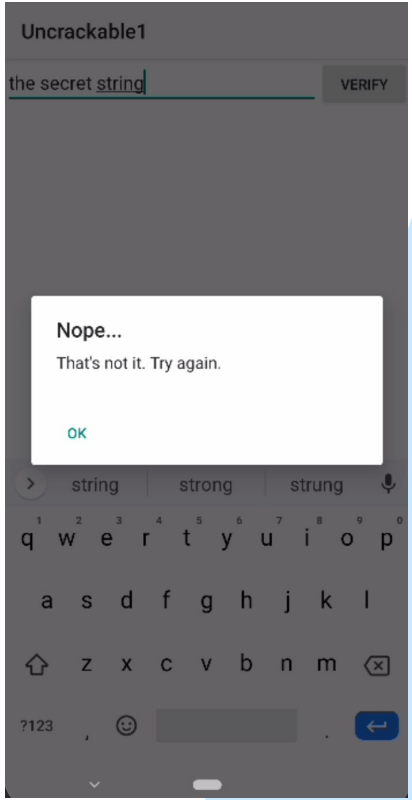
# Example 2 - Crypto Analysis

Tools to be used:

jadx-gui

Frida

OWASP Crackme Level 1



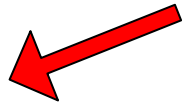

https://github.com/OWASP/owasp-mstg/tree/master/Crackmes#uncrackable-app-for-android-level-1

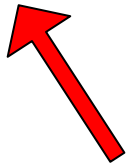
NowSecure

# Example 2

```
public static boolean a(String str) {
    byte[] bArr;
    String str2 = "8d127684cbc37c17616d806cf50473cc";
    byte[] bArr2 = new byte[0];
    try {
        bArr = sg.vantagepoint.a.a.a(b(str2), Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=", 0));
    } catch (Exception e) {
        StringBuilder sb = new StringBuilder();
        sb.append("AES error:");
        sb.append(e.getMessage());
        Log.d("CodeCheck", sb.toString());
        bArr = bArr2;
    }
    return str.equals(new String(bArr));
}

public static byte[] b(String str) {
```





NowSecure

# Example 2

```
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 }
```

NowSecure

# Example 2

## Public constructors

### SecretKeySpec

Added in API level 1

```
public SecretKeySpec (byte[] key,
                String algorithm)
```

Constructs a secret key from the given byte array.

This constructor does not check if the given bytes indeed specify a secret key of the specified algorithm. For example, if the algorithm is DES, this constructor does not check if `key` is 8 bytes long, and also does not check for weak or semi-weak keys. In order for those checks to be performed, an algorithm-specific *key specification* class (in this case: `DESKeySpec`) should be used.

NowSecure

# Example 2

```
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 }
```

NowSecure

# Example 2

## doFinal

Added in API level 1

```
public final byte[] doFinal (byte[] input)
```

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The bytes in the `input` buffer, and any input bytes that may have been buffered during a previous `update` operation, are processed, with padding (if requested) being applied. If an AEAD m
authentication tag is appended in the case of encryption, or verifie
new buffer.

Upon finishing, this method resets this cipher object to the state it
That is, the object is reset and available to encrypt or decrypt (depe
call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

### DECRYPT_MODE

```
public static final int DECRYPT_MODE
```

Constant used to initialize cipher to decryption mode.

Constant Value: 2 (0x00000002)

NowSecure

# Example 2

```
var aescrypt = Java.use("sg.vantagepoint.a.a");
```

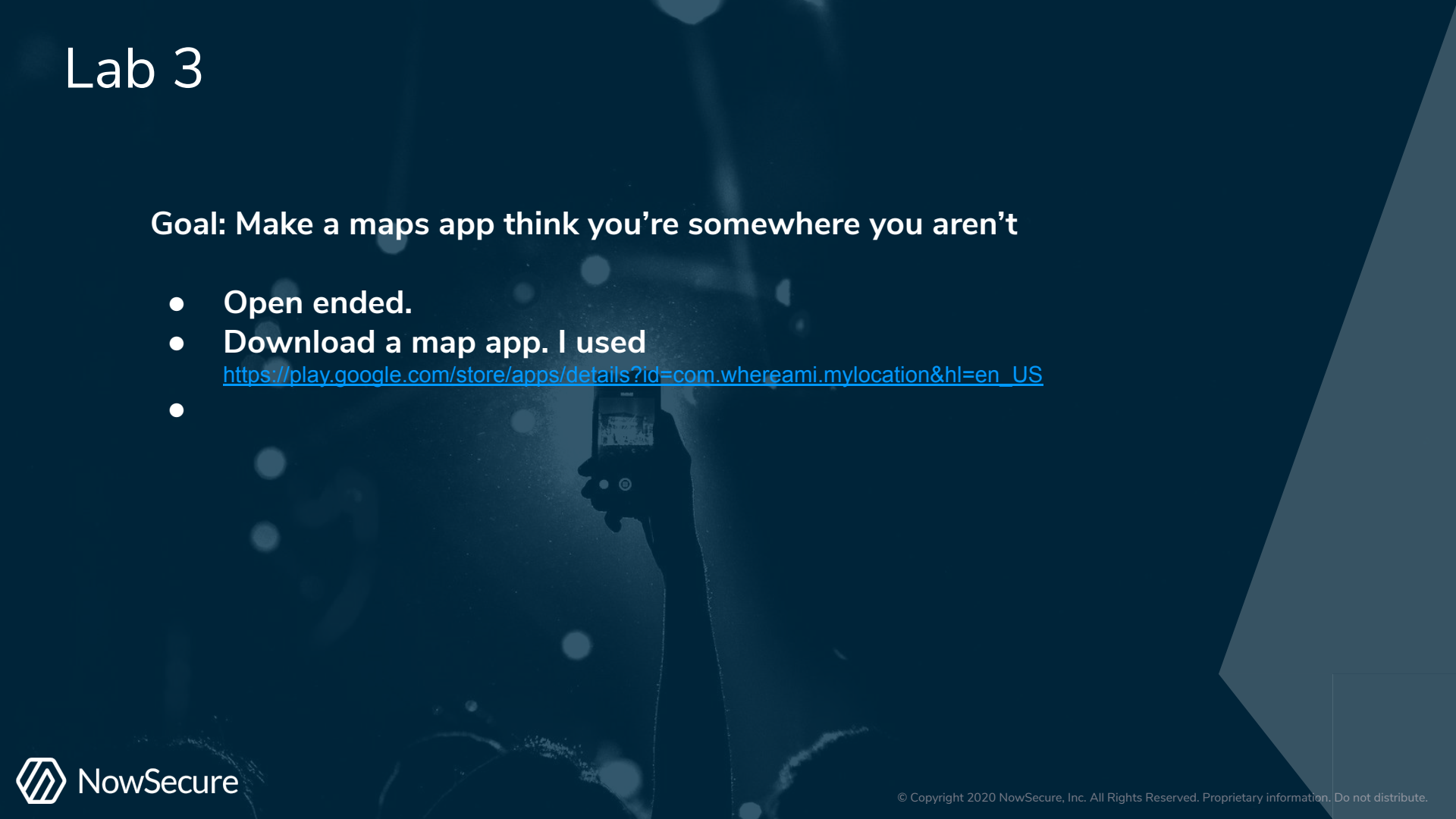
```
19    aescrypt.a.implementation = function(arr1, arr2) {
20      var originalReturn = this.a(arr1, arr2);
21      var decryptedValue = '';
22      for (var i=0; i < originalReturn.length; i++){
23        decryptedValue += String.fromCharCode(originalReturn[i]);
24      }
25      console.log("[+++] Decrypted: " + decryptedValue);
26      return originalReturn;
27    };
28  });
```

NowSecure

# Lab 3

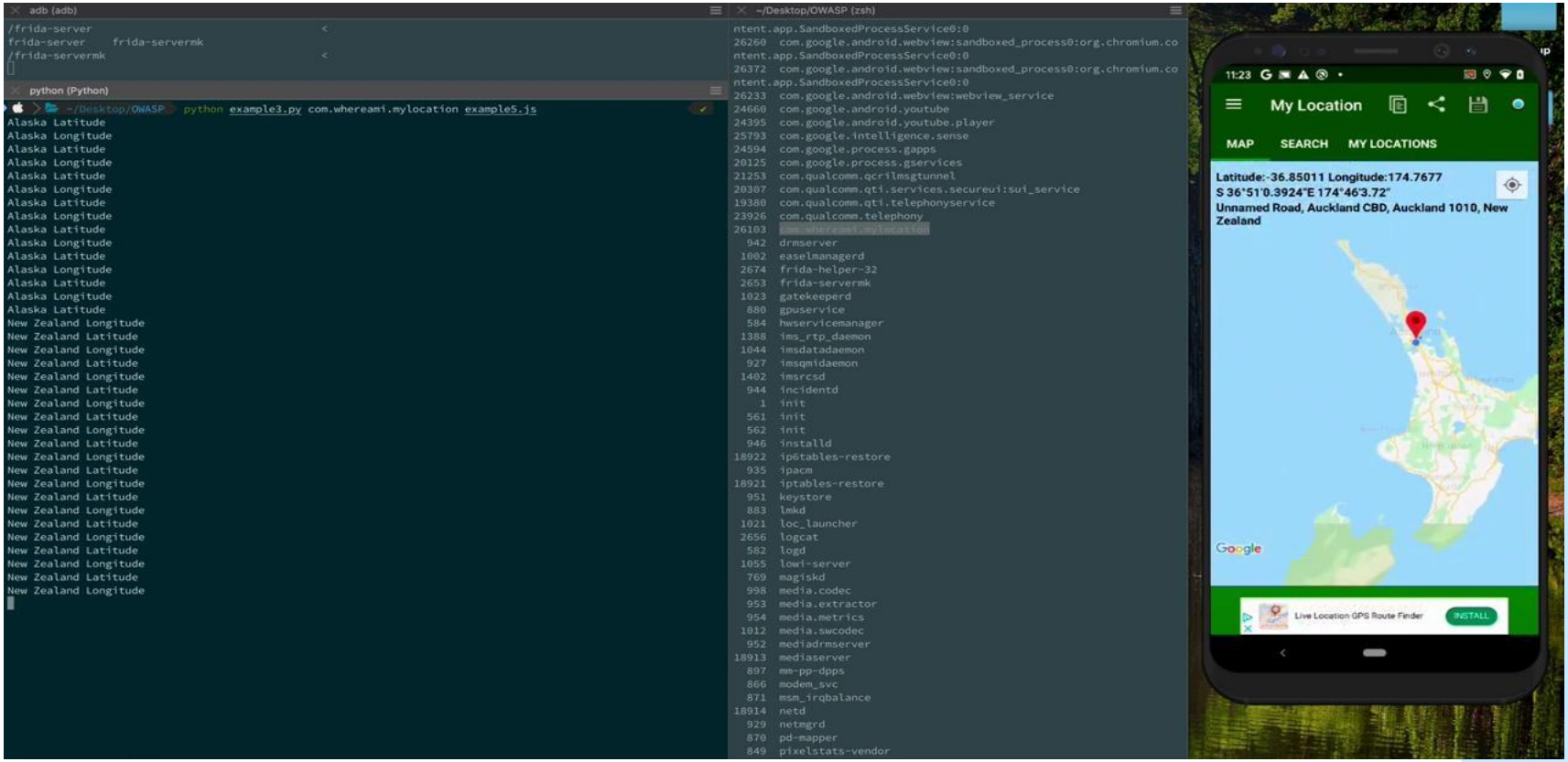
**Goal: Make a maps app think you're somewhere you aren't**

- **Open ended.**
- **Download a map app. I used** https://play.google.com/store/apps/details?id=com.whereami.mylocation&hl=en_US
-



NowSecure

# Demo




NowSecure

# Example 3 - Having Fun with GPS

```
const latitude = 66.160507;
const longitude = -153.369141;
const latitude2 = -36.850109;
const longitude2 = 174.767700;

var requestCount = 0;

Java.perform(function () {
  var appLocation = Java.use("android.location.Location");
  appLocation.getLatitude.implementation = function() {
    if (requestCount < 15) {
      console.log('Alaska Latitude');
      requestCount++;
      return latitude;
    }
    else {
      console.log('New Zealand Latitude');
      return latitude2;
    }

  }
  appLocation.getLongitude.implementation = function() {
    if (requestCount < 15) {
      console.log('Alaska Longitude');
      requestCount++;
      return longitude;
    }
    else {
      console.log('New Zealand Longitude');
      return longitude2;
    }
  }
})
```

NowSecure

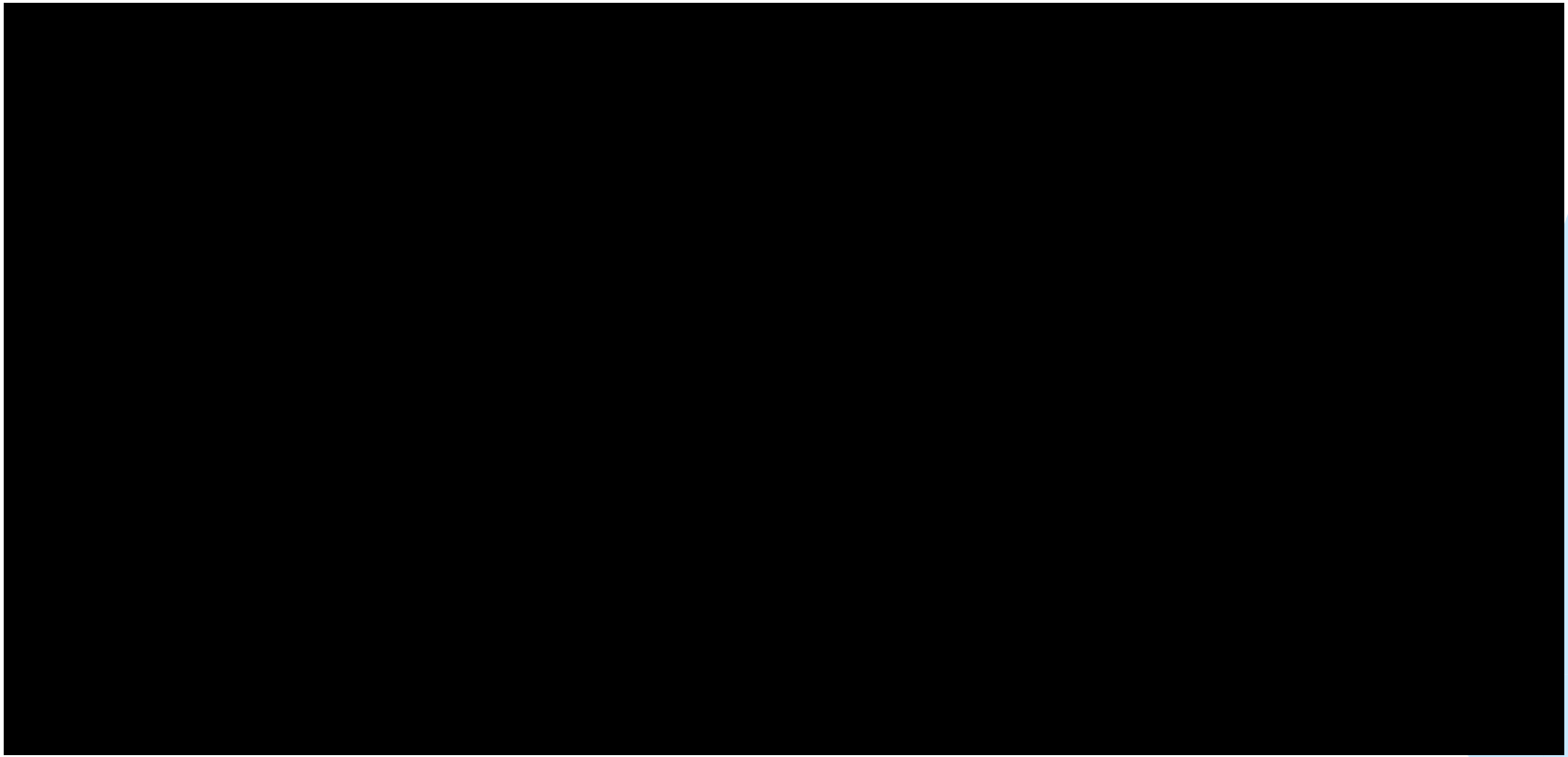
# Lab 4

**Goal: SSL Unpinning**

- **Open ended.**
- **Download an app that's performing SSL pinning (once you can't intercept with Burp or mitmproxy)**
- **Identify the type of pinning implementation**
- **See if you can find a bypass (or write one if you're daring!)**

NowSecure

# Demo



NowSecure

# Example 4 - SSL Unpinning

With SSL Pinning, an application will only connect to a known (or set of known) whitelisted server(s). **Essentially the server's certificate or public key are hardcoded into the application** and validated before establishing a secure SSL/TLS communication channel.

Bypassing SSL Pinning gives you the ability to observe and manipulate traffic sent and received from the server, a powerful tool when reversing mobile apps.

*NOTE: In this example I'm using a freely available script for Pinning Bypass. I did not write this script!*

*https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/*

NowSecure

# Example 4 - What happened?

`TrustManager`s are responsible for managing the trust material that is used when making trust decisions, and for deciding whether credentials presented by a peer should be accepted.

`TrustManager`s are created by either using a `TrustManagerFactory`, or by implementing one of the `TrustManager` subclasses.

```
var CertificateFactory = Java.use("java.security.cert.CertificateFactory");
var FileInputStream = Java.use("java.io.FileInputStream");
var BufferedInputStream = Java.use("java.io.BufferedInputStream");
var X509Certificate = Java.use("java.security.cert.X509Certificate");
var KeyStore = Java.use("java.security.KeyStore");
var TrustManagerFactory = Java.use("javax.net.ssl.TrustManagerFactory");
var SSLContext = Java.use("javax.net.ssl.SSLContext");
```

NowSecure

# Example 4

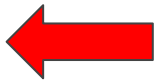
A certificate factory for X.509 must return certificates that are an instance of `java.security.cert.X509Certificate`, and CRLs that are an instance of `java.security.cert.X509CRL`.

```
// Load CAs from an InputStream
console.log("[+] Loading our CA...")
var cf = CertificateFactory.getInstance("X.509");

try {
  var fileInputStream = FileInputStream.$new("/data/local/tmp/cert.crt");
}
catch(err) {
  console.log("[o] " + err);
}

var bufferedInputStream = BufferedInputStream.$new(fileInputStream);
var ca = cf.generateCertificate(bufferedInputStream);
bufferedInputStream.close();

ar certInfo = Java.cast(ca, X509Certificate);
console.log("[o] Our CA Info: " + certInfo.getSubjectDN());
```



1. Visit http://burp on your mobile device.
2. Download the certificate and copy to /data/local/tmp, rename to cert.cer and install.
   a. *This effectively bypasses proper certificate validation*
3. Now update the Frida script with your cert's location/name. (I renamed it again to a .crt but not required.)

NowSecure

# Example 4

```
// Create a TrustManager that trusts the CAs in our KeyStore
console.log("[+] Creating a TrustManager that trusts the CA in our KeyStore...");
var tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
var tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);
console.log("[+] Our TrustManager is ready...");
```

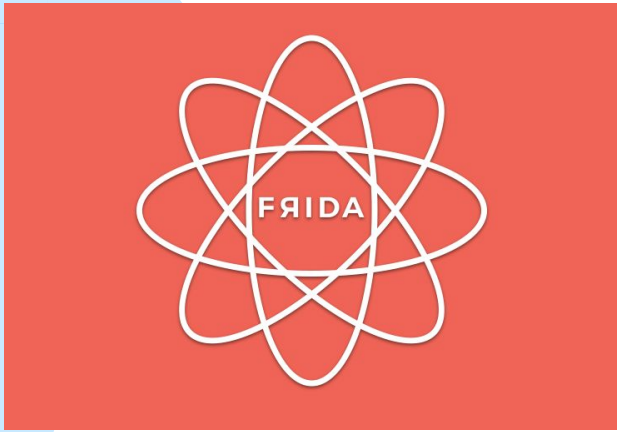
```
SSLContext.init.overload("[Ljavax.net.ssl.KeyManager;", "[Ljavax.net.ssl.TrustManager;", "java.security.SecureRandom").implementation = function(a,b,c) {
  console.log("[o] App invoked javax.net.ssl.SSLContext.init...");
  SSLContext.init.overload("[Ljavax.net.ssl.KeyManager;", "[Ljavax.net.ssl.TrustManager;", "java.security.SecureRandom").call(this, a, tmf.getTrustManagers(), c);
  console.log("[+] SSLContext initialized with our custom TrustManager!");
}
```

javax.net.ssl.SSLContext.init(km, tm, random)

```
public final void init (KeyManager[] km,
                        TrustManager[] tm,
                        SecureRandom random)
```

NowSecure

# Resources for Learning More




https://codeshare.frida.re/


https://bit.ly/2WU8IsG


https://bit.ly/2Z4lK46

Awesome Frida

https://github.com/dweinstein/awesome-frida


https://bit.ly/2zzgb2X

NowSecure

# Q&A - 30 minutes

Ask your questions!

NowSecure

# THANK YOU!

NowSecure