REPLY
SPIKE

MOBILE APP HACKING WITH FRIDA

Nils Milchert | Technical Solutions & Consulting

REPLY
SPIKE

AGENDA

WHAT IS FRIDA?

CHALLENGE

- Essential part of many security assessments is to interact with code that has already been compiled, e.g.
  - Mobile Apps
  - Windows Clients
- Often, it is not enough to assess the functionality available via GUI or CMD-line
- Sourcecode is rarely available
- FRIDA allows working with the code anyway!

WHAT IS FRIDA?

"It's Greasemonkey for native apps, or, put in more technical terms, it's a dynamic code instrumentation toolkit. It lets you inject snippets of JavaScript or your own library into native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX. Frida also provides you with some simple tools built on top of the Frida API. These can be used as-is, tweaked to your needs, or serve as examples of how to use the API."

WHAT IS FRIDA?

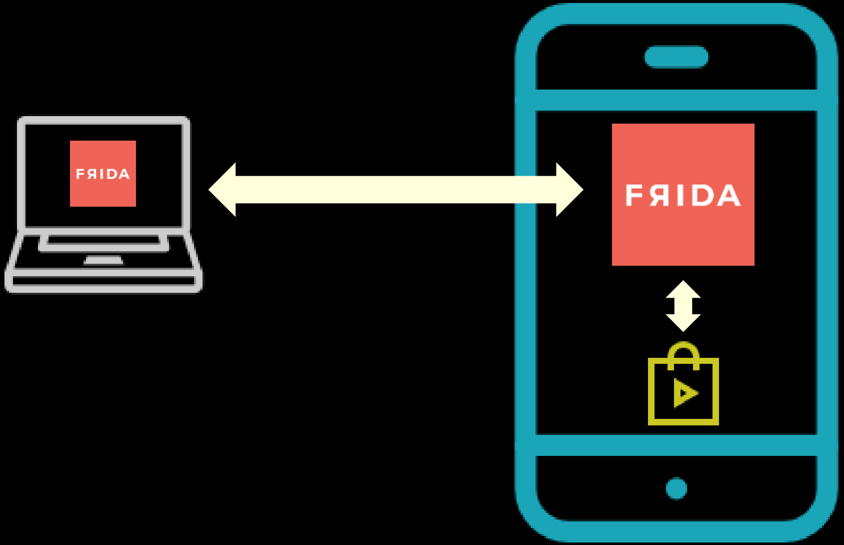
- Framework to dynamically interact with running Code
- Inspect applications:
  - function calls with parameters and return values
  - instance variables
  - GUI components
- Manipulate application logic:
  - change variable values
  - replace function implantations
  - call functions

WHAT IS FRIDA?

- Client – Server Architecture
- Server runs on the device
- Client runs on your PC

- Works either via USB or via network
- Jailbreak/rooting not required
  but strongly recommended

- JavaScript using Java (Android) or ObjC (iOS)



WHAT IS IT FOR?
SCENARIO 1 - PROBLEM

- An application is sending encrypted traffic to a server or is receiving encrypted traffic
- You don‘t have access to the source code
- You (obviously) want to read what the two are chatting about

WHAT IS IT FOR?

SCENARIO 1 - SOLUTION

- Hook the method responsible for encrypting and decrypting traffic
- Make the encryption method displays its parameters
  (which presumably contains the plain text of the message)
- Make the decryption method displays its return value
  (which presumably contains the plain text of the message)
- Profit!

WHAT IS IT FOR?

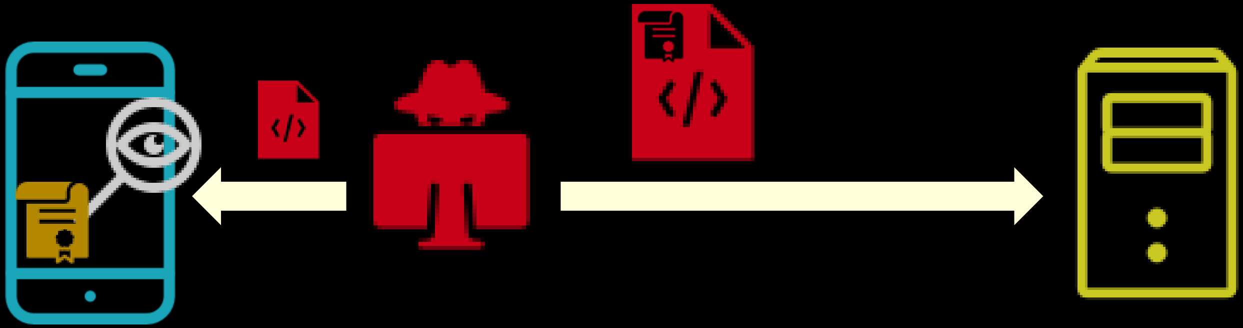
SCENARIO 2 - PROBLEM

- The application adds a signature to every server request
- You assume, there is a super-epic command injection vulnerability
- The server will however reject your payload without correct signature

WHAT IS IT FOR?

SCENARIO 2 - SOLUTION

- Identify the function responsible for singing the payload
- Manually call it with your evil-knievel payload
- Get the return value and submit it to the server
- Profit!



WHAT IS IT FOR?

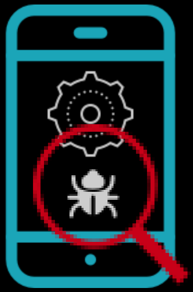
SCENARIO 3 - PROBLEM

- The developer added some juicy debugging feature to the app
- The code is still available in the production app but obfuscated or compiled bytecode
- You can‘t trigger the debugging features because the respective buttons are hidden

WHAT IS IT FOR?

SCENARIO 3 - SOLUTION

- Dump the GUI components
- Watch out for „hidden“ elements
- Make them appear again
- Profit!



INSTALLATION AND BASIC USAGE

INSTALLATION ON YOUR PC

- Requires Python (latest 3.x is highly recommended)
- Should be as straight forward as typing:
- `$ pip install frida-tools`

INSTALLATION AND BASIC USAGE

INSTALLATION ON THE DEVICE

- Requires shell access (preferably SSH)
- Download the `frida-server` for your OS and architecture from: https://github.com/frida/frida/releases
- Make sure that client and server are the same version!
- Copy `frida-server` to the device
- Note: if you are running 64bit, it‘s `frida-server-arm64`

INSTALLATION AND BASIC USAGE

IMPORTANT SWITCHES FRIDA SERVER

`-l <Listen-IP[:Port]>` –	listen on specified IP (and optionally port). By default only listens on USB
`-D`	Detach and become a daemon

INSTALLATION AND BASIC USAGE

IMPORTANT SWITCHES FRIDA CLIENT

`-H <DEVICE_IP[:Port]>`	connect via network to IP [required]
`-n <App-Name>`	Inject <App-Name>
`-p < Process ID>`	Inject <Process ID>
`-l <script.js>`	Execute <script.js> on connect

INSTALLATION AND BASIC USAGE

FIRST RUN

On the Device

```
1)  ./frida-server –l
0.0.0.0 -D
```

On your PC

```
2) frida-ps –H <Device-IP>
```

```
3a)frida –H <Device-IP> -p
<PID>
```

```
3b)frida –H <Device-IP> -n
<APP-Name>
```

FRIDA COOKBOOK

CLASSES

iOS

- `ObjC.classes` contains a reference to all classes

```
var NSString = ObjC.classes.NSString;
NSString.stringWithString_("Hello World");
```

Android

`Java.use()` allows working with classes

```
Java.perform(function () {

const JavaString =
Java.use('java.lang.String');

var exampleString1 = JavaString.$new('Hello
World');

});
```

FRIDA COOKBOOK

CALL METHODS

iOS

```
var NSString = ObjC.classes.NSString;
NSString.stringWithString_("Hello World");

NSString = NSString.uppercased()
```

Android

```
Java.perform(function () {

const JavaString =
Java.use('java.lang.String');

var exampleString1 = JavaString.$new('Hello
World');

exampleString1 = exampleString1.toUpperCase()

});
```

FRIDA COOKBOOK

ENUMERATE ALL CLASSES (IOS)

```
for (var className in ObjC.classes)  // Iterate through all Objective C Classes
    {
        if (ObjC.classes.hasOwnProperty(className))
        {
            console.log(className); // Print classname
        }
}
```

FRIDA COOKBOOK

ENUMERATE ALL CLASSES (ANDROID)

```
Java.perform(function() {                                       // Switch to Java context

Java.enumerateLoadedClasses({                                   // Iterate through all loaded classes
            onMatch: function(className) {                      // callback function for every class

                        console.log(className);

                        },

            onComplete: function() {}                           // empty callback function

            });

});
```

FRIDA COOKBOOK

GET METHODS

iOS

```
var NSString = ObjC.classes.NSString;

NSString.$methods;

NSString.$ownMethods;
```

Android

```
 Java.perform(function () {

 var StringC= Java.use(„java.lang.String");

console.log(Object.getOwnPropertyNames(Object
.getPrototypeOf(StringC)))

        });
```

FRIDA COOKBOOK

LIST ALL METHODS (IOS)

```
for (var className in ObjC.classes)
    {
        if (ObjC.classes.hasOwnProperty(className))
        {
            console.log("[+] Class: " + className);
            var methods = eval('ObjC.classes.' + className + '.$methods');
            for (var i = 0; i < methods.length; i++)
            {
                console.log("\t[-] Method: "+methods[i]);
            }
        }
}
```

FRIDA COOKBOOK

HOOKING A METHOD (IOS)

```
var class = ObjC.classes.EncryptText;
var func = class.encrypt;

Interceptor.attach(func.implementation, {
    onEnter: function(args){
            console.log(args.toString());
            }
    onLeave: function(retval) {
            // here you could potentially overwrite the return value
            }
});
```

FRIDA COOKBOOK

HOOKING A METHOD (ANDROID)

```
Java.perform(function() {
const StringBuilder = Java.use('java.lang.StringBuilder');
StringBuilder.$init.overload('java.lang.String').implementation = function (arg) {
            var partial = "";
            var result = this.$init(arg);
            if (arg !== null) {
                partial = arg.toString().replace('\n', '').slice(0,10);
            }
            console.log('new StringBuilder("' + partial + '");')
            return result;
        }
});
```

BRIDA

MAKING IT SHINY

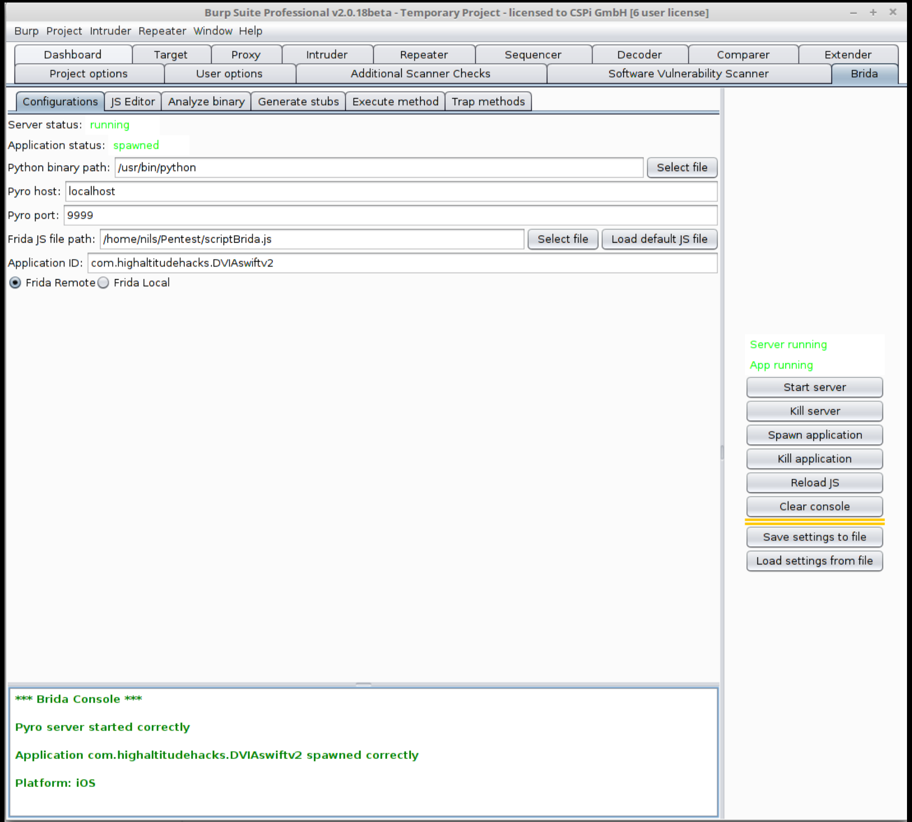
- Plugin for BurpSuite which allows us to work with a GUI
- Setup:
  1. Download BRIDA Release

     (https://github.com/federicodotta/Brida/releases/download/v0.2/Brida_0.2.jar)
  2. Add BRIDA to Burp
  3. Install Pyro4 (pip install pyro4)
  4. Download scriptBrida.js (https://github.com/federicodotta/Brida/releases/download/v0.2/scriptBrida.js)
  5. Open BurpSuite!
  6. Tunnel FRIDA:

```
ssh -L 27042:127.0.0.1:27042 root@192.168.1.7
```

BRIDA

MAKING IT SHINY


Burp Suite Professional v2.0.18beta - Temporary Project - licensed to CSPi GmbH [6 user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender
Project options User options Additional Scanner Checks Software Vulnerability Scanner Brida
Configurations JS Editor Analyze binary Generate stubs Execute method Trap methods
Start server
Kill server
Spawn application
Kill application
Reload JS
Clear console
Save settings to file
Load settings from file

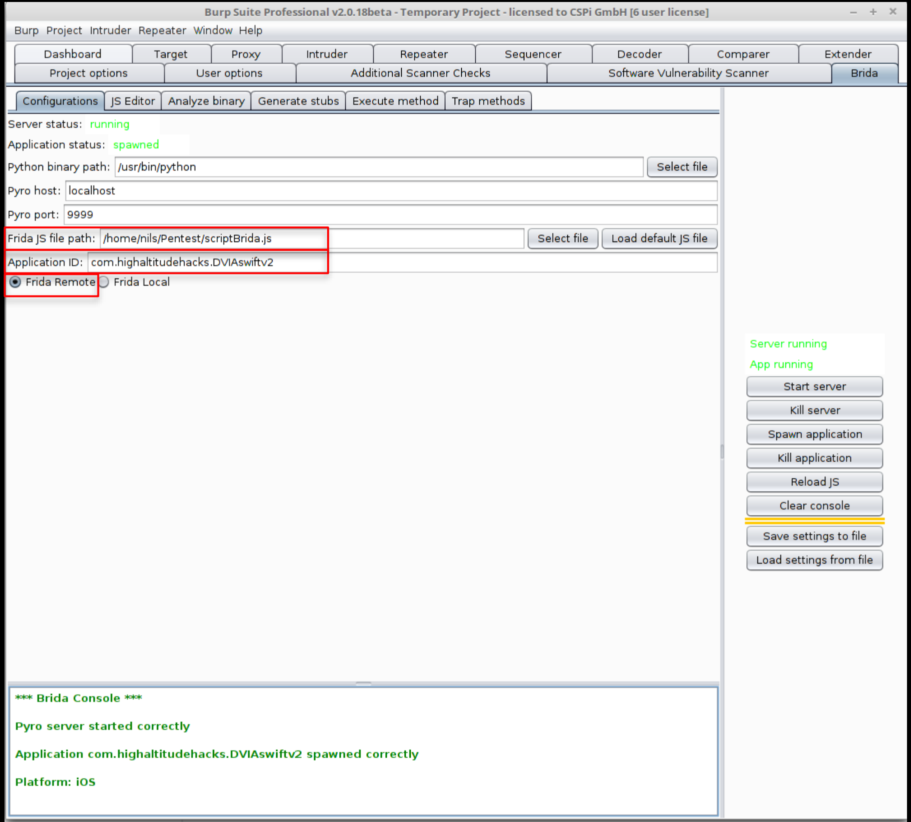
Server status: running
Application status: spawned
Python binary path: /usr/bin/python
Pyro host: localhost
Pyro port: 9999
Frida JS file path: /home/nils/Pentest/scriptBrida.js
Application ID: com.highaltitudehacks.DVIAswiftv2
[✓ Frida Remote] Frida Local

Server running
App running

```
*** Brida Console ***

Pyro server started correctly

Application com.highaltitudehacks.DVIAswiftv2 spawned correctly

Platform: iOS
```

BRIDA

MAKING IT SHINY


Burp Suite Professional v2.0.18beta - Temporary Project - licensed to CSPi GmbH [6 user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender
Project options User options Additional Scanner Checks Software Vulnerability Scanner Brida
Configurations JS Editor Analyze binary Generate stubs Execute method Trap methods
Select file
Select file Load default JS file
Start server
Kill server
Spawn application
Kill application
Reload JS
Clear console
Save settings to file
Load settings from file

Server status: running
Application status: spawned
Python binary path: /usr/bin/python
Pyro host: localhost
Pyro port: 9999
Frida JS file path: /home/nils/Pentest/scriptBrida.js
Application ID: com.highaltitudehacks.DVIAswiftv2
[✓ Frida Remote] Frida Local

Server running
App running

```
*** Brida Console ***

Pyro server started correctly

Application com.highaltitudehacks.DVIAswiftv2 spawned correctly

Platform: iOS
```

DEMO TIME

THANK YOU!

Nils Milchert

Senior Security Consultant

n.milchert@reply.de

+49 171 1759376

SPIKE REPLY