Fun with FRIDA

Dynamic Binary Instrumentation on Android

Workshop by

Michael Helwig
(@c0dmtr1x)

Hendrik Spiegel
(@pspacecomplete)

Dynamic *what?*

- Dynamic Binary Instrumentation (DBI)
- **Modify** and **analyze** a process / binary at runtime
- Do stuff you can do with a debugger **without** a debugger
- **Script it!**

FRIDA

Hello, FRIDA

www.frida.re

- Supports Windows, Linux, Mac, iOS, QNX, and **Android**
- Developed by a nice Norwegian guy (@oleavr)
- Open Source (License: *wxWindows Library Licence*)
- Works by injecting a Javascript runtime (GumJS)

FRIDA helps you to

- Intercept network traffic, crypto calls, file access etc.
- Pentest and fuzz apps
- Overcome protection mechanisms
- Analyse unknown apps (Malware)
- Solve crackmes
- Build your own analysis scripts and frameworks

Demo: Counting in C

```c
int main (int argc, char * argv[])
{
  int i = 0;
  printf("f() is at %p\n", f);
  while (1)
  {
    f (i++);
    sleep (1);
  }
}
```

```c
void f (int n)
{
  printf ("Number: %d\n", n);
}
```

Demo: Counting in C

```
$ ./test
f() is at 0x556bbad846da
PID: 10922
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
```

FRIDA

How does it work?

FRIDA

Instrumentation

- Interceptor API
  - Hooking functions with trampoline JMPs
  - No “real” instrumentation, easily detectable (e.g. code checksums)
  - Still effective and common approach
- Stalker API
  - Dynamic recompilation, modified version gets executed
  - Stealthier
  - Better performance
  - More complicated and error prone

FRIDA

How does it work?

```
$ ./test
f() is at 0x556bbad846da
PID: 10922
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
```

FRIDA

Hooking (Interception)

```
[0x556bbad846da]> pd 12
            ;-- f:
            0x556bbad846da      55              push rbp
            0x556bbad846db      4889e5          mov rbp, rsp
            0x556bbad846de      4883ec10        sub rsp, 0x10
            0x556bbad846e2      897dfc          mov dword [rbp - 4], edi
            0x556bbad846e5      8b45fc          mov eax, dword [rbp - 4]
            0x556bbad846e8      89c6            mov esi, eax
            0x556bbad846ea      488d3df30000.   lea rdi, str.Number:__d ; 0x556bbad847e4 ; "Number: %d\n"
            0x556bbad846f1      b800000000      mov eax, 0
            0x556bbad846f6      e8b5feffff      call sym.imp.printf
            0x556bbad846fb      90              nop
            0x556bbad846fc      c9              leave
            0x556bbad846fd      c3              ret
[0x556bbad846da]>
```

FRIDA

Hooking (Interception)

```
[0x556bbad846da]> pd 13
            ;-- f:
        ,=  0x556bbad846da      e929190000      jmp 0x556bbad86008
        |   0x556bbad846df      90              nop
        |   0x556bbad846e0      90              nop
        |   0x556bbad846e1      90              nop
        |   0x556bbad846e2      897dfc          mov dword [rbp - 4], edi
        |   0x556bbad846e5      8b45fc          mov eax, dword [rbp - 4]
        |   0x556bbad846e8      89c6            mov esi, eax
        |   0x556bbad846ea      488d3df30000.   lea rdi, str.Number:__d ; 0x556bbad847e4 ; "Number: %d\n"
        |   0x556bbad846f1      b800000000      mov eax, 0
        |   0x556bbad846f6      e8b5feffff      call sym.imp.printf
        |   0x556bbad846fb      90              nop
        |   0x556bbad846fc      c9              leave
        |   0x556bbad846fd      c3              ret
[0x556bbad846da]>
```

FRIDA

Try it yourself - C counter demo

- Start the counter
- Run Frida
- Check & modify the script

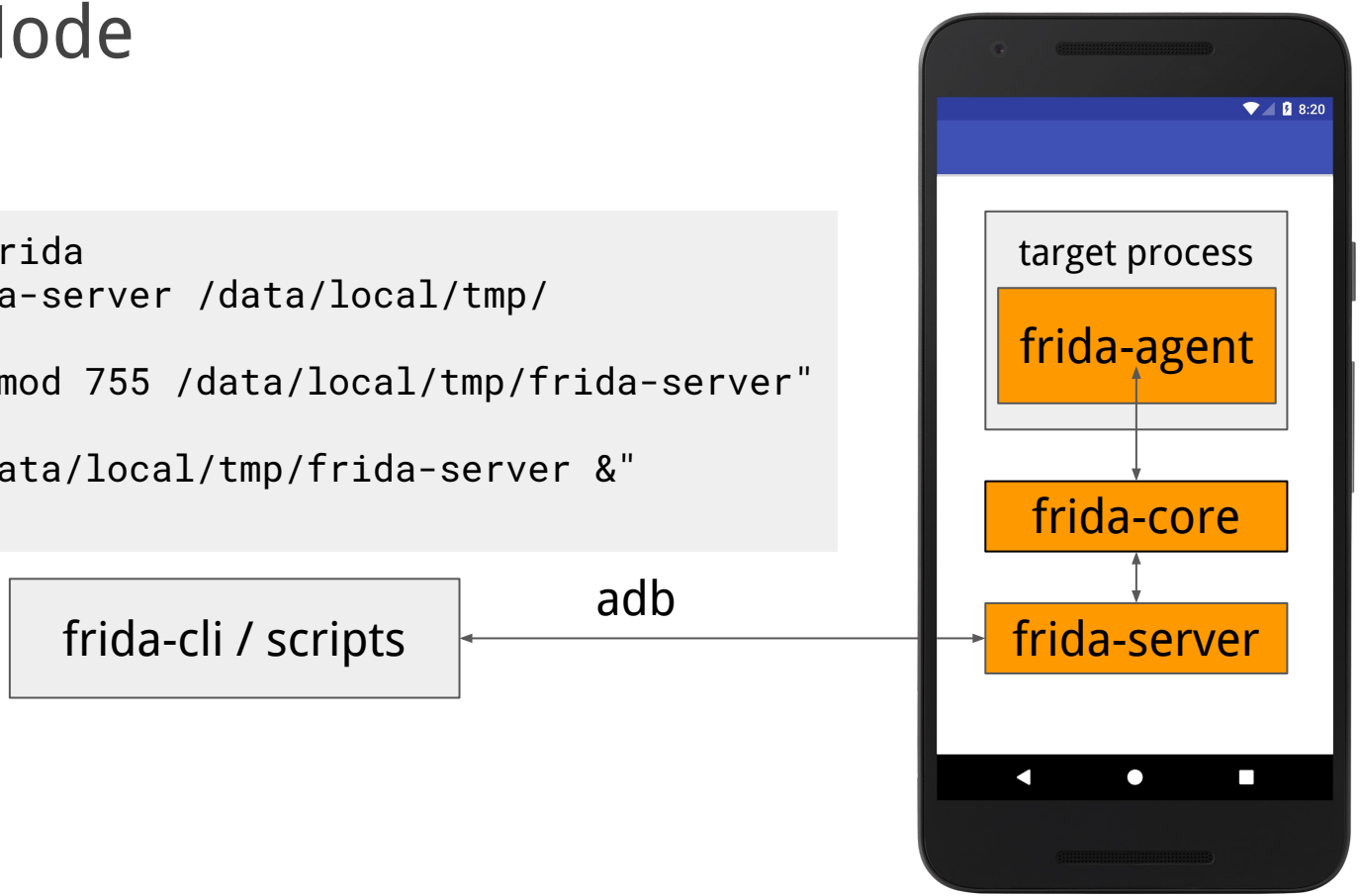
FRIDA

Injected Mode

Installation

```
pip install frida
adb push frida-server /data/local/tmp/
adb root
adb shell "chmod 755 /data/local/tmp/frida-server"

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



FRIDA

Modes of Operation (on Android)

- **Injected** into a process
  - Possible on rooted devices
  - Frida handles the injection

- **Embedded** as shared library (frida-gadget.so)
  - non-rooted devices
  - *Repackaging* and *resigning* required, adapt permissions & smali

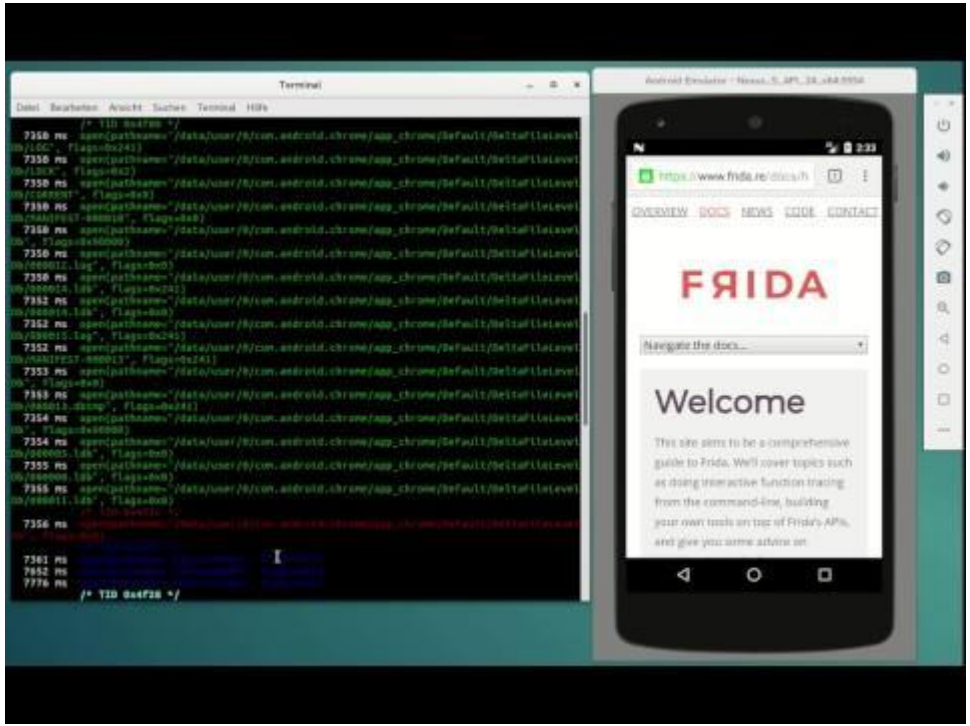
FRIDA

Tracing low level calls

- Trace low level calls to C function
  - E.g. trace `send*, write*, open*, recv*` to track network and file communication
  - trace native library calls easily
- Generates handler script
  - Check __handlers__ directory
  - modify & reuse

FRIDA

Tracing low level calls



FRIDA

Excercise - low level call tracing with Frida (Linux)

- Start Firefox
- Go to your working directory in the shell
- Use frida-trace to trace low level calls in the application (try open, recv, …)
- Afterwards, find the script (__handler__)
- Modify it (add additional logging statements) and run the modified version

FRIDA

Excercise - low level call tracing with Frida (Android)

- Start the emulator
- Start an application
- Go to your working directory in the shell
- Use frida-trace to trace low level calls in the application.
- Find the script, modify it, run the modified version

FRIDA

Frida Java Api

- `Java.perform(fn)` ensure that the current thread is attached to the VM and call `fn`
- `Java.enumerateLoadedClasses(callbacks)` enumerate loaded classes
  - `callbacks` **object**:
    - `onMatch`: called for **each** loaded class
    - `onComplete`: called when **all** classes have been enumerated

FRIDA

Example: Discovering Classes

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

```
frida -U -l ./frida-scripts/list_classes.js
com.example.app
```

```
…
[*] java.util.HashMap
[*] sun.security.jca.ProviderList$3
[*] java.util.Arrays
[*] java.util.regex.MatchResult
[*] [Ljava.util.Comparators$Natural
...
Finished
```

Excercise

- Start an app in the emulator
- Create an enumeration script
- Log all loaded classes to the console

FRIDA

Interacting with frida: Messages

- Send messages between processes
  - python wrapper ⟷ injected javascript
- You can send messages from the app running on Android back to the Python script in your console
- Useful if you want to write stuff to a file or store it in a local database directly (without having to transfer it from the emulator)

FRIDA

Interacting with Frida: Messages

```python
from __future__ import print_function
import frida, sys

#Arguments
script_name = sys.argv[1]
app_name = sys.argv[2]

#Message function: output messages from frida
def on_message(message, data):
    print("[*] {0}".format(message['payload']))

#Read script from file
with open(script_name, 'r') as scriptfile:
    script_data=scriptfile.read()

#Attach
process = frida.get_usb_device().attach(sys.argv[2])

#Create frida script and register message function
script = process.create_script(script_data)
script.on('message', on_message)

#Start
print('[*] Loading script...')
script.load()
sys.stdin.read()
```

```java
Java.perform( function() {
    Java.enumerateLoadedClasses(
        {
          "onMatch":function(match){
              send(match);
           },
           "onComplete":function(){
              send("Finished");
           }
        }
    );
});
```

FRIDA

Overwriting various stuff with Frida's Java API

- Frida provides a javascript wrapper for Java objects and classes
- Easily intercept / hook / modify:
  - instance variables
  - instance methods
  - static class variables
  - static class methods
  - private and public methods
- Create new objects (`$new`)
- Find existing objects on the heap (`Java.choose`)

FRIDA

Hooking Methods Example

```
01  // Get wrapper for class
02  main = Java.use("com.example.app.MainActivity");
03
04  // Overwrite function definition
05  main.myFunction.implementation = function(arg1,arg2){
06
07        console.log(“arg1: “ + arg1 + “, arg2: “ + arg2)
08
09        //call original function with modified args
10        return this.myFunction(“foo”,”bar”);
11  }
```

FRIDA

Reminder: Disassembling APKs

You still might want to disassemble an APK to identify methods and classes

- APK = ZIP
- E.g. APKTOOL to extract Manifest and decompile to smali
- Or just throw DEX into a disassembler / decompiler (maybe use dex2jar before)
- Various decompilers for Java available, BytecodeViewer has a nice GUI
- For native libraries: Usual stuff like Radare2, IDA, Binja, etc.
- Deobfuscation: E.g. Simplify, dex2jar & Deobfuscator

FRIDA

Try it yourself - overwriting methods with Frida

- Open “Fun With Frida 1” in the emulator
- You see a bunch of “Hello, World” strings
- **Task: Convert all “Hello, World” strings to “Hello, Frida”**
- Hints:
  - Disassemble the APK to make it more realistic - or get the source from github (https://github.com/mhelwig/funwithfrida1)
  - Get classes via my_class = Java.use(...)
  - You can overwrite variables with the “`value`” property (`my_class.my_var.value = ...`)
  - You can overwrite methods with `implementation`
  - You can overwrite overloaded methods with `overloaded`
  - You will find stuff on the heap
  - **You don’t need to dive into the native library (we will get to that later)**
  - https://www.frida.re/docs/

FRIDA

Try it yourself - Make the interception interactive

- Open “needleRemover” in the emulator
- Task: When a new text is set in the text activity intercept the call and interactively replace the message
- Hints:
  - Use the skeleton in `tasks/2`
  - For an example how sending/receiving works check:
    - `/home/frida/tasks/enumeration/*`
  - `Messages: https://www.frida.re/docs/messages/ : Blocking receives in the target process`

FRIDA

Timing

- When you attach frida to a running process, the app has already started
- Frida can also hook into Zygote and spawn the app for you
- Inject code before the app has started
- Start frida with “-f” option
- Frida pauses at startup to give you the chance for input. This easily runs into a timeout. Use “--no-pause” to skip this.

FRIDA

Root detection

```
Java.perform(function() {
    main = \
Java.use("net.codemetrix.funwithfrida2.MainActivity");

    main.detectRoot.implementation = function(){
        return false;
    }
});
```

App Code (MainActivity)

```
private boolean detectRoot() {
        String[] a = new String[8];
        a[0] = "/system/xbin/su";
        a[1] = "/system/app/Superuser.apk";
        a[2] = "/system/xbin/daemonsu";
        a[3] = "/system/etc/init.d/99SuperSUDaemon";
        a[4] = "/system/bin/.ext/.su";
        a[5] = "/system/etc/.has_su_daemon";
        a[6] = "/system/etc/.installed_su_daemon";
        a[7] = "/dev/com.koushikdutta.superuser.daemon/";

       int i = 0;
       while(i < a.length)
       {
            if (new java.io.File(a[i]).exists())
            {
                return true;
            }
            i++;
       }
       return false;
}
```

FRIDA

Try it yourself - circumventing root detection

- Open “Fun with Frida2” in the emulator
- Start the app - what is happening?
- Hint:
  - The name of the root detection method is “detectRoot”
  - When should the method be overwritten? (Check the “-f” option of frida)

FRIDA

A word on native methods

- You can also use Frida to modify native methods
- Didn’t work (on Android) when we were preparing the workshop, but worked in the past (and could work again in the future)
- You can still locate loaded module addresses and overwrite arbitrary stuff in memory (so you can still do it but it takes more effort)
- Check the “Module” und “Interceptor” in the api docs

A word on native methods

- This is what hooking native methods could look like (compare frida-trace):

```
var functionPtr =
Module.findExportByName("libnative-lib.so","Java_net_codemetrix_funwithfrida_MainActivity_stringFromJNI");

Interceptor.attach(functionPtr, {
        onEnter:function(args) {
             console.log("[*] On Enter called");
        },
        onLeave:function(retval){
            var stringClass = Java.use("java.lang.String");
            var stringInstance = stringClass.$new("Hello, Frida");
            retval.replace(stringInstance);
        }
});
```

FRIDA

Overwriting strings in memory

- Another solution to change the native method in FunWithFrida1:
  - Determine Module base address
  - Determine String address from offset
  - Change memory protection and overwrite
  - x86 solution:

```
base_address = Module.findBaseAddress("libnative-lib.so");
string_offset = parseInt("0x00020de0",16);          //Determined via disassembler
string_address = parseInt(base_address,16) + string_offset;

pointer = new NativePointer(string_address);
Memory.protect(pointer,32,"rw-");                   //Change memory protection
Memory.writeUtf8String(pointer,"Hello, Frida");
```

FRIDA

Overwriting strings in memory

Want to try it?

- What's the offset of the string in your native library?
- Use `Module.findBaseAddress` to find the base address of the module in memory
- Calculate the position of the string
- Use a `NativePointer` to access the memory with frida
- etc.

FRIDA

HTTPS Interception

Until Android **6.0**

- Install your Burp Certificate in Android
- Set Burp as Proxy
- Intercept traffic
- Done!

FRIDA

HTTPS Interception

Since Android **7.0**

- Install your Burp Certificate in Android
- Set Burp as Proxy → Certificate is ignored
- ToDo:
  - Do not use 7.0
  - Fallback to old behaviour

FRIDA

HTTPS Interception on Android 7.0

```
class NetworkSecurityConfig

...

public static final Builder getDefaultBuilder(int targetSdkVersion, int
targetSandboxVersion) {

...
if (targetSdkVersion <= Build.VERSION_CODES.M) {
// User certificate store, does not bypass static pins.
  builder.addCertificatesEntryRef(
      new CertificatesEntryRef( UserCertificateSource.getInstance(), false));

}
...
```

FRIDA

HTTPS Interception (with Pinning)

- New with Android 7.0
- XML based
- Easy to use (for developers)
- Defeat
  - Get APK
  - Unpack
  - Modify
  - Rebundle/Sign
  - Redeploy

FRIDA

HTTPS Interception (with Pinning)

- New with Android 7.0
- XML based
- Easy to use (for developers)
- Defeat
  - ~~Get APK~~
  - ~~Unpack~~
  - ~~Modify~~
  - ~~Rebundle/Sign~~
  - ~~Redeploy~~

Task: use Frida

- install `tasks/3/pinning.apk` on the emulator
- Go to the SSLActivity
- Hint: use `adb log` or `pidlog` to see what is going wrong

FRIDA

Links & Stuff

Frida Project Page
https://www.frida.re

Curated list of resources
https://github.com/dweinstein/awesome-frida

OWASP Mobile Security Testing Guide
https://github.com/OWASP/owasp-mstg

FRIDA