Frida

Dynamic Intrumenttion* framework

Abhijeet Sawant

Special Thanks to @oleavr

* All the speeling mistakes are intentional, to keep everybody focus :)

Expectation

- **Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime** -- Chinese Proverb

Frida

- Dynamic instrumentation framework
  - What is instrumentation?
  - Where to use this? Is it really required ?
- Allows injection of JS or our own libraries into Native application on **Windows, macOS, GNU/Linux, iOS, Android, and QNX**
- We will cover generic ways, so that you can use across all the supported platform
- Provides some sample tools build on top of Frida API i.e frida-trace, frida-ps etc

Some Internal's of frida

Digging Dipper*

- Frida is written in C known as Gum, its also call frida-core
- Using GumJS (Google's V8 engine), it expose frida core to JS and then with various bindings it allows scripting with help of python, Node.js, Swift, .NET, Qml etc
- A GumJS allows bi-directional communication channel between JS running inside debugee and our script

Pre-requistee*

1. Should know basics of Python, JS
2. Basic understanding of operating system
3. Learning curve : stiff

Installation

- Python 3.x recommended
- `pip install frida`
- or for lazy minds, download pre-compiled binary from here
- We will only focus on macOS, for more detail instrunction's kindly visit Frida Installation Guide

Mode of Operations

Frida supports **three** operation mode, those are listed below

Injection:-

- Suitable for security professional
- loads frida-core as shared library into debugee process i.e **GumJS**
- allows you to list installed packages (*frida-ps*), connected devices (*frida-ls-devcies*), running process and instrument them
- *frida-server* runs on **localhost:27042**
- Requires jailbroken /or rooted device

Embedded

- Frida provides frida-gadget, a shared library to include inside our application and compiled
- **Doesn't require jailbroken devices** but requires access to **source code**

Preloaded

- Load frida-gadget with help of *LD_PRELOAD or DYLD_INSERT_LIBRARIES*
- For above two methods (**Will cover in part 2**)
  - Android :- Android Injection
  - iOS :- iInject , insert_dylib

Frida command line tools

- **frida-ps** : list all the process running, installed application (in case of mobile)

```
frida-ps -i -a
frida-ps -D ID / --device=ID
frida-ps -U (USB)
frida-ps -R (Remote)
```

- **frida-discover** :- presents summary of all the calls with count and there respective library

```
frida-discover -n NAME / -p PID
```

- **frida-trace** :- Track function calls and generates script to be able to play around inside **handler** directory

```
frida-trace -I MODULE / -i func-name
```

- **frida-kill** :- Kill given Process
- **frida-ls-process** :- list devices
- **frida** :- Most important, an interactive frida with javascript runtime, autocomplete

Demo Time

Python binding for frida

skaleton* script

```
import frida
import sys
script_js = ""
with open("FILENAME","r") as scriptFile:
    script_js = scriptFile.read()
def on_message(message,data):
    print(message)  # callback to receive message
session = frida.get_local_device().attach("demo4")
script = session.create_script(script_js)
#callback registration
script.on('message',on_message)
script.load()       # script loading
sys.stdin.read()    # to keep script running
```

Demo Time

Javascript API

Let's start from basic

- `ptr(s)` : short hand for new NativePointer(s), where s is string
- `send(message,[data])` : data should be ArrayBuffer and message must be serializable JSON
- `recv([type, ] callback)` : reciver
- `hexdump(target, [options])` : hexdump from given target or NativePointer
- `console.[log|warn|error](line)` :
- `rpc.exports` : Object containg functions or variable, available as **export** to your binding application
- `Frida.version and Frida.heapSize` : property

Demo Time

Process

Let's focus on Process. Assume `Process` as **prefix** for below API i.e `Process.arch``

- Let's explore Process Object
- `isDebuggerAttached()`
- `getCurrentThreadId()`
- `enumerateThread(callbacks)` : here callbacks is object with onMatch: function(thread) and onComplete: function ()
- `[find|get]ModuleByAddress(address)` and `[get|find]ModuleByName(name)`
- `enmerateModules(callbacks)` : onMatch: function(module) and onComplete
- `enumerateRanges(protection|specifier, callbacks)` : protection is 'rw-' or 'rwx' and callbacks is again onMatch: function(range) and onComplete()
- Sync vs Async version

Demo Time

Module

It's now Module time. Again kindly assume **prefix** as `Module` for below API

- Let's explore Module Object
- `enumerateImports(name,callbacks)` : callbacks is onMatch(imp) and onComplete()
- `enumerateExports(name,callbacks)`
- `ensureInitialized(name)`
- `findExportByName(module|null, exp)` : return address of export name given in exp from Module, Module can be null (costly search)

Demo Time

NativeFunction

- `new NativeFunction(address,returnType,argType[,abi])` :
  - address is NativePointer
  - returnType is any valid type

NativeCallback

- `new NativeCallback(func, returnType, argTypes[, abi])` : returns nativecallback
  - arguments similar to NativeFunction

Interceptor

Intercept native functions, Assume `Interceptor` as **prefix**

- `attach(target,callbacks)`: here callbacks is object with onEnter(args) and onLeave(retval)
  - "retval" is NativePointer
  - "target" is NativePointer as well
  - **this** variable
- `replace(target, replacement)`: replace function at target, replacement is NativeCallback
- `revert(target)` : revert the target to original state

Frida Code Repo

Code Repo

- https://codeshare.frida.re/
- Some tools based on frida
  - https://github.com/snooze6/FiOS (Thanks to Kaleem)
  - https://github.com/sensepost/objection

References

Reference

- https://www.frida.re/docs/
- https://github.com/frida/frida-python
- https://www.frida.re/docs/presentations/