Getting fun with Frida

Turbo Talk – Ekoparty

Nahuel Riva

October 2016
Getting fun with Frida

De party

All de night
Agenda
Agenda

- Intro
  - What’s DBI?
  - Why do we need DBI?
  - How do I perform DBI? (frameworks)

- Frida
  - What’s Frida?
  - Why would I need Frida?
    - Differences with other frameworks
  - How do I use Frida
    - API
      - Interceptor
      - Stalker
    - Tools based on Frida

- Demos
Intro
Intro – What’s DBI?

- Definition taken from: http://uninformed.org/index.cgi?v=7&a=1&p=3

- “Dynamic Binary Instrumentation (DBI) is a method of analyzing the behavior of a binary application at runtime through the injection of instrumentation code. [...] makes it possible to gain insight into the behavior and state of an application at various points in execution.”
Intro – What’s DBI?

- Instrumentation code executes as part of the normal instruction stream after being injected

- Instrumentation code will be entirely transparent to the application that it's been injected to

- Instrumentation code executes at runtime
Intro – Why do we need DBI?

• As an alternative
  • Debuggers
  • API hooking engines

• Evolution
  • More complex tasks to achieve (profiling, taint analysis, detection of possible bugs)
Two main DBI frameworks:

- **PIN**: proprietary framework written in C/C++. Works on Windows/Linux/OSX/Android and i386/AMD64. 
  

- **DynamoRIO**: originally a proprietary framework then open sourced (BSD). Created by HP (Dynamo optimization system) and MIT (RIO research group). Works on Windows/Linux and i386/AMD64.
  
Intro – How do I perform DBI? (frameworks)

• In both cases, you write a Pin/DynamoRIO tool using C/C++ language and inject C/C++ code

• Compile the Pin/Dynamo tool as a .dll/.so

• Inject the library into the target process using a command-line tool/GUI application
Intro – How do I perform DBI? (frameworks)

Pintool example (source/tools/ManualExamples/inscount0.cpp):

```c
int main(int argc, char * argv[])
{
    // Initialize pin
    if (PIN_Init(argc, argv)) return Usage();

    OutFile.open(KnobOutputFile.Value().c_str());

    // Register Instruction to be called to instrument instructions
    INS_AddInstrumentFunction(Instruction, 0);

    // Register Fini to be called when the application exits
    PIN_AddFiniFunction(Fini, 0);

    // Start the program, never returns
    PIN_StartProgram();

    return 0;
}
```
ofstream OutFile;

// The running count of instructions is kept here
// make it static to help the compiler optimize docount
static UINT64 icount = 0;

// This function is called before every instruction is executed
VOID docount() { icount++; }

// Pin calls this function every time a new instruction is encountered
VOID Instruction(INS ins, VOID *v)
{
    // Insert a call to docount before every instruction, no arguments are passed
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}

KNOB<string> KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool",
"o", "inscount.out", "specify output file name");
// This function is called when the application exits
VOID Fini(INT32 code, VOID *v)
{
    // Write to a file since cout and cerr maybe closed by the application
    OutFile.setf(ios::showbase);
    OutFile << "Count " << icount << endl;
    OutFile.close();
}
For example, Pin can be executed as follow:

- `pin.bat -t pintool.dll [pintoolargs] --program.exe [programargs]`
- `pin.bat -pid<programpid> -t pintool.dll [pintoolargs]`
Intro – How these frameworks work?

• JIT compiler
  • Input: binary code
  • Output: equivalent code with introspection code
  • The code is generated only when it is needed

• The only code that is executed is the code generated by the JIT compiler

• The original code remains in memory just as a reference but it is never executed
Frida
Frida – What’s Frida?

- Dynamic instrumentation toolkit

- Scriptable
  - Execute **Javascript** programs inside another process. It uses V8 and Duktape and JavaScriptCore (deprecated) engines.

- Multi-platform and multi-arch
  - Windows/Mac/Linux/Android/iOS/QNX – i386/AMD64/ARM/ARM64

- It has bindings for Python, .NET, C and Node.js

- Open-source (LGPL v2)
Frida – Why would I need Frida?

• For reverse engineering in general
  • Dynamic binary instrumentation
  • Debugging

• To develop introspection tools **very quickly** to help you in the RE process
Frida – Pros & Cons against other frameworks

- Pros
  - It has bindings for other languages like .NET, Python, C
  - No need to compile the tool
  - Rapid tool development
  - Continuous development (new features and bug fixing)

- Cons
  - Less mature than other DBI frameworks (contains bugs)
  - Lack of some functionality
  - Less granularity than other frameworks
How do I use Frida?
Frida – How do I use Frida?

• First, you need to install it:
  
  • Windows:
    ▪ C:\Users\travesti>pip install frida

  • Linux:
    ▪ travesti@palermo:~$ sudo pip install frida

  ▪ Then …
Frida – How do I use Frida?

- As easy as this:

  ```python
  >>> import frida
  >>> session = frida.attach("notepad.exe")
  >>> print([x.name for x in session.enumerate_modules()])
  
  [u'notepad.exe', u'ntdll.dll', u'kernel32.dll', u'KERNELBASE.dll', u'ADVAPI32.dll',
   u'RPCRT4.dll', u'GDI32.dll', u'USER32.dll', u'KERNELBASE.dll',
   u'ole32.dll', u'OLEAUT32.dll', u'VERSION.dll', u'IMM32.DLL',
   u'MSCTF.dll', u'actuser.dll', u'accdetoured.dll', u'msvcp60.dll', u'CRYP
   TBASE.dll', u'uxtheme.dll', u'dwmapi.dll', u'CLBCatQ.DLL', u'frida-agent-64.dll'
   , u'DNSAPI.dll', u'WS2_32.dll', u'NSI.dll', u'WINMM.dll', u'PSAPI.DLL', u'nt
   nt{/}a.dll', u'WLDAP32.dll']
  ```
Frida – Architecture

Your tool

- Script object
- frida-node
  - JavaScript & C++
- frida-core
  - (C API, statically linked)
  - Vala & C

Target app

- frida-agent
  - Shared library injected by frida-core
  - Vala & C
- gumjs
  - C++ (Vala)
- frida-gum
  - C & arm

P2P D-Bus across platform-specific transport

Bi-directional exchange of JSON messages:
E.g. your-script.js calls send(1234), which transmits:

```
   { "type": "send", "payload": 1234 }
```

Unhandled exception transmits:

```
   { "type": "error", ... }
```
Frida – JavaScript API

- Its Javascript API has different components to interact with a process (http://www.frida.re/docs/javascript-api):
  - console
  - Process
  - Module
  - Memory
  - Thread
  - Socket
  - File
  - Instruction
Frida – JavaScript API - Console

- **console**: used for output.
  - `console.log(line)`
  - `console.warn(line)`
  - `console.error(line)`
Frida – JavaScript API - Process

- **Process**: functions and properties used to interact with a process.

  - Process.arch, Process.platform
  - Process.isDebuggerAttached
  - Process.enumerateThreads(callbacks)
  - Process.findModuleByAddress(address)
  - Process.findModuleByName(name)
  - Process.enumerateModules(callbacks)

- [...]
Frida – JavaScript API - Module

- **Module**: used to interact with modules residing in the process.
  
  - `Module.enumerateImports(name, callbacks)`
  - `Module.enumerateExports(name, callbacks)`
  - `Module.enumerateRanges(name, protection, callbacks)`
  - `Module.findBaseAddress(name)`
  - `Module.findExportByName(moduleOrNull, exp)`
  - `[...]
Frida – JavaScript API - Memory

- **Memory**: used to interact with memory pages residing in a given process.

  - Memory.scan(address, size, pattern, callbacks)
  - Memory.alloc(size)
  - Memory.copy(dst, src, n)
  - Memory.protect(address, size, protection)
  - Memory.read*/write*
  - MemoryAccessMonitor (monitor read/write/execute)

- […]
Frida – JavaScript API - Thread

- **Thread**: used to interact with threads from a process.
  - Thread.backtrace([context, backtracer])
  - Thread.sleep(delay)
**Socket** used to handle sockets.

- `Socket.type(handle)`
- `Socket.localAddress(handle)`
- `Socket.peerAddress(handle)`
Frida – JavaScript API - File

- **File**: used to handle file I/O.
  
  - `File(filePath, mode)`
  - `write`
  - `read`
  - `flush`
  - `close`
Frida – JavaScript API - Instruction

- **Instruction**: used to get information about a given instruction from process’s code.

  - Instruction.parse(target)
Frida – Interceptor/Stalker

• Frida has two main components exposed through its API:

  • **Interceptor**
    • Normal operation mode (hooking)
    • No stealthiness

  • **Stalker**
    • Instrumentation per-se
    • Stealth (kind of)
    • Lack of functionality (CALL/RET)
    • More details: [https://medium.com/@oleavr/anatomy-of-a-code-tracer-b081aad0b0df8](https://medium.com/@oleavr/anatomy-of-a-code-tracer-b081aad0b0df8)
Frida – How do I use Interceptor?

- Interceptor example:

```javascript
def main(target_process):
    heapalloc_rva = getExportedFunctionRva('RtlAllocateHeap', 'ntdll.dll')

    session = frida.attach(target_process)
    script = session.create_script(""
    var RtlAllocateHeapAddr = Module.findBaseAddress('ntdll.dll').add(0x08);
    console.log('HeapAlloc address: ' + RtlAllocateHeapAddr.toString());

    console.log('>> Hooking ntdll!RtlAllocateHeap...');

    Interceptor.attach(RtlAllocateHeapAddr, {
        onEnter: function (args) {
            console.log('[+] RtlAllocateHeap called from ' + this.returnAddress.toString());
            console.log('[+] HeapHandle: ' + args[0].toString());
            console.log('[+] Flags: ' + args[1].toString());
            console.log('[+] Size: ' + args[2].toString());
        },
        onLeave: function (args) {
            console.log('[+] Returned address: ' + args[0].toString());
        }
    });

    """

    script.on('message', on_message)
    script.load()
    raw_input('[] Press <Enter> at any time to detach from instrumented program.\n\n')
    session.detach()
```

Frida – Interceptor example output
Frida – Interceptor at low level (API hook)
Frida – Interceptor stub
Frida – Stalker

- Stalker example:

```javascript
function StalkerExample()
{
    threadIds = [];
    Process.enumerateThreads(
        onMatch: function (thread)
        {
            threadIds.push(thread.id);
            console.log(“Thread ID: “ + thread.id.toString());
        },
        onComplete: function ()
        {
            threadIds.forEach(function (threadId)
            {
                Stalker.follow(threadId,
                {
                    events: {call: true},
                    onReceive: function (events)
                    {
                        console.log(“onReceive called.”);
                    },
                    onCallSummary: function (summary)
                    {
                        console.log(“onCallSummary called.”);
                    }
                });
            });
        });
    }
}
StalkerExample();
```
Frida – How Stalker works?

• Stalker at low level:

```
00001000 push ebp
00001001 mov ebp, esp
00001003 call 00001234
00001008 mov esp, ebp
0000100A pop ebp
0000100B ret
```

```
00004000 call log_handler
00004005 push ebp
00004006 call log_handler
0000400B mov ebp, esp
0000400D call log_handler
00004012 push 00001008 ; CALL stack side-effect
00004013 push 00001234 ; arg 2/2: branch target
00004014 push exec_ctx ; arg 1/2: execution context
00004019 call gum_exec_ctx_replace_current_block_with
```

• Hint: See `gum_exec_ctx_obtain_block_for` in frida-gum/gum/backend-x86/gumstalker-x86.c
Tools based on Frida
Tools based on Frida

• **frida-cli**: command line interpreter which emulates an IPython console for rapid prototyping and easy debugging.
Tools based on Frida

- **frida-ps**: command line tool for listing processes.
Tools based on Frida

- **frida-trace**: command like tool to dynamically trace function calls.
Tools based on Frida

- **frida-heap-trace**: trace RtlAllocateHeap, RtlFreeHeap and RtlReAllocateHeap function calls and arguments and log them to a file.

  - Combine it with Villoc to create a map for all the heap movements

  - [https://github.com/poxyran/misc/blob/master/frida-heap-trace.py](https://github.com/poxyran/misc/blob/master/frida-heap-trace.py)
Tools based on Frida

- [https://github.com/wapiflapi/villoc](https://github.com/wapiflapi/villoc)
Tools based on Frida

• **fridump**: Universal memory dumper tool. Aimed to dump accessible memory regions from any platform supported by Frida.

  https://github.com/Nightbringer21/fridump
Tools based on Frida
Tools based on Frida

- **frida-extract**: FridaExtract is a [Frida.re](https://github.com/OALabs/frida-extract) based [RunPE](https://github.com/OALabs/frida-extract) extraction tool. Using FridaExtract you can automatically extract and reconstruct a PE file that has been injected using the RunPE method.

https://github.com/OALabs/frida-extract
Tools based on Frida

- **frida-discover**: tool for discovering internal functions in a program. Eg: Cryptoshark: [https://github.com/frida/cryptoshark](https://github.com/frida/cryptoshark)
Tools based on Frida

Cryptoshark and frida-discover are based on Frida’s Stalker API.

They dynamically instrument every thread in a given process and stalk every called function during process execution trying to discover internal functions like statically linked functions.
Conclusions
Conclusions

- When to use **Frida**? If you …

  - Don’t want to download a compiler and compile every time you make a change
  - Need to **quickly** write an introspection tool
  - Need low granularity (this may change in the near future)
  - Need multi-OS/Arch support

- Then …
Conclusions

USE FRIDA

MOTHERFUCKER
Additional information
Additional information

- **Questions** to:
  - [https://twitter.com/oleavr](https://twitter.com/oleavr)

- **Frida** news and docs:
  - [http://www.frida.re/](http://www.frida.re/)

- **Frida** source code:
  - [https://github.com/frida](https://github.com/frida)

- Frida resources:
  - [https://github.com/dweinstein/awesome-frida](https://github.com/dweinstein/awesome-frida)
Acknowledgments & Greetings
Acknowledgments & Greetings

• Ole André V. Ravnås
  • For answering all my question about Frida

• Francisco Falcón
  • For the feedback about this presentation
Questions?
Thank you.