01
Introduction to Dynamic Binary Instrumentation

情報セキュリティ国際会議
CODE BLUE

What is DBI?

The wikipedia definition for instrumentation

“*an ability to monitor or measure the level of a product’s performance, to diagnose errors and to write trace information. Programmers implement instrumentation in the form of code instructions that monitor specific components in a system (…). When an application contains instrumentation code, it can be managed using a management tool. Instrumentation is necessary to review the performance of the application. Instrumentation approaches can be of two types: Source instrumentation and binary instrumentation.*”

What is DBI?

The wikipedia definition for instrumentation

“an ability to monitor or measure the level of a product’s performance, to diagnose errors and to write trace information. Programmers implement instrumentation in the form of code instructions that monitor specific components in a system (…). When an application contains instrumentation code, it can be managed using a management tool. Instrumentation is necessary to review the performance of the application. Instrumentation approaches can be of two types: Source instrumentation and binary instrumentation.”

Too long

Actually too !

What is DBI?

In other words, instrumentation consist of implementing a set of code or instructions into a binary file to gain an insight of it’s behavior during execution.

If we are instrumenting an application, not only can we get information about what is happening but also modify the flow in case that it is needed.

What is DBI?

- There are two possible types of instrumentation
- Binary instrumentation: Interesting to us, allows us to instrument binaries whom code isn’t accessible as long as they run in a system. Despite OSS, lots of software is still closed source.
- Source instrumentation: Add instrumentation code to source code, not feasible if we the source code is not accessible.

What does DBI consist of?

- It’s a technique which consists of injecting instrumentation code that we can in inject in a process allowing us to trace the code **executed**.

  It can be an already running process, a new process or a newly spawned process.

- The instrumentation code is decided by the user.

Types of Analysis

- **Static analysis**: Analyze a binary file without doing nothing more but examining it, that means without executing. Some tools such as IDA, radare2, binja… Allow us to get information about the static properties of a binary.

- **Dynamic analysis**: Consists on running(executing) a sample on an environment which is as close to the reality as possible.

情報セキュリティ国際会議
CODE BLUE

Dynamic Binary Instrumentation

- Instrumenting a binary is not an easy task, it requires complex code and a lot of headaches.

- Luckily, there are frameworks out there that allow users for creating tools with very little effort.

- These frameworks allow us to monitor or modify the behavior of a binary file being executed.

- Although this training is focused on instrumenting Windows malware, DBI can also be used to inspect closed-source software or measure an application's performance.

情報セキュリティ国際会議
CODE BLUE

- Taint analysis
- Fuzzing
- Reverse engineering
- Debugging
- Malware Analysis
- Find vulnerabilities

情報セキュリティ国際会議

CODE BLUE

If we can perform static analysis, why go dyyyyyyyyyynamic?

- Usually, static analysis retrieves very useful information. We can figure out a lot about a binary file by checking it’s disassembly and using powerful reverse engineering tools such as IDA or Radare

- However, at some point we will find binaries that are so huge that performing static analysis won’t be a cost-effective task.

- Static analysis IS boring! (Specially if you are checking disassembly 24 hours 7 days a week)

- Dynamic analysis will allow us to get a quick insight about the behavior of a program, and we can let it take different execution paths to ensure that we are getting as much information as possible.

情報セキュリティ国際会議
CODE BLUE

DBI Frameworks

- As we mentioned before, there are different DBI frameworks that simplify the task of developing instrumentation tools: Intel PIN, Frida, DynamoRIO

- These frameworks provide the user of a relatively simple API to instrument binary files.

- Using these tools don’t require modifying the original binary in any way. A very useful perk, specially if we don’t have source-code access to rebuild the program.

  We will only list the three most popular DBI frameworks out there, however there are other available frameworks out there.

情報セキュリティ国際会議
CODE BLUE

Intel PIN

- PIN is an instrumentation framework created by Intel.
- We can only create tools of PIN for non-commercial purposes, however we are able to test it as long as the PINtool end is not a commercial one
- The source code of Intel’s PIN is closed, sorry!
- It’s a maintained framework, and stable due to it’s long-lasting existence in the DBI frameworks field.
- Limited to Windows, Linux and MacOS

DynamoRIO

- DynamoRIO is an open source framework for RunTime introspection.

- We can inspect code and modify runtime code by creating tools using this framework. Due to its BSD license, it’s possible to create commercial tools.

- Allows the user to create analysis tools for Windows and Linux binaries.

情報セキュリティ国際会議
CODE BLUE

FRIDA

- It’s an open-source DBI framework. Currently sponsored by NowSecure.

- We are not limited to Windows, Linux and MacOS. We can also instrument binaries on Android and iOS platforms.

- We can also instrument code on other architectures aside from x86 and x64, such as ARM and (on the works) MIPS.

- As a major advantage compared to other frameworks, not only it’s easy to use but also it’s blazingly fast to develop tools.

- We can access Frida’s API not only by one or two binding languages, but a lot! The API is available in Python2/3, Swift, C, Java… You can choose the language you wish!

- Mainly used for mobile instrumentation.

- Compared to PIN it’s not as mature, but it being heavily developed gets any problems solved rather quickly

- Easiest to install out of all the 3 mentioned DBI frameworks

情報セキュリティ国際会議
CODE BLUE

What we know until know

- There is no perfect DBI framework, each framework has its pros and cons.
- In case that we want to develop commercial tools (and we don’t have the money for PIN), we can go with DynamoRIO or Frida
- If we want to instrument code fast: Frida wins, but also PIN wouldn’t be slow.
- Easiest one: Frida or PIN
- If we want to instrument mobile platforms: Frida
- Documentation: All of them are fine, Frida still has some documentation issues but are being solved with time. (See new best-practices section)

情報セキュリティ国際会議
CODE BLUE

Frida

Learning about Frida internals

情報セキュリティ国際会議

CODE BLUE

How does it work?

- Woa owa! You have told us many good things about Frida, however we still don’t know how it works.

- No problem! We will get there soon!

Frida internals

The core is written in C, and it injects the V8 Engine into the target process.

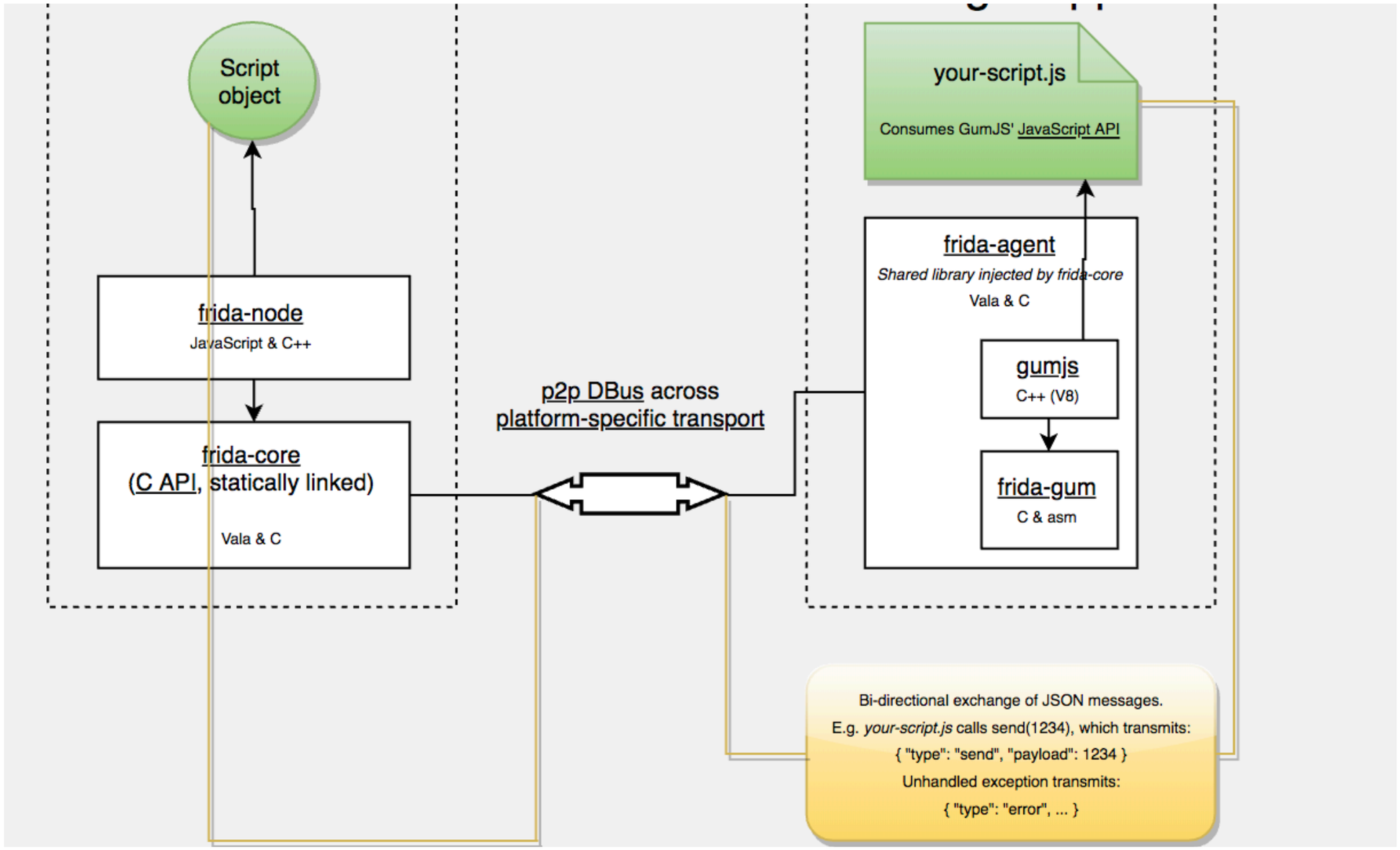
W-w-wait wat? JS??????

The injected JS engine allows for JS code to be executed with full access to memory, calling native functions from inside the process or even hooking functions. This injected JS code will be able to communicate with the tool that you have developed with ease.

If we combine Python and JS, we can build tools way too fast compared to other frameworks.

If you prefer to create your APP in any other language, no worries, you are free to! You can access also the Frida API from C or Swift if you want.

情報セキュリティ国際会議
CODE BLUE



Frida's internals

GumJS

- GumJS allows us to run JavaScript code into a runtime with full access to Gum's APIs. Hence, we are able to hook functions, scan the process memory, read and write memory (read function arguments and return values…).

Modes of operation

- Injected: frida-core acts as a logistics layer that packages GumJS into a shared library. This library is then injected into an existing binary to create a two-way communication channel. It’s also possible to enumerate devices, and running apps.

- Embedded: Some operating system don’t allow us to share, such as non-rooted Android devices or non-jailbroken iPhones. In this situation, Frida-gadget is embedded inside the program that we want to instrument as a shared library.

- Preloaded: We can also preload the shared library similar to the way we do via LD_PRELOAD

情報セキュリティ国際会議
CODE BLUE

Frida-gadget

- It is a shared library whose purpose is to be loaded into other programs that we want to instrument, in case that injecting into the process is not an available option.

- For example, on Android applications we have to rebuild the APK to include this shared library into the application code

- We can also integrate it by modifying the source code, patching one of its libraries, or using dynamic linking such as LD_PRELOAD

情報セキュリティ国際会議
CODE BLUE

Supported operation types

- **Listen** is the default interaction of frida-gadget. It exposes itself the same way Frida-server does.

- For early instrumentation, Gadget’s constructor is blocking until the app performs an attach() call or resume()

- This means that existing tools such as Frida-trace won’t work as intended in this case.

- By using the **Script interaction** It’s also possible to instrument a file before the program;’s entry point is executed.
- Gadget will call the init() method of our script and wait for it to return before the program is able to reach the entrypoint. It also guarantees we won’t miss execution on its early stages.

- By using the ScriptDirectory interaction, we are able to tamper with system-wide software or libraries. This way, we are able to set a directory with scripts and ensure that these scripts are only injected into the applications that we require

- For more detailed information please refer to https://www.frida.re/docs/gadget/

```
'use strict';

rpc.exports = {
  init: function (stage, parameters) {
    console.log('[init]', stage,
JSON.stringify(parameters));

    Interceptor.attach(Module.findExportByName(null,
'open'), {
      onEnter: function (args) {
        var path = Memory.readUtf8String(args[0]);
        console.log('open("' + path + '")');
      }
    });
  },
  dispose: function () {
    console.log('[dispose]');
  }
};
```

Interceptor API

- The interceptor API has different methods: attach, replace…
- With this API we can hook the desired functions provided we know its memory address. (inline hooking)
- When we are attaching to a process, we have access to onEnter and onLeave
- onEnter provides us with access to the hooked functions arguments, and we don’t need to know how many arguments are present.
- If the script is killed, the hooks are reverted from the instrumented sample.

情報セキュリティ国際会議
CODE BLUE

Stalker

- Stalker is an API to do code-tracing

- With Stalker we are not modifying code, and it’s useful when the don’t know what functions are being called.

- Stealthier than the Interceptor APi

情報セキュリティ国際会議
CODE BLUE

Tools built on top of frida

- Frida CLI: It’s a CLI for Frida. We can perform actions directly from the CLI instead of having to develop script. Useful for tests or integration with other tools
- Frida-ps: It lists the current running processes in the system.
- Frida-trace: It traces an already running process or newly spawned process. We can also filter the calls that we want to monitor. In case that we want to hook a group of functions, such as Nt*, a stub file for every sys call will be created. We can then add the remaining code to each file.
- frida-discover: This tool will inspect a running processes threads to retrieve the called functions.
- Frida-ls-devices: This tool will list the available devices on the current system.
- Frida-kill

Useful resources

- https://www.frida.re/docs/hacking/
- https://www.frida.re/docs/gadget/
- https://www.frida.re/docs/frida-cli/
- https://www.frida.re/docs/examples/android/

Recommended resources

- **Building**: https://www.frida.re/docs/building/
- **Troubleshooting**: https://www.frida.re/docs/troubleshooting/
- **Best practices**: https://www.frida.re/docs/best-practices/
- **Javascript API**: https://www.frida.re/docs/javascript-api/

情報セキュリティ国際会議
CODE BLUE