Frida 기본

발표자: 양유진

링크: https://youtu.be/XHUPVvsw6_Q

1. DBI란?

Dynamic Binary Instrumentation

- 바이너리를 동적으로 실행하면서 분석하는 행위/도구를 의미함
- 다양한 DBI 구현 방법이 있음
  - Debugger Script를 이용하는 방법
  - **Hooking을 이용하는 방법**
  - 코드캐쉬와 CPU 가상화를 이용한 에뮬레이팅
- 다양한 DBI Framework 존재
  - Intel PIN, FRIDA, Valgrind 등이 있음

2. Frida란?

- Ole André Vadla Ravnås가 개발한 오픈소스 DBI Framework
  - 무료로 설치 가능하고, 설치 후 실행 단계가 간단함
- Python 기반의 라이브러리로 구성되었으며 다양한 Frida API (**JavaScript**, C, Swift) 제공
- Script와 함께 사용하여 실행 중인 프로그램 동작의 모니터링/수정/기록 가능
  - script 작성 시 다양한 언어를 사용할 수 있으며 주로 Python과 node.js를 많이 사용함
  - 스크립트 언어이기 때문에 PIN과 달리 컴파일 없이 작성/수정 가능하고, 실행 속도가 빠름
- 기본적으로 C/S 구조로 동작하며 내부적으로 동작할 수도 있음
  - 바이너리에 Framework library를 인젝션하여 파이프를 만들어 놓고, 이 파이프를 이용해 명령을 주고 받으며 바이너리 조사 가능

2. Frida란?

- 다양한 운영체제/플랫폼을 지원하여 큰 확장성을 가짐
  - Windows, macOS, Linux, iOS, Android, watchOS 등 지원함 (모바일 앱 분석에 자주 사용하는 걸로 보임)
  - Intel 아키텍처만 지원하는 PIN과 달리 **ARM 아키텍처**를 지원함
- Frida 주요기능
  - 애플리케이션 디버깅 가능
  - 탈옥/루팅되지 않은 단말기에서도 사용 가능
  - 함수 후킹 가능 (함수 재 작성, 특정 함수에 연결하여 반환 값 변경 등)
  - 실시간 트래픽 스니핑 가능
  - 암호 해독 가능

3. 실습 환경 구성 - 우분투 환경 구성

(가상머신 사용시)

- Ubuntu Linux 22.04 LTS
- 가상머신 Oracle Virtual box / VMware 설치

1) frida 설치

```
yj@yj-VirtualBox:~/KCMVP$ pip install frida-tools
```

2) 리눅스 커널 설정 (커널 파라미터 변경)

```
yj@yj-VirtualBox:~/KCMVP$ sudo sysctl kernel.yama.ptrace_scope=0
```

하위 프로세스가 아닌 프로세스도 추적할 수 있게 해줌.

4. Frida 실습1 - 출력 Hooking

hello.c

```c
#include <stdio.h>
#include <unistd.h>

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

int main(int argc, char * argv[]){
    int i=0;

    printf("f() is at %p\n", f);

    while(1){
        f(i++);
        sleep(1);
    }
}
```

1초에 한 번씩 +1 한 숫자 출력하는 코드

4. Frida 실습1 - 출력 Hooking

hook.py

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

session = frida.attach("send_hello")
script = session.create_script("""
Interceptor.attach(ptr("%s"), {
    onEnter: function(args) {
        send(args[0].toInt32());
    }
});
""" % int(sys.argv[1], 16))
def on_message(message, data):
    print(message)
script.on('message', on_message)
script.load()
sys.stdin.read()
```

호출되는 함수를 후킹하여 출력하는 스크립트

4. Frida 실습1 - 출력 Hooking

hook.py

```
from __future__ import print_function
import frida
import sys

session = frida.attach("send_hello")
script = session.create_script("""
Interceptor.attach(ptr("%s"), {
    onEnter: function(args) {
        send(args[0].toInt32());
    }
});
""" % int(sys.argv[1], 16))
def on_message(message, data):
    print(message)
script.on('message', on_message)
script.load()
sys.stdin.read()
```

프로세스 이름

- Frida 시작 및 프로세스 연결
- Frida에서 사용할 수 있도록 script코드 생성

(시작주소, 실행할 실제 코드)
hooking 함수의 시작 주소를 가져와
함수의 인수 출력

- handler 정의 (message 매개 변수 console에 출력)

- Frida script에서 보낸 메시지를 처리할 callback 함수 설정
- 생성한 script 코드 로드
- script가 동작되기 전에 종료되는 문제 예방

4. Frida 실습1 - 출력 Hooking

1) hello.c 출력 코드 컴파일

관련 경고를 모두 출력

```
yj@yj-VirtualBox:~/KCMVP$ gcc -Wall hello.c -o send\_hello
```

2) 컴파일한 파일(send_hello) 실행

```
yj@yj-VirtualBox:~/KCMVP$ ./send_hello
f() is at 0x5627d2307169
Number: 0
Number: 1
Number: 2
```

함수 시작주소

3) 새로운 터미널 열고 후킹 시작

```
yj@yj-VirtualBox:~/KCMVP$ python3 hook.py 0x5627d2307169
{'type': 'send', 'payload': 25}
{'type': 'send', 'payload': 26}
```

4. Frida 실습2 - 함수 인수 변조

modify.py

```
import frida
import sys

session = frida.attach("send_hello")
script = session.create_script("""
Interceptor.attach(ptr("%s"), {
    onEnter: function(args) {
        args[0] = ptr("1998");
    }
});
""" % int(sys.argv[1], 16))
script.load()
sys.stdin.read()
```

(시작주소, 실행할 실제 코드)
함수 인수값 ”1998”로 변조

호출되는 함수의 인수 변조하는

4. Frida 실습2 - 함수 인수 변조

1) send_hello 프로세스 실행하고, 다른 터미널에서 후킹 시작

```
yj@yj-VirtualBox:~/KCMVP$ python3 modify.py 0x556d67773169
```

2) 변조한 인수 출력

```
yj@yj-VirtualBox:~/KCMVP$ ./send_hello
f() is at 0x556d67773169
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: 1998
Number: 1998
```

4. Frida 실습3 - 정수 인젝션 및 함수호출

call.py

```
import frida
import sys

session = frida.attach("send_hello")
script = session.create_script("""
const h = new NativeFunction(ptr("%s"), 'void', ['int']);
h(2000);
h(2000);
h(2000);
""" % int(sys.argv[1], 16))
script.load()
```

- 첫 번째 인수(시작주소)에 위치한 함수를 호출하는 native 함수 생성
- h에 함수가 저장되고, 이는 바로 hello.c의 f()가 됨
- script에서 h()를 호출한 만큼 출력됨

4. Frida 실습3 - 정수 주입 및 함수호출

1) send_hello 프로세스 실행하고, 다른 터미널에서 후킹 시작

```
yj@yj-VirtualBox:~/KCMVP$ ./send_hello
f() is at 0x559f7a35e169
```

```
yj@yj-VirtualBox:~/KCMVP$ python3 call.py 0x559f7a35e169
```

2) 실행결과

```
Number: 13
Number: 14
Number: 15
Number: 2000
Number: 2000
Number: 2000
Number: 16
Number: 17
Number: 18
```

4. Frida 실습4 - 문자열 인젝션 및 함수호출

hello.c

```c
#include <stdio.h>
#include <unistd.h>

int f(const char * s)
{
  printf ("String: %s\n", s);
  return 0;
}

int main(int argc, char * argv[])
{
  const char * s = "Testing!";

  printf ("function f() is at %p\n", f);
  printf ("string s is at %p\n", s);

  while (1)
  {
    f (s);
    sleep (1);
  }
}
```

1초에 한 번씩 문자열 "String: Testing!"을 출력하는 코드

4. Frida 실습4 - 문자열 인젝션 및 함수호출

stringhook.py

```
from __future__ import print_function
import frida
import sys

session = frida.attach("hi")
script = session.create_script("""
const st = Memory.allocUtf8String("TESTMEPLZ!");
const f = new NativeFunction(ptr("%s"), 'int', ['pointer']);
    // In NativeFunction param 2 is the return value type,
    // and param 3 is an array of input types
f(st);
""" % int(sys.argv[1], 16))
def on_message(message, data):
    print(message)
script.on('message', on_message)
script.load()
```

- 메모리에 공간 할당하여 문자열(TESTMEPLZ!)을 저장하고, 이 문자열을 참조하는 NativePointer 객체를 반환함
- 함수를 호출하는 native 함수 생성해서 문자열 주소 할당

4. Frida 실습4 - 문자열 인젝션 및 함수호출

1) hi.c 출력 코드 컴파일

```
yj@yj-VirtualBox:~/KCMVP$ gcc -Wall hi.c -o hi
```

2) 컴파일한 파일(hi) 실행

```
yj@yj-VirtualBox:~/KCMVP$ ./hi
function f() is at 0x558deb8f0169
string s is at 0x558deb8f1010
String: Testing!
String: Testing!
```

3) 새로운 터미널 열고 후킹 시작

```
yj@yj-VirtualBox:~/KCMVP$ python3 stringhook.py 0x55f8eddd0169
```

4) 실행 결과

```
String: Testing!
String: Testing!
String: TESTMEPLZ!
String: Testing!
String: Testing!
```

감사합니다

3. 실습 환경 구성 - macOS 환경 구성

1) homebrew 설치

https://brew.sh/index_ko

2) python 3.x 설치 (python2 사용 불가)

```
% brew install python3
```

3) pip3 version upgrade (23.2.1 가능)

```
% pip3 install --upgrade pip
Successfully installed pip-23.2.1
```

4) frida 설치

```
% pip3 install frida
```